

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Розробка додатку з аналізу рентгенівських знімків та
генерації радіологічних звітів»**

Виконав студент групи ІПЗм-24-1
СКАКОВСЬКИЙ Владислав Олександрович

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
ЧИЖМОТРИЯ Олексій Володимирович

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«18» вересня 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **СКАКОВСЬКИЙ Владислав Олександрович**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Розробка додатку з аналізу рентгенівських знімків та генерації радіологічних звітів»,**

затверджена Наказом закладу вищої освіти від **«18» вересня 2025 р., №491/с**

Вихідні дані для роботи: **веборієнтований додаток для аналізу рентгенівських знімків та генерації радіологічних звітів.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Роман	22.09.2025р.	22.09.2025р.
2	Савіцький Роман	22.09.2025р.	22.09.2025р.
3	Савіцький Роман	22.09.2025р.	22.09.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота магістра складається зі вступу, 3-х розділів, висновків, списку використаних джерел та додатків. Повний обсяг роботи становить 60 сторінок основного тексту, ілюстровано 35 рисунками та 1 таблицею. Список використаних джерел налічує 30 найменувань.

Метою роботи є підвищення ефективності та точності радіологічної діагностики лікарем-радіологом шляхом розробки гібридної програмної системи для автоматизованого аналізу рентгенівських знімків та генерації текстових звітів.

В першому розділі проведено аналіз предметної області та існуючих методів автоматизації в радіології. Обґрунтовано вибір гібридного підходу Concept Bottleneck, який розділяє процес на етапи детекції ознак та генерації тексту, забезпечуючи інтерпретованість результатів.

В другому розділі розроблено метод та спроектовано архітектуру системи. Запропоновано вдосконалений метод візуального аналізу, що дозволяє об'єднувати візуальні дані з клінічними метаданими пацієнта. Розроблено стратегію адаптації великої мовної моделі з використанням методу QLoRA для генерації медичних звітів та спроектовано сервіс-орієнтовану архітектуру.

В третьому розділі здійснено програмну реалізацію компонентів системи. Проведено експериментальне дослідження на датасетах CheXpert та Indiana University Reports. Результати показали високу точність візуальної класифікації та семантичну коректність згенерованих звітів. Виконано оптимізацію моделі до формату Q5_K_M для розгортання на обмежених ресурсах.

В результаті створено діючий прототип вебдодатку з інтерфейсом Human-in-the-loop, який дозволяє лікарю верифікувати автоматично визначені патології перед формуванням фінального звіту, що підвищує надійність системи у клінічному використанні.

КЛЮЧОВІ СЛОВА: РЕНТГЕНОГРАФІЯ, ГЛИБИННЕ НАВЧАННЯ, CONCEPT BOTTLENECK, LATE FUSION, LLM, BIOMISTRAL, QLORA, HUMAN-IN-THE-LOOP.

ABSTRACT

The master's thesis consists of an introduction, 3 chapters, conclusions, a list of references, and appendices. The total volume of the work is 60 pages of the main text, illustrated with 35 figures and 1 table. The list of references includes 30 titles.

The aim of the work is to improve the efficiency and accuracy of radiological diagnostics by a radiologist by developing a hybrid software system for automated analysis of X-ray images and generation of text reports.

In the first chapter, an analysis of the subject area and existing automation methods in radiology was conducted. The choice of the Concept Bottleneck hybrid approach, which divides the process into stages of feature detection and text generation, ensuring the interpretability of results, was justified.

In the second chapter, a method was developed, and the system architecture was designed. An improved method of visual analysis, allowing the combination of visual data with patient clinical metadata, was proposed. A strategy for adapting the large language model using the QLoRA method for generating medical reports was developed and a service-oriented architecture based was designed.

In the third chapter, the software implementation of the system components was carried out. Experimental research was conducted on the CheXpert and Indiana University Reports datasets. The results showed high accuracy of visual classification and semantic correctness of the generated reports. Optimization of the model to the Q5_K_M format for deployment on limited resources was performed.

As a result, a working prototype of a web application with a Human-in-the-loop interface was created, allowing the doctor to verify automatically detected pathologies before generating the final report, which increases the reliability of the system in clinical use.

KEYWORDS: RADIOGRAPHY, DEEP LEARNING, CONCEPT BOTTLENECK, LATE FUSION, LLM, BIOMISTRAL, QLORA, HUMAN-IN-THE-LOOP.

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ...	10
1.1 Аналіз процесу радіологічної діагностики та огляд існуючих методів та рішень	10
1.2 Обґрунтування вибору технологічного підходу.....	14
1.3 Постановка задачі дослідження	16
Висновки до першого розділу	18
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ ТА РОЗРОБКА МЕТОДУ ГЕНЕРАЦІЇ ЗВІТІВ.....	19
2.1 Архітектура програмної системи.....	19
2.2 Розробка методу візуального аналізу та генерації тексту	21
2.3 Проєктування інтерфейсу та API	27
Висновки до другого розділу	29
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ.....	30
3.1 Засоби розробки та підготовка даних.....	30
3.2 Реалізація компонентів системи.....	39
3.3 Експериментальне дослідження якості роботи	50
Висновки до третього розділу	54
ВИСНОВКИ	55
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	56
ДОДАТКИ.....	60

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AUC-ROC - Area Under the Receiver Operating Characteristic Curve

BERT - Bidirectional Encoder Representations from Transformers

BLEU - Bilingual Evaluation Understudy

CBM - Concept Bottleneck Model

CLAHE - Contrast Limited Adaptive Histogram Equalization

CNN - Convolutional Neural Network

DICOM - Digital Imaging and Communications in Medicine

EDA - Exploratory Data Analysis

End-to-End - Наскрізне навчання

ETL - Extract, Transform, Load

GGUF - GPT-Generated Unified Format

Late Fusion - Пізнє злиття

LLM - Large Language Model

LoRA - Low-Rank Adaptation

ML - Machine Learning

MLP - Multilayer Perceptron

NLP - Natural Language Processing

ORM - Object-Relational Mapping

PACS - Picture Archiving and Communication System

PEFT - Parameter-Efficient Fine-Tuning

Q5_K_M - 5-bit quantization with medium K-quants

QLoRA - Quantized Low-Rank Adaptation

RGB - Red, Green, Blue

RIS - Radiology Information System

ROUGE - Recall-Oriented Understudy for Gisting Evaluation

SOA - Service-Oriented Architecture

ViT - Vision Transformer

MIC - Медична інформаційна система

ВСТУП

Актуальність теми. Сучасна медична діагностика генерує величезні обсяги візуальних даних, серед яких рентгенівські знімки грудної клітки займають одне з провідних місць за частотою призначень. Зростання навантаження на лікарів-радіологів призводить до професійного вигорання та збільшення ймовірності діагностичних помилок, особливо при аналізі рутинних випадків або в умовах обмеженого часу.

Існуючі методи автоматизації, що базуються на класичному комп'ютерному зорі (Computer Vision), зазвичай обмежуються класифікацією наявності патології, не надаючи деталізованого опису, необхідного для клінічної практики. З іншого боку, стрімкий розвиток великих мовних моделей (LLM) відкриває нові можливості для генерації природної мови. Проте, пряме застосування генеративних моделей до медичних зображень часто стикається з проблемою «галюцинацій» – генерації правдоподібного, але фактично невірної тексту, що є критичним у медицині.

У зв'язку з цим, актуальним завданням інженерії програмного забезпечення є розробка гібридних інтелектуальних систем, які поєднують точність візуальних класифікаторів із гнучкістю генеративних мовних моделей, забезпечуючи при цьому контрольованість та інтерпретованість результату.

Мета і задачі дослідження. Метою роботи є підвищення ефективності роботи лікаря-радіолога шляхом розробки програмної системи для автоматизованої генерації попередніх текстових звітів рентгенівських знімків, що забезпечує високу узгодженість між візуальними ознаками та текстовим описом.

Для досягнення поставленої мети необхідно вирішити такі задачі:

1. Провести аналіз предметної області та існуючих методів автоматичного опису медичних зображень (Image Captioning, Radiology Report Generation).

2. Розробити метод двоетапної генерації звітів, що включає проміжний етап виділення клінічних сутностей (тегів) для мінімізації фактичних помилок генерації.

3. Спроектувати сервіс-орієнтовану архітектуру програмної системи, що включає вебсервер та окремий модуль інференсу моделей машинного навчання.

4. Виконати адаптацію та навчання моделі класифікації зображень (EfficientNet-B0 з механізмом Late Fusion) та донавчання (Fine-tuning) великої мовної моделі (BioMistral-7B) на спеціалізованих радіологічних датасетах.

5. Здійснити програмну реалізацію системи у вигляді вебзастосунку з використанням фреймворків Django та BentoML.

6. Провести експериментальне дослідження розробленої системи, оцінити якість класифікації та генерації тексту за допомогою автоматичних метрик (AUC-ROC, BLEU, BERT Score).

Об’єкт дослідження – процес автоматизованого аналізу медичних діагностичних зображень та генерації опису природною мовою.

Предмет дослідження – методи та алгоритми побудови гібридних систем машинного навчання (CNN + LLM) для формування структурованих радіологічних звітів.

Методи дослідження. У роботі використано методи:

- системного аналізу – для визначення вимог до програмного продукту;
- глибинного навчання (Deep Learning) – для розпізнавання образів (згорткові нейронні мережі, Late Fusion) та обробки природної мови (Transformer-based LLM);
- програмної інженерії – для проектування архітектури, розробки API та клієнтської частини;
- математичної статистики – для оцінки якості роботи моделей.

Наукова новизна одержаних результатів.

Удосконалено метод генерації радіологічних звітів шляхом впровадження архітектури з «інформаційним вузьким горлом» (Concept Bottleneck), де проміжним представленням виступає структурований набір клінічних тегів, що, на відміну від класичних End-to-End підходів, дозволяє підвищити фактичну точність згенерованого тексту та забезпечує прозорість прийняття рішень системою.

Набув подальшого розвитку метод візуального аналізу медичних зображень через застосування архітектури пізнього злиття (Late Fusion), що дозволяє об'єднувати візуальні ознаки знімка з клінічними метаданими (вік, стать) для підвищення точності діагностики.

Набув подальшого розвитку підхід до спеціалізації великих мовних моделей для вузькопрофільних медичних задач через використання технік ефективного донавчання (QLoRA) та стратегій декодування для мінімізації галюцинацій.

Практичне значення одержаних результатів. Результатом роботи є діючий прототип програмної системи, який дозволяє завантажувати рентгенівські знімки формату DICOM/PNG, автоматично визначати патології та генерувати проєкт текстового звіту. Розроблена система може бути використана як асистент лікаря-радіолога для скорочення часу на рутинний опис знімків («second opinion»).

Розроблене програмне забезпечення реалізовано мовою Python з використанням фреймворків PyTorch, Django та BentoML, що забезпечує масштабованість та можливість легкої інтеграції в існуючі медичні інформаційні системи (MIS). Оптимізація моделі до формату Q5_K_M дозволяє розгортати систему на обладнанні з обмеженими ресурсами.

Публікації. За темою атестаційної магістерської роботи було опубліковано статтю та тези [1-2].

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз процесу радіологічної діагностики та огляд існуючих методів та рішень

Процес радіологічної діагностики є критично важливою складовою сучасної доказової медицини, що забезпечує неінвазивну візуалізацію внутрішніх анатомічних структур для виявлення патологічних змін. З точки зору інженерії програмного забезпечення, автоматизація цього процесу вимагає глибокого розуміння клінічного робочого потоку (workflow), специфіки медичних даних та структури прийняття рішень лікарем. Дослідження вказують на зростання навантаження на радіологів, що призводить до збільшення ризику діагностичних помилок [3].

Традиційний сценарій аналізу рентгенівських знімків починається з отримання зображення та його передачі до системи архівування та передачі зображень (PACS – Picture Archiving and Communication System). Лікар-радіолог аналізує візуальні дані на діагностичних моніторах високої роздільної здатності, оцінюючи якість знімка, проєкцію та експозицію, після чого переходить до пошуку візуальних аномалій. Завершальним етапом є формування текстового звіту, який зберігається в радіологічній інформаційній системі (RIS) та стає юридичним документом, що обґрунтовує подальше лікування.

Основним стандартом обміну візуальними даними в цьому процесі є DICOM (Digital Imaging and Communications in Medicine). Важливою особливістю для розробки автоматизованої системи є те, що DICOM-файл являє собою не просто растрове зображення, а складний об'єкт, який містить критично важливі метадані, такі як ідентифікатори пацієнта, параметри дослідження, розмірність матриці та бітову глибину. Текстові дані та звіти, в свою чергу, часто передаються та зберігаються згідно зі стандартами HL7 (Health Level 7) або FHIR, що накладає вимоги на формат вихідних даних проєктованої системи [4]. Сучасні дослідження наголошують на важливості інтеграції мультимодальних даних (зображення + текст + метадані) для підвищення точності діагностики [5].

Кінцевим продуктом роботи радіолога є структурований звіт, а не просто набір міток. Типовий протокол опису рентгенографії грудної клітки складається з кількох логічних блоків. Першим є «Показання» (Indication), що описує клінічну причину обстеження, наприклад, скарги на кашель або підозру на пневмонію; ця інформація слугує важливим контекстом для інтерпретації знімка. Основною частиною є секція «Опис» або «Знахідки» (Findings), де детально фіксуються спостереження щодо стану легеневих полів, плевральних синусів, тіні серця, середостіння та кісткового каркаса. Завершується звіт секцією «Висновок» (Impression), яка містить стисле резюме виявлених патологій або диференційний діагноз.

При розробці алгоритмів комп'ютерного зору для цього завдання необхідно враховувати візуальні характеристики основних патологій. Більшість діагностично значущих ознак на рентгенограмі можна класифікувати за типом зміни прозорості легеневої тканини. Наприклад, пневмонії та ателектази проявляються як ділянки затінення (консолідації) різної інтенсивності та локалізації. Плевральний випіт характеризується накопиченням рідини, що візуально згладжує реберно-діафрагмальні кути. Кардіомегалія визначається через вимірювання кардіоторакального індексу (збільшення поперечного розміру серця). Пневмоторакс, навпаки, проявляється як ділянка підвищеної прозорості без легеневого малюнка, відокремлена вісцеральною плеврою.

Отже, задача автоматизації радіологічного опису є мультимодальною проблемою, що вимагає не лише детекції наявності патології, але й її коректної локалізації та опису природною мовою. Висока ціна помилки першого роду (пропуск патології) вимагає застосування архітектурних рішень, які забезпечують інтерпретованість результатів, дозволяючи лікарю верифікувати проміжні висновки моделі перед фіналізацією звіту. Аналіз даних CheXpert показав значну мультимітковість (більшість пацієнтів мають кілька патологій одночасно), що вимагає використання спеціалізованих методів класифікації [6].

Фундаментом будь-якої системи аналізу зображень є візуальний енкодер – компонент, що перетворює вхідне зображення у векторне представлення або

карту ознак. У медичній сфері довгий час домінували глибокі згорткові нейронні мережі (CNN). Однією з найбільш впливових архітектур стала ResNet (Residual Networks), яка завдяки використанню skip-connections дозволила ефективно тренувати мережі глибиною в сотні шарів. Однак у задачах аналізу рентгенограм, де патологічні зміни можуть бути візуально тонкими та займати незначну частину зображення, кращі результати продемонструвала архітектура DenseNet (Densely Connected Convolutional Networks) [7]. У цій архітектурі кожен шар отримує карти ознак від усіх попередніх шарів, що забезпечує максимальний потік інформації та ефективне використання параметрів. Зокрема, модель DenseNet-121 стала своєрідним стандартом (baseline) для задач мультикласової класифікації патологій грудної клітки [8]. Сучасні роботи також демонструють ефективність архітектури EfficientNet, яка використовує метод складеного масштабування (Compound Scaling) для досягнення кращого балансу між точністю та обчислювальною ефективністю, що є важливим для впровадження в клінічну практику [9].

З 2020 року набули поширення візуальні трансформери (Vision Transformers – ViT), які відмовилися від операції згортки на користь механізму саморозуміння (Self-Attention). ViT розбиває зображення на патчі та аналізує глобальні залежності між ними [10]. Хоча ViT демонструють кращу стійкість до шумів, їх використання в медицині обмежене потребою у надзвичайно великих обсягах даних для попереднього навчання, тому гібридні підходи або CNN залишаються актуальними для задач з обмеженою вибіркою. Важливим напрямком є також використання методів злиття даних, які дозволяють інтегрувати візуальну інформацію з клінічними метаданими пацієнта (вік, стать), підвищуючи точність діагностики вікозалежних патологій [11].

На ранніх етапах розвитку автоматизованої генерації звітів домінували два підходи: шаблонний (Template-based) та пошуковий (Retrieval-based). Перший метод обмежувався механічним заповненням змінних у фіксованих структурах тексту, що забезпечувало синтаксичну коректність, проте суттєво обмежувало інформативність висновку. Другий підхід базувався на знаходженні візуально

схожого архівного знімка та дублюванні його опису. Це створювало значні ризики для пацієнта, оскільки ігнорування індивідуальних клінічних особливостей могло призвести до критичних діагностичних помилок.

Прорив відбувся з появою генеративних нейронних мереж, які працюють за принципом Encoder-Decoder. У класичній реалізації CNN-енкодер перетворює зображення на вектор контексту, а рекурентний декодер (RNN, LSTM або GRU) генерує текст слово за словом. Головним недоліком таких систем була схильність до генерації повторюваних, безпечних фраз («Легені чисті», «Серце не розширене») незалежно від вмісту зображення, а також проблема «забування» контексту при генерації довгих звітів. Для подолання цих обмежень було впроваджено механізми візуальної уваги (Visual Attention). Вони дозволяють моделі фокусуватися на відповідних ділянках зображення під час генерації кожного конкретного слова.

Сучасний етап розвитку пов'язаний із використанням архітектури Transformer та великих мовних моделей. Мульти模альні моделі, такі як CLIP або LLaVA, навчаються зіставляти зображення і текст у єдиному семантичному просторі. Проте застосування універсальних LLM (наприклад, GPT-4 або Llama) у медицині стикається з критичною проблемою – «галюцинаціями». Модель, навчена генерувати найбільш ймовірне продовження тексту, може переконливо описати патологію, якої насправді немає на знімку, або пропустити критичну знахідку [12]. Використання спеціалізованих медичних моделей, таких як BioMistral, у поєднанні з методами ефективного донавчання (QLoRA), дозволяє зменшити цей ризик та підвищити точність генерації [13].

У відповідь на це виник новий клас архітектур – Concept Bottleneck Models (CBM) [14]. Ідея полягає в тому, щоб не генерувати текст безпосередньо з зображення (End-to-End), а розділити процес на два етапи:

1. Передбачення проміжного набору клінічно значущих понять або тегів (наприклад, «наявна консолідація», «відсутній пневмоторакс»).
2. Генерація тексту звіту виключно на основі цих передбачених понять.

Такий підхід створює «інформаційне вузьке горло» (bottleneck), яке фільтрує шум і змушує модель обґрунтовувати свій висновок конкретними знахідками. Це дозволяє лікарю верифікувати роботу системи на проміжному етапі (перевірити список тегів) перед генерацією фінального тексту, що значно підвищує довіру до автоматизованої системи та її безпеку в клінічному використанні. Саме цей гібридний підхід покладено в основу даної магістерської роботи.

1.2 Обґрунтування вибору технологічного підходу

Вибір архітектури програмної системи для медичної діагностики не може базуватися виключно на метриках точності моделі. Критичними факторами стають інтерпретованість результатів, стійкість до помилок (robustness) та можливість верифікації рішень. На основі аналізу, проведеного в попередніх підрозділах, для реалізації дипломної роботи обрано двоетапний гібридний підхід (Pipeline Approach) із використанням концепції Concept Bottleneck. Нижче наведено детальне обґрунтування цього вибору через порівняння з альтернативними рішеннями.

Глобально існуючі методи класифікують на дві основні категорії: монолітні моделі (End-to-End) та компонентні системи (Pipeline). Підхід End-to-End працює за принципом «чорної скриньки», де нейронна мережа отримує зображення і безпосередньо генерує текст. Натомість у Pipeline-системах процес генерації є поетапним і включає виокремлення ознак, формування проміжного представлення даних та подальше створення текстового опису.

Використання End-to-End моделей (наприклад, на базі архітектури Transformer, що з'єднує візуальний енкодер і текстовий декодер) має суттєвий недолік у медичному домені – неконтрольованість генерації. Такі моделі схильні до так званих «галюцинацій», коли система генерує правдоподібний опис патології, яка відсутня на знімку, лише тому, що ця патологія часто зустрічалася у навчальній вибірці поруч із іншими, реально наявними ознаками [15]. У монолітній архітектурі розробнику вкрай важко відстежити, що саме стало

тригером для генерації конкретного речення, що знижує довіру до системи з боку лікарів [16].

Перевага Pipeline-підходу полягає у розмежуванні сфер відповідальності. Один модуль фокусується виключно на комп'ютерному зорі та пошуку аномалій, в той час як інший лише формулює опис. Такий поділ дозволяє точно локалізувати помилки. Якщо система пропустила діагноз, розробник може легко перевірити чи це збій на етапі розпізнавання, чи некоректна обробка вхідних даних мовною моделлю.

Ключовою особливістю обраного підходу Concept Bottleneck є впровадження інформаційного обмеження у вигляді клінічних тегів (Concepts) [17]. Замість передачі неінтерпретованого вектора ознак (embedding) від зображення до мовної моделі, система передає явний список знайдених класів (наприклад: ['Atelectasis: Positive', 'Pleural Effusion: Negative']).

Такий підхід забезпечує:

- фактологічну точність (Factuality);
- human-in-the-loop;
- юридичну прозорість.

Велика мовна модель виступає у ролі інтерпретатора структурованих даних у природну мову, а не генератора ідей. Це мінімізує ризик вигадування неіснуючих хвороб [18].

Проміжний етап (список тегів) є зрозумілим для людини. Лікар може переглянути список знайдених патологій та скоригувати його (додати або видалити тег) ще до генерації тексту звіту. Це перетворює систему з «автопілота» на керованого асистента. У разі лікарської помилки можна чітко встановити, на якому етапі алгоритм надав некоректну інформацію.

З погляду програмної інженерії, ключовою перевагою обраної архітектури є реалізація принципу слабкої зв'язаності (Loose Coupling). Це дає розробникам гнучкість у модернізації системи, дозволяючи оновлювати компоненти незалежно одне від одного. Наприклад, можна безболісно замінити візуальний класифікатор на більш сучасний аналог, скажімо, перейшовши з DenseNet на

EfficientNet, і це не вимагатиме повного перенавчання мовної моделі чи зміни логіки її роботи.

Економічна ефективність навчання також є вагомим аргументом на користь цього підходу. Тренування окремого класифікатора вимагає значно менше ресурсів, ніж навчання громіздкої мультимодальної моделі з нуля. Те саме стосується і мовного модуля: донавчання (Fine-tuning) LLM для перетворення набору тегів у текст є обчислювально дешевшою операцією порівняно зі складними наскрізними методами, про що свідчать дослідження [19].

Крім того, така модульність суттєво спрощує контроль якості та тестування. Розділення відповідальності дозволяє писати ізольовані Unit-тести для кожного етапу: окремо перевіряти метрики точності (Accuracy/AUC) для класифікатора та окремо оцінювати якість згенерованого тексту на основі фіксованого набору тегів, не змішуючи можливі помилки різних підсистем.

Таким чином, вибір гібридної архітектури з проміжним представленням класів є найбільш обґрунтованим для розробки надійної медичної інформаційної системи, оскільки він балансує між високою якістю генерації тексту сучасними LLM та суворими вимогами до точності та інтерпретованості медичної діагностики.

1.3 Постановка задачі дослідження

На основі проведеного аналізу предметної області та вибору технологічного підходу, можна сформулювати задачу дослідження як розробку програмної системи для автоматизованого формування радіологічних висновків, що мінімізує розходження між візуальними ознаками на знімку та їх текстовим описом.

Нехай вхідними даними є цифрове зображення рентгенограми грудної клітки, представлене у вигляді багатовимірної матриці пікселів. Цільовим результатом є текстовий опис (радіологічний звіт), що складається з послідовності слів (токенів) із заданого медичного словника.

Класична постановка задачі генерації звітів (End-to-End) полягає у знаходженні такого відображення вхідного зображення у текст, яке максимізує

ймовірність генерації правильної послідовності слів, враховуючи контекст зображення та всі попередні згенеровані слова.

У даній роботі, з метою підвищення інтерпретованості та фактологічної точності, задача декомпозується на два послідовні етапи з використанням проміжного простору клінічних понять (тегів), де кожне поняття відповідає певній патології (наприклад, «Пневмонія», «Набряк»).

Першим етапом є побудова класифікатора, який для вхідного зображення повертає вектор ймовірностей наявності кожної з відстежуваних патологій. Навчання цього модуля відбувається шляхом мінімізації функції втрат (наприклад, бінарної перехресної ентропії), що вимірює розбіжність між передбаченою ймовірністю та реальним діагнозом для кожного класу незалежно.

Другим етапом є побудова генеративної моделі, яка створює текст звіту не на основі «сирих» пікселів зображення, а виключно на основі бінаризованого вектора виявлених патологій (тегів). Задача полягає у знаходженні такої послідовності слів, яка є найбільш статистично ймовірною для даного набору клінічних знахідок при фіксованих параметрах мовної моделі.

Таким чином, загальна задача зводиться до навчання параметрів візуального енкодера для точного розпізнавання хвороб та адаптації параметрів великої мовної моделі для коректного перекладу набору діагнозів у зв'язний природно-мовний текст.

Для практичної реалізації розробленого методу система повинна задовольняти наступним вимогам:

Функціональні вимоги:

- система повинна приймати на вхід зображення у форматі PNG/JPG (для демонстраційних цілей);
- система має автоматично визначати наявність не менше 10 основних патологій грудної клітки (Пневмонія, Ателектаз, Кардіомегалія, Плевральний випіт та ін.);
- користувач (лікар) повинен мати можливість переглянути та відредагувати список виявлених патологій перед генерацією тексту;

- генерація тексту має відбуватися у форматі структурованого звіту («Findings», «Impression»).

Нефункціональні вимоги:

- середнє значення метрики AUC-ROC для класифікатора має бути не нижче 0.70;
- час обробки одного знімка (від завантаження до отримання тексту) не повинен перевищувати 5-10 секунд на сучасному GPU.

Висновки до першого розділу

У першому розділі магістерської роботи проаналізовано проблему автоматизації радіологічних описів та встановлено, що критичною вимогою до таких систем є не лише точність, а й інтерпретованість результатів. Огляд існуючих методів показав, що поширені End-to-End моделі схильні до генерації неправдивих медичних даних («галюцинацій»), що унеможлиблює їх безпечне використання в клінічній практиці без додаткового контролю.

На основі порівняльного аналізу було обрано гібридний технологічний підхід Concept Bottleneck Models. Він передбачає розділення задачі на два етапи: детекцію клінічних ознак (тегів) та генерацію тексту на їх основі. Таке архітектурне рішення забезпечує прозорість прийняття рішень, дозволяє реалізувати механізм верифікації лікарем (Human-in-the-loop) та відповідає принципам модульності програмної інженерії.

У підсумку сформульовано задачу дослідження як розробку програмної системи, що складається з модуля візуальної класифікації та спеціалізованої великої мовної моделі. Визначено ключові вимоги до продукту, такі як точність класифікації за метрикою AUC-ROC не нижче 0.70 та мінімізація фактичних помилок у тексті. Отримані результати є теоретичним підґрунтям для подальшого проєктування архітектури системи у другому розділі.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ ТА РОЗРОБКА МЕТОДУ ГЕНЕРАЦІЇ ЗВІТІВ

2.1 Архітектура програмної системи

Проектування архітектури програмного забезпечення визначає структуру компонентів та їх взаємодію. Для реалізації системи обрано сервіс-орієнтовану архітектуру (SOA), де вебдодаток та модуль машинного навчання фізично та логічно розділені. Такий підхід вирішує проблему конфлікту залежностей та масштабування: вебсервер на базі Django обробляє користувацькі запити, а спеціалізований фреймворк BentoML обслуговує моделі штучного інтелекту [20].

Система складається з двох основних вузлів, що взаємодіють через мережу: вебсервісу (Django Host) та ML-сервісу (BentoML Service).

Вебсервіс поєднує функції інтерфейсу та оркестратора процесів. Архітектурно він побудований навколо моделі XRayReport, яка виступає ядром системи. Вона акумулює в собі весь контекст поточної діагностичної сесії та зв'язує дані з їх візуальним представленням.

Критично важливим аспектом моделі даних є зберігання клінічних метаданих пацієнта, які використовуються для мультимодального аналізу:

- patient_age (вік пацієнта);
- patient_sex (стать (Чоловіча/Жіноча));
- image (файл рентгенівського знімка);
- findings (JSON-результати роботи класифікатора).

Бізнес-логіка розподілена між контролерами (Views). Зокрема, upload_xray вимагає заповнення форми з демографічними даними перед завантаженням файлу.

ML-сервіс реалізований як окремий високопродуктивний сервер для інференсу моделей. Він використовує абстракцію Runner для керування завантаженням ваг моделей у пам'ять та механізм Adaptive Batching для оптимізації ресурсів. Сервіс надає відкриті API endpoints: /analyze_vision, який приймає тензор зображення та метаданих та повертає ймовірності класів (Visual

Extractor), та /generate_report, який приймає список тегів і повертає згенерований текст (LLM).

Взаємодія між Django та BentoML відбувається за протоколом HTTP (REST). Використання BentoML дозволяє інкапсулювати складну логіку препроцесингу зображень всередині ML-сервісу. Процес інтеграції передбачає, що Django отримує зображення у контролері upload_xray, зберігає його в об'єкт XrayReport і передає у BentoML. Останній виконує трансформацію зображення (зміна розміру, нормалізація), проганяє його через нейромережу, а результат повертає у форматі JSON, який зберігається у полі findings моделі.

Специфікація API базується на двох ключових POST-запитах. Перший метод звертається до адреси /analyze_vision та приймає зображення разом із метаданими, повертаючи у відповідь словник ймовірностей патологій. Другий запит спрямовується на ендпоінт /generate_report, де на основі вхідного JSON зі списком тегів формується готовий текстовий рядок звіту.

Процес обробки запиту реалізує концепцію Human-in-the-loop з мережевою взаємодією [21]. Життєвий цикл обробки одного знімка можна розділити на два етапи: візуальний аналіз та генерацію тексту (рис. 2.1).

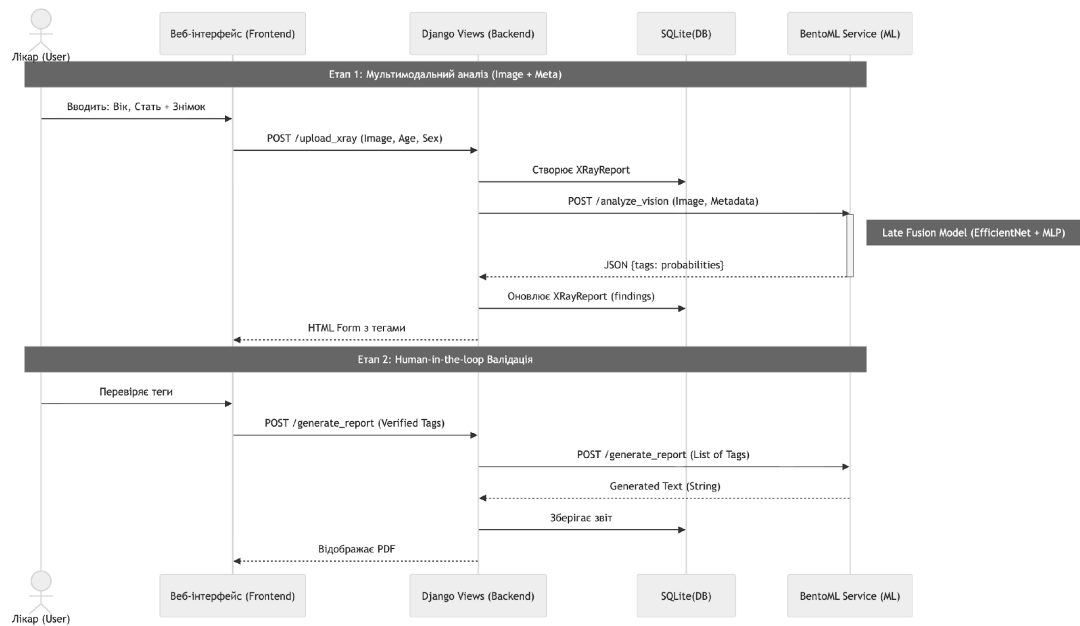


Рис. 2.1 – Діаграма потоків даних

Спочатку лікар на сторінці завантаження вводить вік та стать пацієнта та відправляє рентгенівський знімок. Вебсервер Django зберігає файл та робить синхронний запит до ендпоінту аналізу зображення сервісу BentoML. Отримані автоматичні теги записуються в базу даних. На наступному кроці користувач перенаправляється на сторінку верифікації, де бачить визначені патології та може змінити їх статус. Після підтвердження, верифіковані дані передаються на сервер, який формує запит до мовної моделі BentoML. Отриманий текст фіналізується, зберігається в базі даних та відображається лікарю у вигляді готового звіту.

Така архітектура дозволяє ефективно розподілити навантаження, розгортаючи ML-частину на окремому сервері з GPU, тоді як вебсервер може функціонувати на менш потужному обладнанні.

2.2 Розробка методу візуального аналізу та генерації тексту

Модуль візуального аналізу є першою ланкою в конвеєрі генерації звітів. Традиційні підходи комп'ютерного зору аналізують лише піксельні дані, ігноруючи клінічний контекст. Однак у радіології інтерпретація знімка часто залежить від демографічних показників: наприклад, прозорість легеневої тканини у 20-річного та 80-річного пацієнта відрізняється, і те, що є нормою для літньої людини, може бути патологією для молодшої [22].

Для вирішення цієї проблеми розроблено метод Мультимодального пізнього злиття (Multi-modal Late Fusion), який дозволяє нейронній мережі приймати рішення на основі одночасної обробки зображення та табличних даних (вік, стать) [23].

Замість використання стандартної згорткової нейронної мережі, розроблено композитну архітектуру, що складається з двох гілок обробки даних та блоку злиття (рис. 2.2).

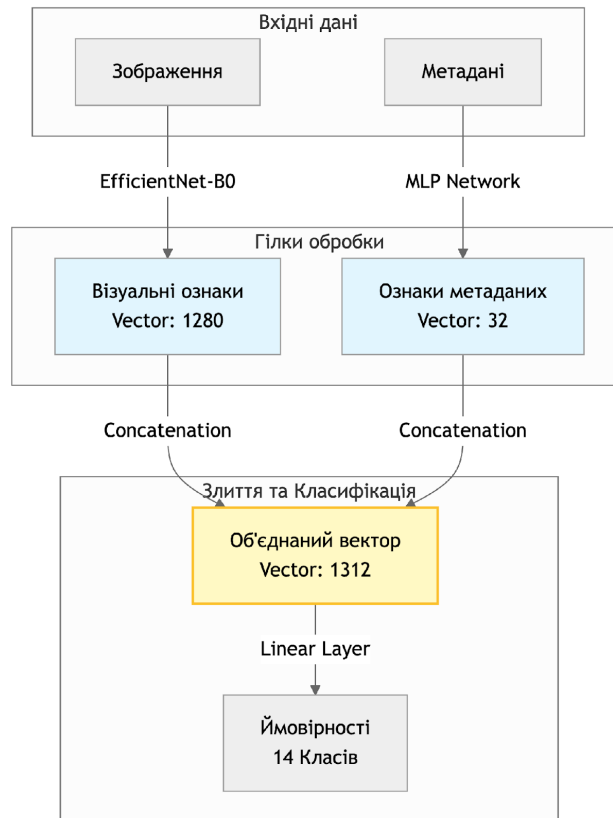


Рис. 2.2 – Діаграма загальної архітектури методу візуального аналізу

Візуальна гілка (Image Branch) базується на архітектурі EfficientNet-B0 з використанням бібліотеки `timm`. Для отримання векторного представлення зображення параметр `num_classes` встановлено в нуль, що дозволяє вилючити з архітектури фінальний класифікаційний шар та отримати початковий вектор ознак (embedding) розмірністю $D_{img} = 1280$. На вхід гілки подається тензор зображення розміром $(B, 3, 240, 240)$. Така конфігурація якнайкраще ілюструє головну перевагу цієї моделі, такою як оптимальний баланс між точністю та обчислювальними витратами [24].

Гілка метаданих (Metadata Branch) призначена для обробки нормалізованих значень віку та закодованої статі. Вона реалізована як невеликий багатошаровий перцептрон (MLP), що приймає на вхід вектор розмірності 2. Структура цієї підмережі включає лінійний шар Linear $(2 \rightarrow 32)$, нормалізацію BatchNorm1d, функцію активації ReLU та шар регуляризації Dropout(0.1). Виходом гілки є вектор ознак метаданих розмірністю $D_{meta} = 32$.

Завершальним етапом архітектури є блок злиття (Fusion Block) та класифікації. На цьому кроці вектори ознак з обох гілок об'єднуються в єдине представлення шляхом операції конкатенації, як представлено у виразі (2.1):

$$f_{combined} = Concat(f_{img}, f_{meta}) \in R^{1312} \quad (2.1)$$

- f_{img} – вектор візуальних ознак розмірністю 1280;
- f_{meta} – вектор табличних ознак (метаданих) розмірністю 32;
- $f_{combined}$ – результуючий об'єднаний вектор.

Підсумкова розмірність 1312 є сумою розмірностей візуальних (1280) та табличних (32) ознак. Отриманий об'єднаний вектор подається на фінальний повнозв'язний шар Linear (1312 $\rightarrow$ 14), який трансформує його у ймовірності наявності 14 цільових патологій.

Для покращення якості візуальних ознак розроблено пайплайн препроцесингу, який розпочинається з компенсації низького локального контрасту знімків. Оскільки це ускладнює виявлення патологій у затінених зонах, застосовується метод Contrast Limited Adaptive Histogram Equalization (CLAHE) [25]. Обробка виконується в колірному просторі YCbCr, де еквалізується лише канал яскравості (Y), після чого зображення конвертується назад. Це дозволяє підкреслити деталі кісткової тканини та легеневого малюнка без створення шумів.

Оскільки EfficientNet попередньо навчена на ImageNet (кольорові фото), одноканальні рентгени конвертуються в 3-канальні шляхом дублювання каналу яскравості. Це забезпечує сумісність з вагами першого згорткового шару.

Зображення масштабуються до 240×240 пікселів з використанням білінійної інтерполяції. Цей розмір є оптимальним компромісом: він трохи більший за стандартний вхід EfficientNet-B0 (224x224), що дозволяє зберегти дрібні деталі, але не створює надмірного навантаження на GPU.

Значення пікселів приводяться до діапазону [0, 1] та стандартизуються з використанням середнього та відхилення ImageNet.

Критичним етапом для мультимодального навчання є правильна підготовка табличних даних, оскільки нейронні мережі чутливі до масштабу

вхідних значень. Процес перетворення клінічних даних у тензор зображено на діаграмі рис. 2.3.

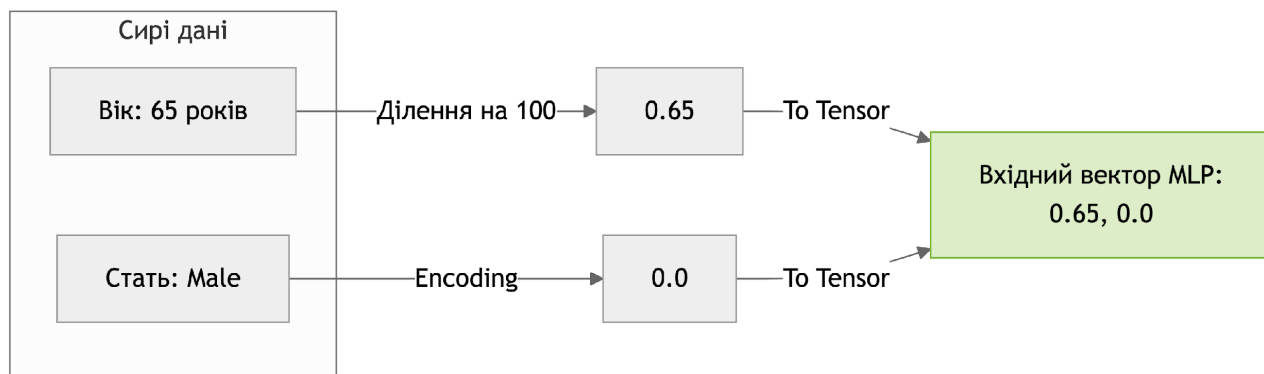


Рис. 2.3 – Перетворення клінічних даних у тензор

Сирі значення віку (наприклад, «65» або «25») діляться на 100. Це приводить діапазон значень до інтервалу $[0.0, 1.0]$, що узгоджується з діапазоном значень після нормалізації зображення. Пропущені значення (NaN) заповнюються медіаною.

Категоріальна змінна статі кодується числовими значеннями:

- male = 0.0;
- female = 1.0;
- unknown = 0.5.

Зображення обробляються за таким стандартним пайплайном, як конвертація в RGB, зміна розміру до 240×240 пікселів та нормалізація статистиками ImageNet. Такий підхід дозволяє моделі вивчати складні залежності, наприклад: «Розширення тіні серця (Cardiomegaly) є більш імовірним для пацієнтів старшого віку», або «Наявність сторонніх предметів (Support Devices) корелює з важкістю стану». Це значно підвищує точність класифікації в неоднозначних випадках.

Другим ключовим компонентом гібридної системи є модуль генерації тексту (Text Generator). Його вхідними даними є верифікований лікарем набір клінічних тегів (результат роботи Visual Extractor та етапу Human-in-the-loop), а вихідними – зв'язний, граматично правильний текст радіологічного звіту, що

відповідає медичним стандартам. Задача формулюється як Conditional Text Generation (умовна генерація тексту).

Для реалізації генеративного компонента проаналізовано сучасні відкриті великі мовні моделі. Використання пропрієтарних API (таких як GPT-4) було відхилено з міркувань безпеки даних пацієнтів та економічної доцільності. Серед Open-Source рішень обрано модель BioMistral-7B [26].

BioMistral-7B є адаптацією Mistral-7B, попередньо навченою на великому корпусі медичних текстів (PubMed Central, медичні підручники, клінічні звіти), що забезпечує краще розуміння медичної термінології порівняно з загальними моделями.

Основою текстової генерації обрано декодерний трансформер (Decoder-only Transformer) із 7 мільярдами параметрів, який завдяки механізму Grouped Query Attention (GQA) забезпечує високу швидкість інференсу без втрати якості. Такий розмір моделі є оптимальним компромісом, що дозволяє запускати систему навіть на споживчих GPU, а ліцензія Apache 2.0 надає можливість для її вільної модифікації та комерційного використання. Оскільки архітектура не є мультимодальною, взаємодія з візуальними даними реалізується через спеціально розроблений структурований промпт, який серіалізує всю необхідну інформацію для генерації звіту.

Структура промпта:

- system instruction (системна інструкція), чітке визначення задачі генерації професійного радіологічного звіту;
- input context (вхідні дані), список підтверджених патологій;
- output structure (структура виходу), явна вказівка на необхідність розділів «Findings» та «Impression». Така структура мінімізує ризик галюцинацій та забезпечує відповідність стандартам медичної документації [27].

Попри спеціалізацію BioMistral на медичній літературі, модель потребує адаптації до специфічного стилю написання радіологічних звітів на датасеті Indiana University Chest X-Ray Reports. Повне донавчання 7 млрд параметрів

вимагає значних обчислювальних ресурсів. Для ефективної адаптації використано метод QLoRA [28].

Суть методу Low-Rank Adaptation (LoRA) полягає у фіксації всіх ваг попередньо навченої моделі. Натомість, до шарів механізму уваги (зокрема, проєкцій Query та Value) паралельно додаються пари матриць низького рангу A та B , які підлягають навчанню.

Формально процес прямого проходу з адаптованими вагами описується виразом (2.2):

$$W' = W_0 + \Delta W = W_0 + B \cdot A \quad (2.2)$$

де:

- $W_0 \in R^{d \times d}$ – фіксовані ваги попередньо навченої моделі;
- ΔW – матриця адаптації, яка розкладається на добуток двох матриць меншого розміру;
- $A \in R^{r \times d}$ та $B \in R^{d \times r}$ – матриці низького рангу;
- r – гіперпараметр рангу, причому $r \ll d$ (наприклад, $r = 16$).

Для зменшення вимог до відеопам'яті застосовано 4-бітне квантування (quantization).

Для генерації фінального тексту застосовано знижений коефіцієнт температури ($T=0.5$), який наближає розподіл ймовірностей до детермінованого стану, надаючи перевагу фактологічній точності над творчою варіативністю. Аби уникнути зациклених фраз, часто притаманних медичним звітам, використано механізм штрафу за повторення (Repetition Penalty) зі значенням $\lambda=1.2$. Додатково для виключення дублювання цілих фрагментів тексту введено жорстку заборону на повторення послідовностей із трьох і більше слів (No-Repeat N-gram size = 3).

Для впровадження у продакшн фінальна модель проходить етап злиття (Merging): ваги LoRA-адаптерів інтегруються в базову модель у точності bfloat16, після чого модель конвертується у формат GGUF з квантуванням Q5_K_M (5-біт) [29]. Це забезпечує оптимальний баланс між якістю генерації та швидкістю інференсу на CPU/GPU.

2.3 Проєктування інтерфейсу та API

Розробка зручного інтерфейсу користувача та чіткого програмного інтерфейсу є вирішальним фактором для впровадження системи в клінічну практику. Навіть найточніша модель не буде використовуватися лікарями, якщо вона вимагає складних маніпуляцій або порушує звичний робочий процес. У даному підрозділі описано структуру API для взаємодії компонентів та дизайн інтерфейсу, орієнтований на мінімізацію когнітивного навантаження радіолога.

Враховуючи обрану сервіс-орієнтовану архітектуру, проєктування API розділено на два рівні: зовнішній API (для взаємодії вебклієнта з сервером Django) та внутрішній API (для комунікації Django з ML-сервісом BentoML).

Для клієнт-серверної взаємодії розроблено специфікацію на базі архітектурного стилю REST. Обмін даними відбувається у форматі JSON. Визначено наступні ключові точки доступу:

- завантаження досліджень;
- отримання попередніх результатів;
- генерація звіту;
- експорт.

Для завантаження досліджень система приймає файли зображень. На цьому етапі відбувається первинна валідація формату (DICOM/PNG) та перевірка на наявність метаданих.

Під час отримання попередніх результатів система повертає список патологій, автоматично детектованих візуальним екстрактором. Структура відповіді включає назву патології, ймовірність (confidence score) та бінарний статус, визначений пороговим значенням.

Генерація звіту – це критично важливий метод, що приймає відкоригований лікарем список тегів. Тіло запиту містить масив лише тих патологій, які були підтверджені користувачем. Це гарантує, що мовна модель не отримає хибних вхідних даних. Під час експорту, система генерує фінальний документ у форматі PDF, готовий до друку або збереження в лікарняній базі даних.

Внутрішня взаємодія з BentoML реалізована через синхронні HTTP-виклики. Вебсервер Django виступає клієнтом, який серіалізує зображення у бінарний потік для відправки на Visual Extractor та форматує промпти для відправки на Text Generator.

Враховуючи, що радіологи працюють у затемнених приміщеннях і потребують максимальної концентрації, дизайн системи будується навколо концепції Human-in-the-loop, де головну роль відведено екрану верифікації. Інтерфейс цього екрану логічно розділяє робочий простір на дві частини, одна з яких призначена для детального вивчення знімка за допомогою інструментів масштабування, панорамування та регулювання контрастності, що критично важливо для аналізу підозрілих зон. Інша частина відповідає за прийняття рішень і містить інтерактивний перелік із 14 основних патологій. Тут лікар бачить не лише запропоновані системою теги, а й рівень її впевненості, маючи змогу в один клік підтвердити або спростувати діагноз.

Логіка взаємодії побудована так, щоб мінімізувати кількість кліків. Система автоматично активує перемикачі для патологій з високою ймовірністю. Задача лікаря зводиться до швидкого перегляду списку («eye-balling») та, за необхідності, перемикання статусу помилкового тегу. Тільки після натискання кнопки «Підтвердити» активується процес генерації тексту.

Останній етап робочого процесу – робота з текстовим редактором. Згенерований звіт відображається у текстовому полі. Оскільки LLM може допускати стилістичні неточності, лікарю надається можливість правити текст вручну перед збереженням.

Для забезпечення кросбраузерної сумісності інтерфейс розроблено з використанням стандартних вебтехнологій та шаблонізатора Django. Це робить систему універсальною та незалежною від локального оточення, адже для її роботи достатньо наявності звичайного веббраузера без жодних додаткових плагінів чи програм.

Висновки до другого розділу

У другому розділі магістерської роботи обґрунтовано та спроектовано архітектуру програмної системи на основі сервіс-орієнтованого підходу, де вебдодаток на базі Django взаємодіє зі спеціалізованим ML-сервісом BentoML. Таке розділення забезпечує незалежне масштабування компонентів та надійну інтеграцію через REST API, дозволяючи винести ресурсомісткі обчислення на окремий сервер.

Для вирішення задачі діагностики розроблено гібридний метод, що складається з двох послідовних етапів. Модуль візуального аналізу побудовано на базі архітектури EfficientNet-B0, адаптованої для роботи з рентгенівськими знімками через конвертацію в RGB-простір та використання адаптивної еквалізації гістограм.

Модуль генерації тексту використовує спеціалізовану велику мовну модель BioMistral-7B, донавчену методом QLoRA. Використання проміжного представлення у вигляді верифікованих клінічних тегів, структурованого промпт-інжинірингу та детермінованої стратегії декодування мінімізує ризик виникнення «галюцинацій» та забезпечує відповідність згенерованих звітів медичним стандартам.

Спроектований інтерфейс користувача реалізує принцип Human-in-the-loop, надаючи лікарю інструменти для ручної верифікації автоматично визначених патологій перед формуванням фінального тексту. Розроблені архітектурні та алгоритмічні рішення створюють необхідну базу для програмної реалізації та експериментального дослідження системи у третьому розділі.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1 Засоби розробки та підготовка даних

Етап програмної реалізації розпочався з вибору технологічного стеку та формування навчальної вибірки. Враховуючи мультимодальний характер системи, критичним фактором був вибір інструментів, що забезпечують сумісність між класичними вебтехнологіями та спеціалізованими бібліотеками глибокого навчання.

В якості основної мови програмування обрано Python 3.10. Цей вибір обумовлений домінуванням Python у сфері Data Science, наявністю розвиненої екосистеми бібліотек для медичної візуалізації та веброзробки.

Для глибокого навчання обрано PyTorch, який використовується як базовий рушій для побудови та навчання нейронних мереж. Динамічний граф обчислень PyTorch дозволяє гнучко реалізувати нестандартну архітектуру Late Fusion. Hugging Face Transformers & PEFT – це бібліотеки для роботи з великими мовними моделями. Вони забезпечують завантаження ваг BioMistral-7B та реалізацію методу QLoRA для ефективного донавчання. Бібліотека timm (PyTorch Image Models) використана для ініціалізації візуального бекбону EfficientNet-B0 з попередньо навченими вагами ImageNet.

Django виступає ядром системи, відповідаючи за адміністративні функції та роботу зі структурою даних пацієнтів. Натомість BentoML використовується для інкапсуляції ML-моделей в ізольовані сервіси. Такий підхід вирішує критичну проблему черги до GPU-ресурсів та уніфікує процес запуску моделей через механізм Runner.

Для реалізації гібридної архітектури використано комбінацію двох відкритих наборів даних, що дозволило компенсувати недоліки кожного з них окремо.

Основним джерелом для навчання модуля візуального аналізу став датасет CheXpert від Stanford Hospital. Він містить понад 224,000 рентгенограм від 65,000 пацієнтів та використовується для тренування EfficientNet у задачі

мультиміткової класифікації. Ключовою перевагою цього набору є наявність структурованих метаданих (вік, стать) та маркування проєкцій, що критично для реалізації механізму Late Fusion. Розмітка включає 14 класів патологій.

Для донавчання великої мовної моделі використано набір даних Indiana University Chest X-Rays (OpenI) [30]. На відміну від CheXpert, цей набір містить якісні повні текстові звіти (Narrative Reports), що дозволяє використовувати його для fine-tuning моделі BioMistral з метою генерації зв'язного тексту. Обсяг вибірки становить близько 4,000 пар «Знімок-Звіт».

Перед розробкою алгоритмів препроцесингу було проведено детальний розвідувальний аналіз даних (EDA) для обох модальностей (рис. 3.1 – 3.4).

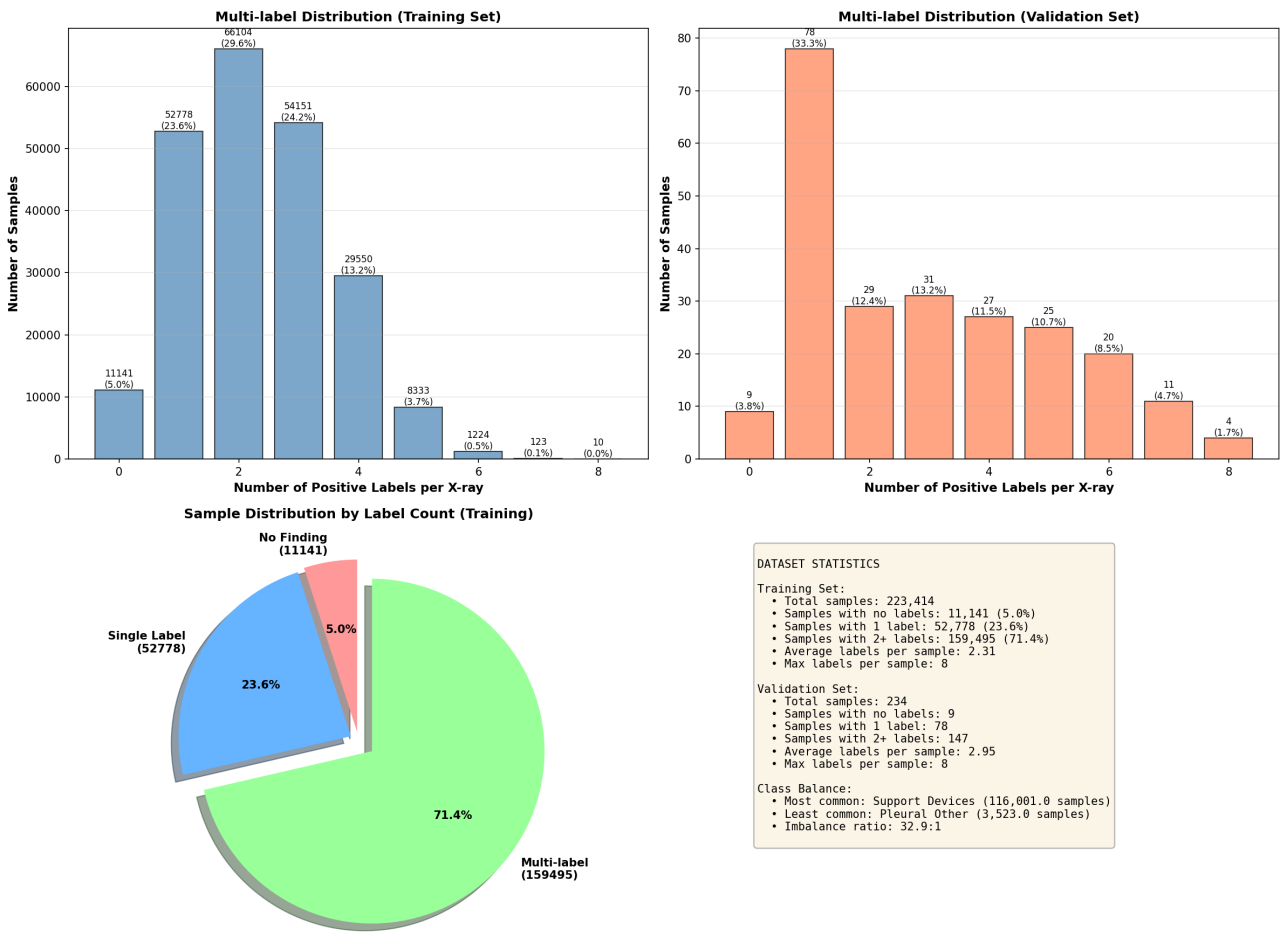


Рис. 3.1 – Аналіз розподілу міток

Аналіз розподілу міток (рис. 3.1) продемонстрував значний дисбаланс класів. Домінуючими класами є «Support Devices» (51.9%), «Lung Opacity» (47.3%) та «Pleural Effusion» (38.6%). Висока частота класу «Support Devices» свідчить про значну частку знімків важкохворих пацієнтів з реанімації, що

створює додатковий візуальний шум (електроди, трубки). Водночас, такі патології як «Pneumonia» (2.7%) та «Fracture» (4.0%) представлені вкрай мало. Дисбаланс сягає співвідношення 62.4:1, що безальтернативно вимагає використання зваженої функції втрат.

Аналіз кількості діагнозів на один знімок спростував можливість використання Multi-class класифікації. Лише 23.6% знімків мають одну мітку, тоді як абсолютна більшість (71.4%) зразків мають дві або більше підтверджених патологій одночасно. Середнє значення становить 2.31 патології на знімок, що обумовлює необхідність використання Sigmoid активації для кожного класу незалежно.

Аналіз сумісної появи діагнозів (рис. 3.2) виявив сильні клінічні кореляції. Найчастішою парою є «Lung Opacity + Support Devices» (27.5% випадків), що підтверджує важкість стану пацієнтів із затемненнями в легенях. Також зафіксовано сильну кореляцію (23.4%) між плевральним випотом та затемненням легень.

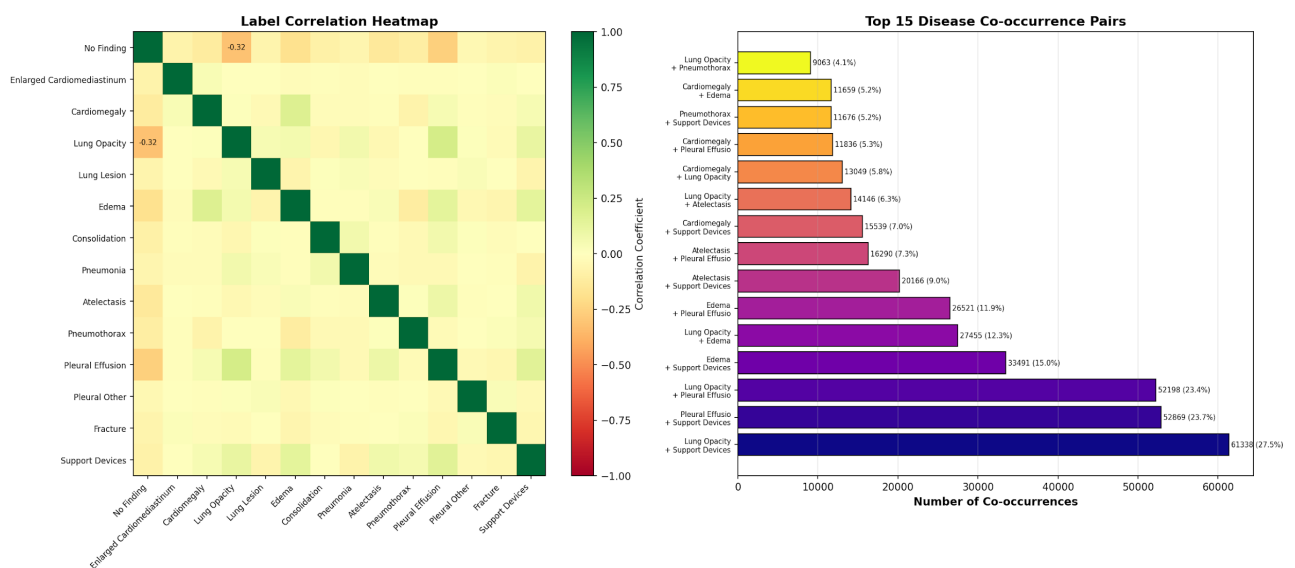


Рис. 3.2 – Аналіз сумісної появи діагнозів

Візуальний огляд підтвердив наявність значної варіабельності якості знімків та артефактів, що обґрунтовує необхідність використання жорсткої аугментації.

Аналіз повноти даних (рис. 3.3) показав, що поле findings (детальний опис) відсутнє у 13.3% звітів, тоді як impression (короткий висновок) – лише у 0.8%.

Оскільки для навчання моделі потрібен детальний опис для формування зв'язного тексту, записи з відсутніми findings підлягають виключенню, що скорочує ефективну вибірку.

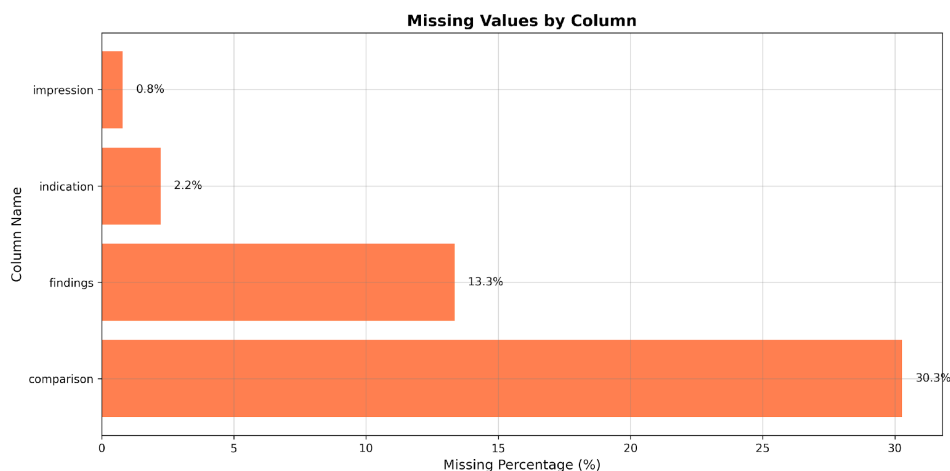


Рис. 3.3 – Діаграма відсутніх значень по полях

Проведений статистичний аналіз текстових атрибутів дозволив виявити ключові особливості структури медичних звітів. Як демонструють гістограми розподілу (рис. 3.4), секція Findings (опис знахідок) є значно об'ємнішою та варіативнішою за секцію Impression (висновок). Розподіл довжини обох секцій має виражену правосторонню асиметрію: медіанна кількість слів становить 27 для Findings проти лише 5 для Impression. Така лаконічність висновків свідчить про те, що лікарі-радіологи схильні формулювати діагноз максимально стисло, часто обмежуючись однією-двома фразами.

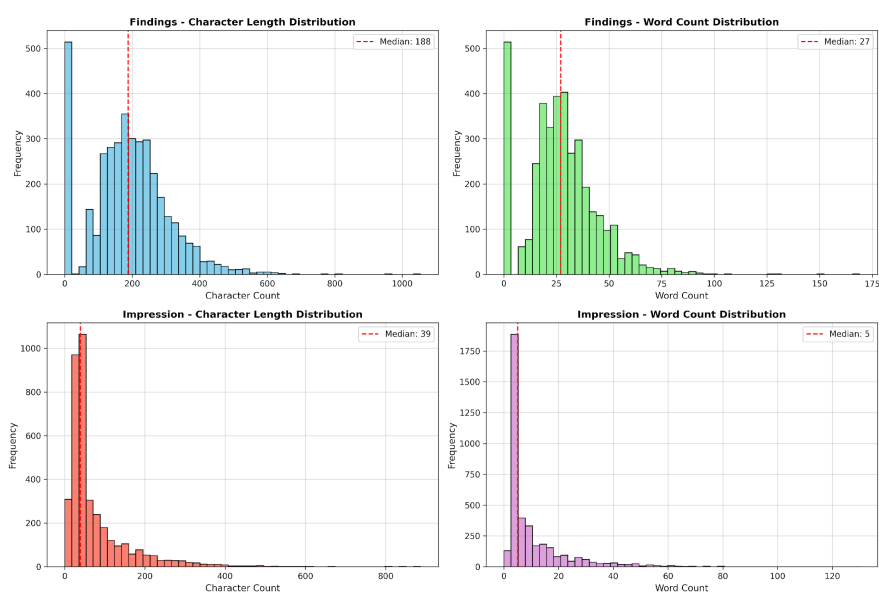


Рис. 3.4 – Розподіл кількості літер і слів

Аналіз кореляційної матриці (рис. 3.5) виявив важливий аспект: між обсягом описової частини (findings_length) та довжиною висновку (impression_length) фактично відсутній лінійний зв'язок (коефіцієнт кореляції Пірсона $r = -0.07$). Це означає, що детальний опис знімка не обов'язково призводить до довгого висновку, і навпаки – короткий опис може супроводжуватися розгорнутим діагнозом. Водночас спостерігається помірна позитивна кореляція ($r = 0.42$) між кількістю патологій (label_count) та довжиною тексту. Це логічно пояснюється тим, що наявність декількох патологій вимагає детальнішого опису та документування.

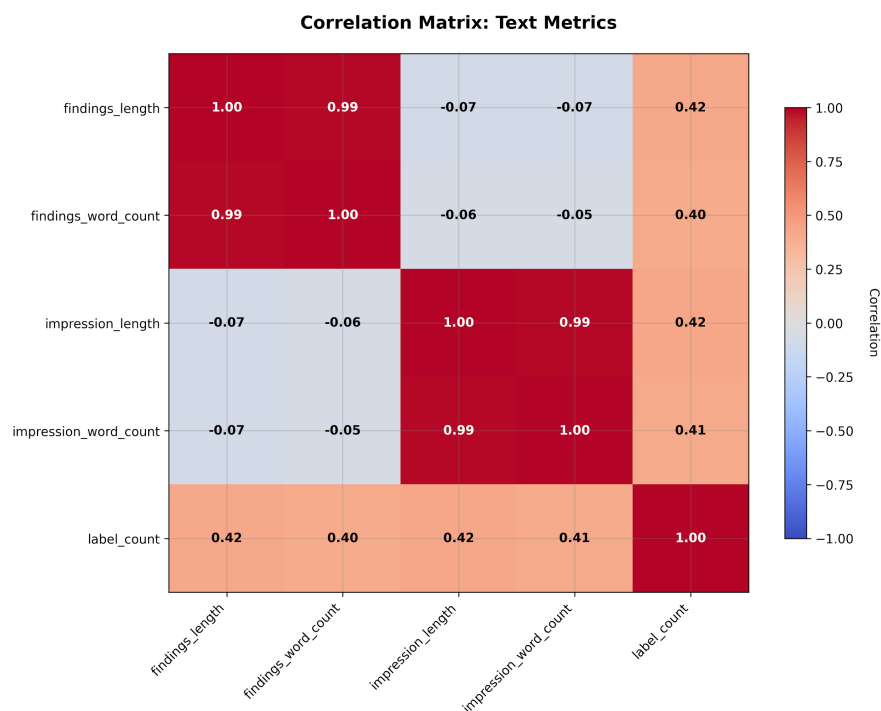


Рис. 3.5 – Матриця кореляції текстових метрик

З технічної точки зору, аналіз граничних значень показав, що навіть найдовший звіт у вибірці не перевищує 175 слів. Враховуючи контекстне вікно моделі BioMistral, яке становить 4096 токенів, така довжина вхідних даних дозволяє подавати на вхід моделі повний текст звітів (разом із промптом та системними інструкціями) без ризику усікання інформації (truncation), що є критично важливим для збереження медичної точності.

Аналіз частотності слів у корпусі (рис. 3.6 – 3.7) виявив суттєве забруднення даних артефактами анонімізації. Токен «xxxx», який

використовується для приховування персональних даних (дат, прізвищ, номерів медичних карток), займає третю позицію за популярністю в обох секціях звіту.

У секції Findings цей токен зустрічається 2401 раз, поступаючись лише загальновживаним термінам «normal» та «pleural», і випереджаючи такі клінічно значущі маркери, як «pneumothorax» (2373) та «effusion» (2213). Аналогічна ситуація спостерігається в секції Impressions, де частота «xxxx» становить 1140 входжень, що співмірно з частотою терміну «disease» (1066).

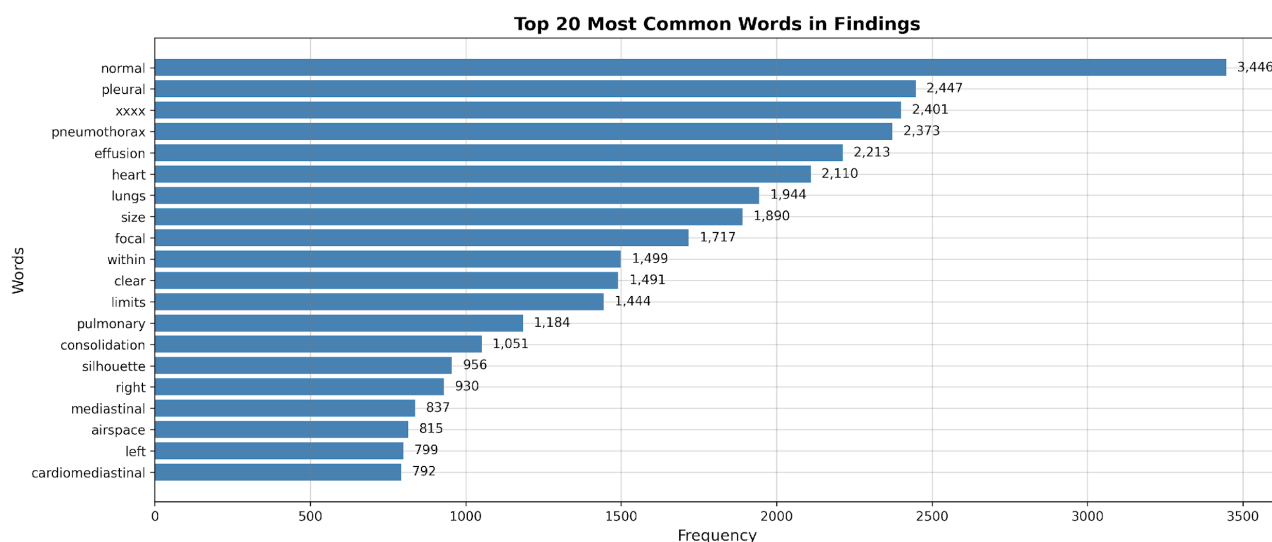


Рис. 3.6 – Топ 20 найбільш поширених слів в Findings

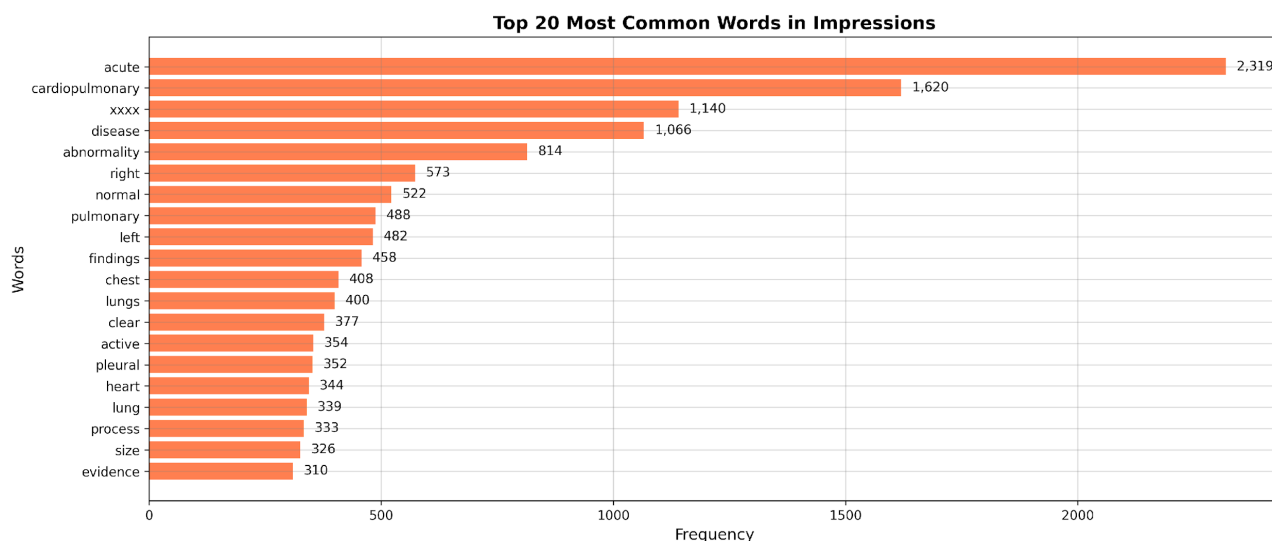


Рис. 3.7 – Топ 20 найбільш поширених слів в Impressions

Така висока щільність «шумового» токена (наприклад, у патернах «Comparison with xxxx») створює ризик зміщення уваги моделі під час навчання. Без належної обробки модель може почати генерувати токен «xxxx» як частину

валідного медичного опису. Це вимагає спеціального кроку препроцесингу: або видалення речень з цим токеном, або навчання моделі ігнорувати його.

Аналіз розподілу найбільш поширених медичних міток (рис. 3.8) демонструє виражене зміщення вибірки у бік відсутності патологій. Абсолютним лідером серед міток є категорія «normal», яка ідентифікована у 1379 випадках, що складає 35.8% від загального обсягу даних. Цей показник більш ніж удвічі перевищує частоту наступних за поширеністю міток – «Lung» (14.4%) та «Opacity» (13.2%).

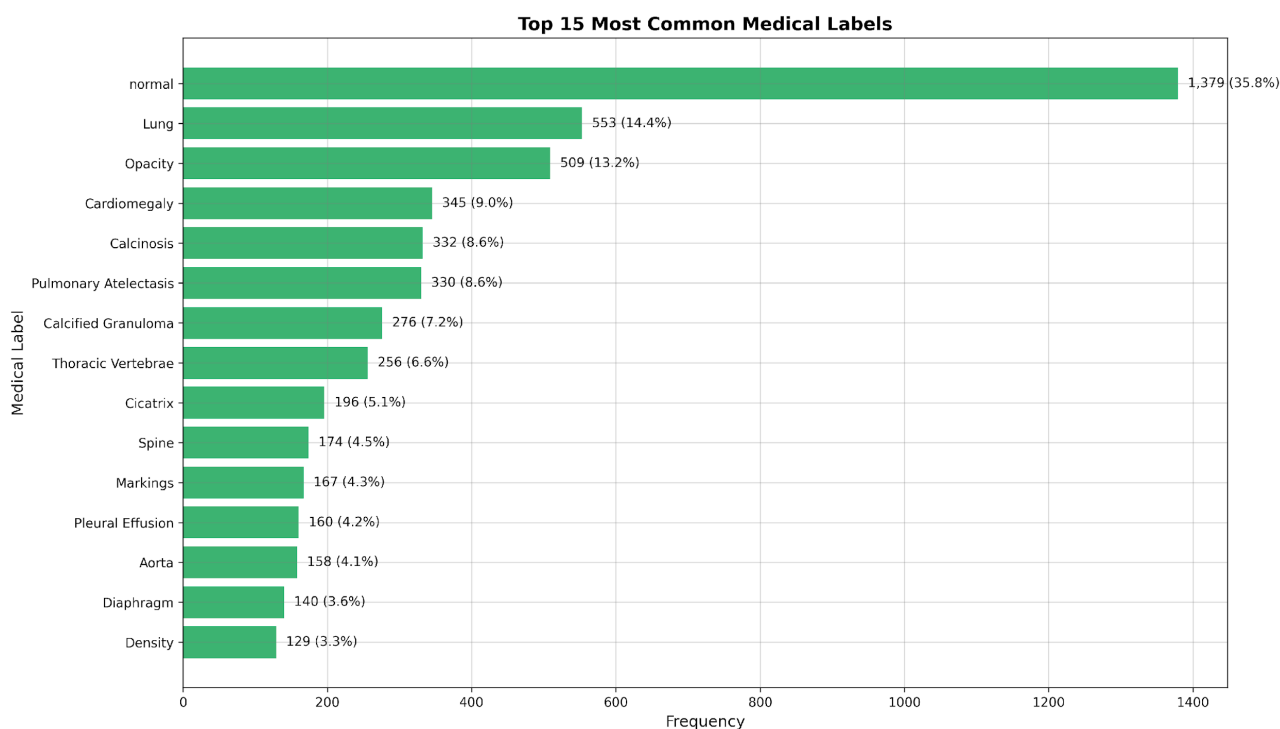


Рис. 3.8 – Топ 15 найбільш поширених термінів

Такий значний розрив між мажоритарним класом («normal») та специфічними патологіями, такими як «Cardiomegaly» (9.0%) або «Pleural Effusion» (4.2%), свідчить про класичну проблему незбалансованих медичних даних (class imbalance). Це створює ризик формування у моделі схильності частіше передбачати норму через її статистичну перевагу, ігноруючи рідкісні, але критичні діагнози.

На основі результатів EDA було розроблено та імплементовано два незалежні ETL-пайплайни.

Процес підготовки даних реалізовано в межах класу FusionChexpertDataset, де обробка мультимодальних вхідних даних розділена на нормалізацію

табличних метаданих та графічний пайплайн (рис. 3.9). Для числових атрибутів, зокрема віку пацієнта, застосовано масштабування у діапазон $[0, 1]$ шляхом ділення на константу 100, тоді як категоріальна змінна статі кодується бінарно із заповненням пропущених значень нейтральним показником 0.5. Для формування цільового вектора міток обрано 14 патологій, при цьому, враховуючи специфіку розмітки CheXpert, всі невизначені діагнози (значення -1) та пропуски замінюються на 0, що відповідає відсутності патології.

```
class FusionChexpertDataset(Dataset):
    def __init__(self, df, transform=None):
        self.df = df
        self.transform = transform
        self.label_columns = [
            'No Finding', 'Enlarged Cardiomeastinum', 'Cardiomegaly',
            'Lung Opacity', 'Lung Lesion', 'Edema', 'Consolidation',
            'Pneumonia', 'Atelectasis', 'Pneumothorax', 'Pleural Effusion',
            'Pleural Other', 'Fracture', 'Support Devices'
        ]
        self.df = self.df.copy()
        self.df['Age'] = self.df['Age'].fillna(self.df['Age'].median())
        self.df['Sex'] = self.df['Sex'].fillna('Unknown')
        self.df['Age_Norm'] = self.df['Age'] / 100.0
        # Encode Sex (Male=0, Female=1, Other=0.5)
        self.df['Sex_Enc'] = self.df['Sex'].map({'Male': 0.0, 'Female': 1.0}).fillna(0.5)
        for col in self.label_columns:
            if col in self.df.columns:
                self.df[col] = self.df[col].fillna(0).replace(-1, 0)

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        img_path = self.df.iloc[idx]['Path']
        try:
            image = Image.open(img_path).convert('RGB')
        except:
            image = Image.new('RGB', (240, 240), color='gray')

        if self.transform:
            image = self.transform(image)
        meta = torch.tensor([
            self.df.iloc[idx]['Age_Norm'],
            self.df.iloc[idx]['Sex_Enc']
        ], dtype=torch.float32)
        labels = self.df.iloc[idx][self.label_columns].values.astype(np.float32)
        labels = torch.from_numpy(labels)
        return image, meta, labels
```

Рис. 3.9 – Клас датасету

Графічний пайплайн розпочинається з примусової конвертації завантажених зображень у колірний простір RGB та генерації заповнювачів у разі пошкодження вихідного файлу. Для покращення якості ознак та локального контрасту легених структур імплементовано алгоритм CLAHE (Contrast Limited Adaptive Histogram Equalization) з порогом відсікання 2.0 та сіткою 8×8 (рис. 3.10). Особливістю реалізації є те, що алгоритм застосовується виключно

до каналу яскравості (Y) після переведення зображення у простір YCbCr, що дозволяє уникнути спотворення колірних артефактів.

```
class ApplyCLAHE:
    def __init__(self, clip_limit=2.0, tile_grid_size=(8, 8)):
        self.clip_limit = clip_limit
        self.tile_grid_size = tile_grid_size
        self.clahe = cv2.createCLAHE(
            clipLimit=self.clip_limit,
            tileGridSize=self.tile_grid_size
        )

    def __call__(self, img: Image.Image) -> Image.Image:
        img_ybcr = img.convert('YCbCr')
        img_np = np.array(img_ybcr)
        y_channel = img_np[:, :, 0]
        y_channel_clahe = self.clahe.apply(y_channel)
        img_np[:, :, 0] = y_channel_clahe

        size = (img_np.shape[1], img_np.shape[0])
        img_clahe = Image.new('YCbCr', size)
        img_clahe.frombytes(img_np.tobytes())
        img_clahe_rgb = img_clahe.convert('RGB')

        return img_clahe_rgb
```

Рис. 3.10 – Клас застосування CLANE

На етапі трансформації реалізовано відмінні стратегії для різних частин вибірки: тренувальний набір піддається аугментації через випадкове горизонтальне відображення та афінні перетворення (малий поворот, трансляція) для підвищення узагальнюючої здатності моделі, тоді як валідаційний набір проходить лише детерміновану зміну розміру до 240×240 пікселів. Завершальним кроком є нормалізація тензорів з використанням стандартних статистичних параметрів набору ImageNet (середнє значення та стандартне відхилення), що забезпечує коректну роботу попередньо навчених верств нейронної мережі (рис. 3.11).

```
def get_train_transforms(image_size=240):
    return transforms.Compose([
        ApplyCLAHE(clip_limit=2.0, tile_grid_size=(8, 8)),
        transforms.Resize((image_size, image_size)),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomAffine(degrees=5, translate=(0.05, 0.05)),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])

def get_val_transforms(image_size=240):
    return transforms.Compose([
        ApplyCLAHE(clip_limit=2.0, tile_grid_size=(8, 8)),
        transforms.Resize((image_size, image_size)),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])
```

Рис. 3.11 – Аугментація зображень

Для навчання формуються пари «Prompt – Completion». Prompt – це серіалізований список MeSH-тегів (наприклад, Tags: [Calcified Granuloma, Normal]). Completion – повний текст секцій Findings та Impression.

Розбиття даних на вибірки виконувалося строго за ідентифікатором пацієнта, щоб уникнути витоку даних. Для датасету CheXpert використано розподіл: 85% – навчальна вибірка, 15% – валідаційна. Для меншого датасету Indiana University використано 90% даних для навчання та 10% для валідації, контроль якості якої здійснювався за метрикою функції втрат (Perplexity).

3.2 Реалізація компонентів системи

Програмна реалізація системи виконана у відповідності до спроектованої сервіс-орієнтованої архітектури. Розробку розділено на три логічні етапи: імплементація модуля комп'ютерного зору з підтримкою мультимодального входу, налаштування генеративної мовної моделі та розробка вебсервісу для інтеграції компонентів.

Ядром візуальної підсистеми є клас EfficientNetFusion, реалізований на базі бібліотеки PyTorch (рис. 3.12). Архітектура моделі передбачає обробку двох потоків даних: фронтального зображення та вектора метаданих.

```

class EfficientNetFusion(nn.Module):
    def __init__(self, num_classes, pretrained=True):
        super().__init__()
        self.image_model = timm.create_model(
            'efficientnet_b0',
            pretrained=pretrained,
            num_classes=0,
            drop_rate=0.3,
            drop_path_rate=0.2
        )
        self.num_image_features = 1280
        self.meta_net = nn.Sequential(
            nn.Linear(2, 32),
            nn.BatchNorm1d(32),
            nn.ReLU(),
            nn.Dropout(0.1)
        )
        self.classifier = nn.Linear(self.num_image_features + 32, num_classes)

    def forward(self, image, meta):
        img_features = self.image_model(image)
        meta_features = self.meta_net(meta)
        combined_features = torch.cat((img_features, meta_features), dim=1)
        output = self.classifier(combined_features)
        return output

```

Рис. 3.12 – Клас EfficientNetFusion

Архітектура системи передбачає паралельну обробку різномірних даних. Візуальний канал базується на бібліотеці `timm` та моделі `EfficientNet-B0`, де параметр `num_classes=0` дозволяє отримувати чистий вектор ознак розмірністю 1280 безпосередньо із зображення. Одночасно з цим окрема повнозв'язна мережа обробляє метадані пацієнта, трансформуючи нормалізовані показники віку та закодовану статтю у компактний 32-вимірний простір ознак.

У методі прямого поширення (`forward`) реалізовано логіку пізнього злиття. Вектор ознак від візуальної гілки та гілки метаданих об'єднуються за допомогою операції конкатенації (`torch.cat`), утворюючи єдиний вектор розмірністю $1280 + 32 = 1312$. Цей вектор подається на фінальний класифікатор, який повертає логіти для 14 класів патологій.

Для забезпечення ефективного навчання на великому обсязі даних (`CheXpert`) реалізовано оптимізований пайплайн завантаження даних через `DataLoader` (рис. 3.13). Використовуються наступні параметри: `num_workers` налаштовано на максимальну кількість ядер CPU для паралельної підготовки батчів, `pin_memory=True` забезпечує швидке копіювання тензорів у пам'ять GPU (`Page-locked memory`), `prefetch_factor` дозволяє завантажувати наступні батчі

наперед, мінімізуючи простій GPU, `persistent_workers=True` утримує процеси-воркери активними між епохами, економлячи час на їх перезапуск.

```
train_loader = DataLoader(  
    train_dataset,  
    batch_size=CONFIG['batch_size'],  
    shuffle=True,  
    num_workers=CONFIG['num_workers'],  
    pin_memory=CONFIG['pin_memory'],  
    prefetch_factor=CONFIG['prefetch_factor'],  
    persistent_workers=True  
)  
  
val_loader = DataLoader(  
    val_dataset,  
    batch_size=CONFIG['batch_size'] * 2,  
    shuffle=False,  
    num_workers=CONFIG['num_workers'],  
    pin_memory=CONFIG['pin_memory'],  
    prefetch_factor=CONFIG['prefetch_factor'],  
    persistent_workers=True  
)
```

Рис. 3.13 – Пайплайн завантаження даних

Для навчання використано оптимізатор AdamW та функцію втрат BCEWithLogitsLoss зі зважуванням класів для компенсації дисбалансу (рис. 3.14).

```
print("Calculating positive weights for weighted BCE loss...")  
label_columns = train_dataset.label_columns  
train_df_for_weights = train_dataset.df  
pos_counts = train_df_for_weights[label_columns].sum()  
neg_counts = len(train_df_for_weights) - pos_counts  
pos_weights = (neg_counts + 1e-6) / (pos_counts + 1e-6)  
pos_weights_tensor = torch.tensor(pos_weights.values, dtype=torch.float32).to(device)  
print(f"Positive weights calculated (sample): {pos_weights_tensor[:5]}")  
  
criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weights_tensor)
```

Рис. 3.14 – Зважена функція витрат

Технічна реалізація текстового модуля спирається на стек інструментів екосистеми Hugging Face, де кожен компонент відповідає за окремий етап роботи з нейромережею. Основу архітектури забезпечує бібліотека transformers, тоді як безпосереднє керування циклом навчання та інструктивним донавчанням покладено на пакет trl. Для оптимізації використання ресурсів GPU задіяно бібліотеки peft та bitsandbytes, які дозволяють застосувати методи ефективного налаштування параметрів та 4-бітного квантування.

```
def load_model(model_id, training_mode="8bit", device_map="auto"):
    bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_compute_dtype=torch.bfloat16,
        bnb_4bit_use_double_quant=True,
    )
    model = AutoModelForCausalLM.from_pretrained(
        model_id,
        quantization_config=bnb_config,
        device_map=device_map,
        trust_remote_code=True,
    )

    tokenizer = AutoTokenizer.from_pretrained(model_id)

    if tokenizer.pad_token is None:
        tokenizer.pad_token = tokenizer.eos_token
        print("Set pad_token = eos_token")

    print("✓ Model loaded successfully")
    print(f" Device map: {device_map}")
    print(f" Model dtype: {model.dtype}")
    print("="*80)

    return model, tokenizer
```

Рис. 3.15 – Завантаження моделі

Конфігурація методу LoRA реалізована через об'єкт `LoraConfig` для задачі CAUSAL_LM, що відповідає архітектурі генеративного декодера. Стратегія навчання передбачає інтеграцію низькорангових адаптерів у всі лінійні шари моделі (включно з блоками уваги та повнозв'язними шарами), що визначається параметром `target_modules`. Для оптимізації використання пам'яті навчання векторів зсуву вимкнено налаштуванням `bias='none'` (рис. 3.16).

```
def apply_lora(model, lora_config_dict, training_mode="8bit"):
    print("Preparing model for k-bit training...")
    model = prepare_model_for_kbit_training(model)

    lora_config = LoraConfig(
        r=lora_config_dict["r"],
        lora_alpha=lora_config_dict["lora_alpha"],
        target_modules=lora_config_dict["target_modules"],
        lora_dropout=lora_config_dict["lora_dropout"],
        bias="none",
        task_type="CAUSAL_LM",
    )

    model = get_peft_model(model, lora_config)

    model.print_trainable_parameters()

    return model, lora_config
```

Рис. 3.16 – Конфігурація LORA

Навчання реалізовано на базі `SFTTrainer` із використанням оптимізатора `paged_adamw_8bit`, що запобігає переповненню пам'яті за допомогою механізму

сторінкового розбиття. Для прискорення обчислень та забезпечення числовий стабільності активовано формати bf16 і tf32 у поєднанні з технікою `gradient_checkpointing`. Стратегія валідації передбачає збереження найкращої версії моделі (`load_best_model_at_end=True`) на основі покрокового моніторингу метрик (рис. 3.17).

```
def create_trainer(model, tokenizer, train_dataset, eval_dataset, lora_config, train_config, output_dir, training_mode="8bit"):
    optimizer = "paged_adamw_8bit"

    training_args = TrainingArguments(
        output_dir=str(output_dir),
        per_device_train_batch_size=train_config["per_device_train_batch_size"],
        gradient_accumulation_steps=train_config["gradient_accumulation_steps"],
        learning_rate=train_config["learning_rate"],
        optim=optimizer,
        num_train_epochs=train_config["num_train_epochs"],
        logging_steps=train_config["logging_steps"],
        save_steps=train_config["save_steps"],
        evaluation_strategy="steps",
        eval_steps=train_config["eval_steps"],
        bf16=True,
        tf32=True,
        save_total_limit=3,
        load_best_model_at_end=True,
        gradient_checkpointing=True if training_mode != "full_precision" else False,
        report_to="none",
        push_to_hub=False,
    )

    trainer = SFTTrainer(
        model=model,
        train_dataset=train_dataset,
        eval_dataset=eval_dataset,
        peft_config=lora_config,
        dataset_text_field="text",
        max_seq_length=train_config["max_seq_length"],
        tokenizer=tokenizer,
        args=training_args,
    )

    print(f"✓ Trainer initialized successfully ({training_mode} mode)")
    return trainer
```

Рис. 3.17 – Конфігурація тренування

Для забезпечення ефективного розгортання системи в умовах обмежених ресурсів було реалізовано пайплайн оптимізації та квантування фінальної моделі.

Після завершення навчання ваги адаптерів LoRA були інтегровані назад у базову модель BioMistral-7B (рис. 3.18). Критично важливим технічним рішенням було виконання злиття у точності bfloat16 (незважаючи на те, що навчання відбувалося у 4-бітах). Це забезпечило найкращу вихідну якість для подальшої конвертації. Використано метод `model.merge_and_unload()`.

```
def merge_adapter_to_base(model_id, adapter_path, output_path):
    print(f"Loading base model in bfloat16: {model_id}")

    base_model = AutoModelForCausalLM.from_pretrained(
        model_id,
        torch_dtype=torch.bfloat16,
        device_map="cpu",
        trust_remote_code=True,
    )

    tokenizer = AutoTokenizer.from_pretrained(model_id)

    print(f"Loading adapter from: {adapter_path}")
    model = PeftModel.from_pretrained(base_model, str(adapter_path))
    print("Merging adapter weights into base model...")
    model = model.merge_and_unload()
    print("✓ Merge complete (bfloat16)")

    print(f"Saving merged model to: {output_path}")
    model.save_pretrained(str(output_path))
    tokenizer.save_pretrained(str(output_path))
    print("✓ Merged model saved successfully")
```

Рис. 3.18 – Функція злиття ваг

Об'єднану модель конвертовано у формат GGUF (GPT-Generated Unified Format) за допомогою інструментарію llama.cpp (рис. 3.19). Цей бінарний формат оптимізований для швидкого завантаження (mmap) та ефективного інференсу на CPU та GPU.

```
!python llama.cpp/convert_hf_to_gguf.py "{MERGED_MODEL_PATH}" \
--outfile "{GGUF_F16_PATH}" \
--outtype f16
```

Рис. 3.19 – Конвертація у GGUF

Для фінального розгортання системи обрано схему Q5_K_M, зображену на рис. 3.20, яка базується на стратегії змішаної бітності. Цей підхід дозволяє зберегти високу когерентність тексту шляхом квантування критично важливих тензорів Attention.wv та FeedForward.w2 до 6 біт, тоді як усі інші компоненти моделі стискаються до 5 біт.

```
!llama.cpp/build/bin/llama-quantize "{GGUF_F16_PATH}" "{GGUF_Q5_PATH}" Q5_K_M
```

Рис. 3.20 – Квантування Q5_K_M

Основу веб-платформи становить фреймворк Django, який через модель даних XRayReport консолідує зображення, метадані та згенеровані висновки в

єдину структуру. Взаємодія з інтелектуальними алгоритмами винесена в окремий мікросервіс під управлінням BentoML. Зокрема, для текстового інференсу застосовано бібліотеку llama-cpp-python, яка працює з локальним файлом ваг `biomistral` у форматі `gguf`, що забезпечує високу швидкість та незалежність від зовнішніх API. Сама логіка роботи побудована за принципом Human-in-the-loop, надаючи лікарю проміжний інтерфейс для верифікації та редагування тегів перед фінальною генерацією звіту.

На рис. 3.21 представлено інтерфейс головної сторінки вебзастосунку. В центральній частині розташована зона для завантаження медичних зображень, що підтримує формати JPG, PNG та DICOM. Нижче знаходиться форма для введення демографічних даних пацієнта, які використовуються моделлю як додатковий контекст.

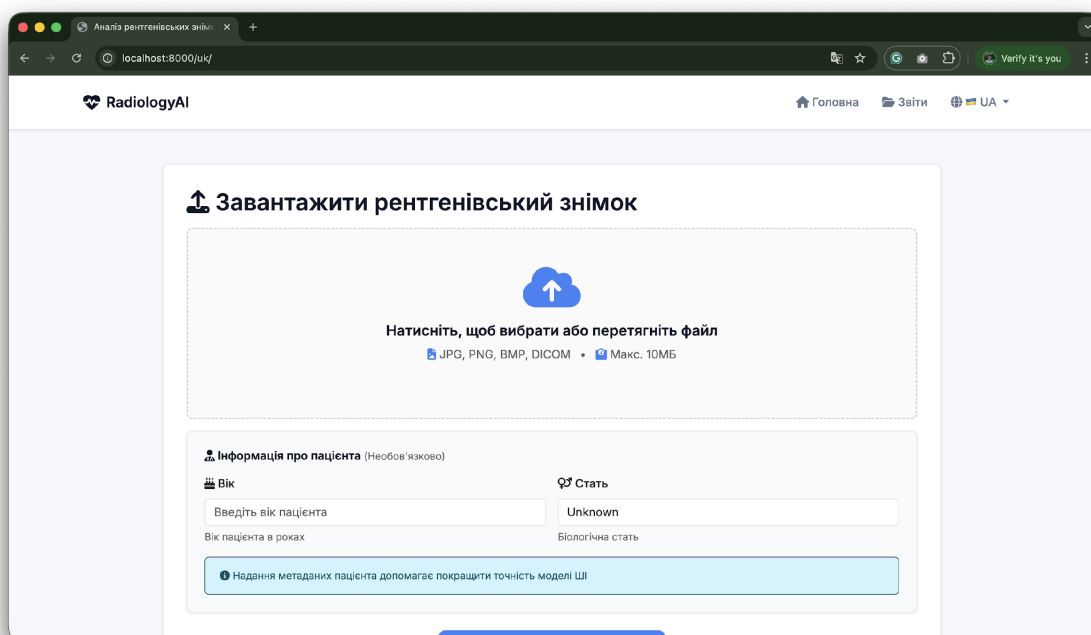


Рис. 3.21 – Головна сторінка

Після успішного завантаження файлу система виконує первинну обробку зображення. Як показано на рис. 3.22, інтерфейс відображає попередній перегляд знімка та активує статус «Готовий до аналізу», дозволяючи лікареві ініціювати процес розпізнавання.

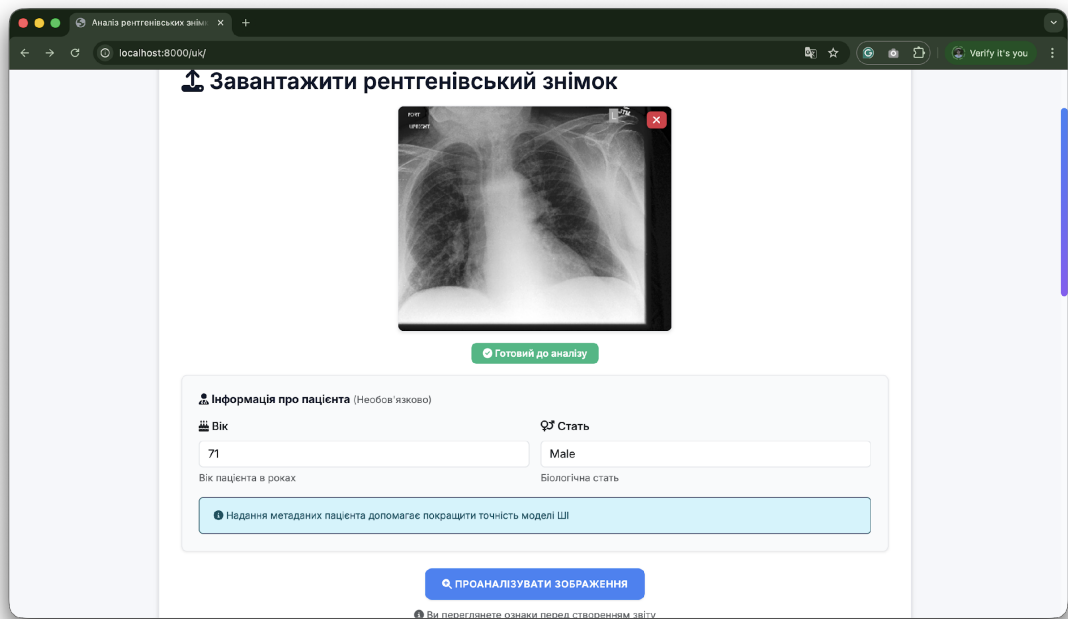


Рис. 3.22 – Інтерфейс підготовки до аналізу знімка

Після завершення обробки користувач перенаправляється на сторінку верифікації результатів (рис. 3.23). У правій панелі відображається інтерактивний список патологій, де синім кольором виділено ознаки, виявлені нейромережею з високою ймовірністю. Лікар має можливість вручну скорегувати цей список, додаючи або видаляючи ознаки перед генерацією звіту.

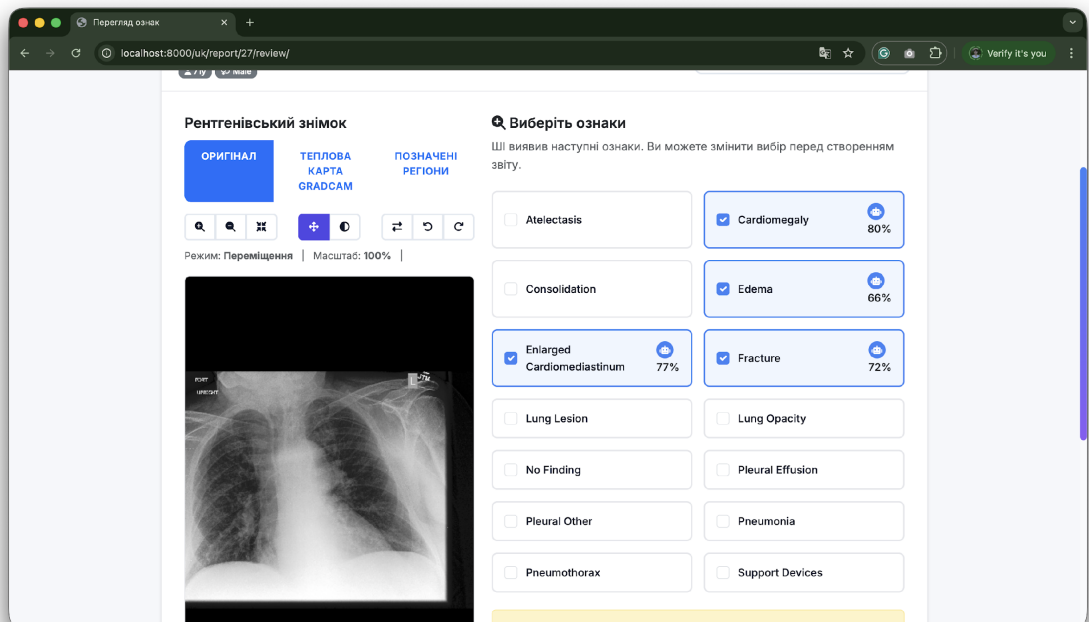


Рис. 3.23 – Сторінка огляду та верифікації виявлених ознак

Для забезпечення прозорості рішень ІІІ у системі реалізовано режим візуалізації теплових карт уваги Grad-CAM (рис. 3.24). Цей інструмент підсвічує області знімка, які мали найбільший вплив на класифікацію патології, дозволяючи лікарю оцінити адекватність роботи моделі.

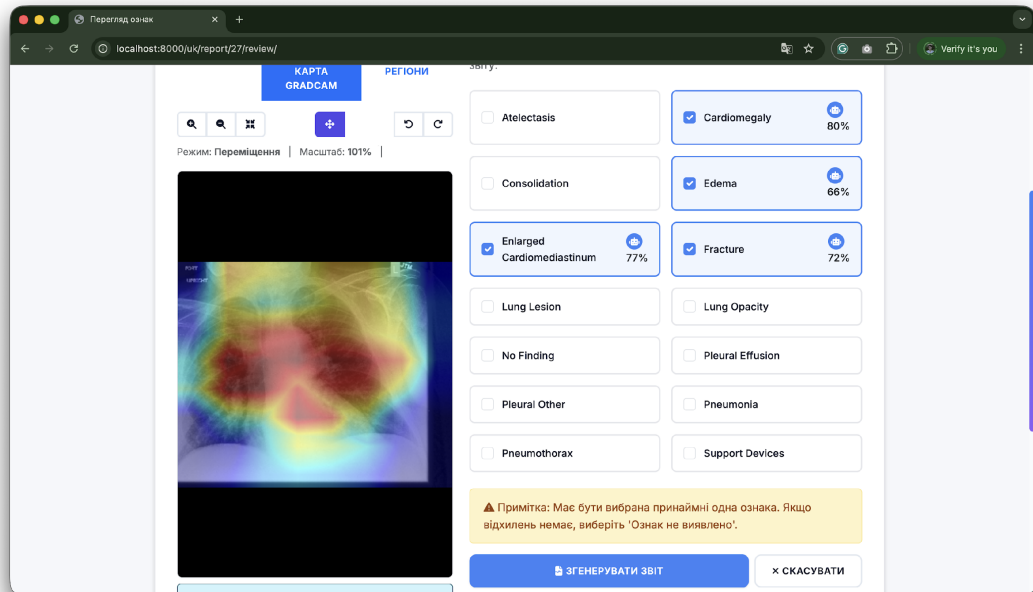


Рис. 3.24 – Візуалізація теплової карти Grad-CAM

Режим «Позначені регіони» (рис. 3.25) візуалізує результати виявлення об'єктів. Система накладає на рентгенівський знімок обмежувальні рамки (bounding boxes) різних кольорів, що чітко локалізують виявлені аномалії.

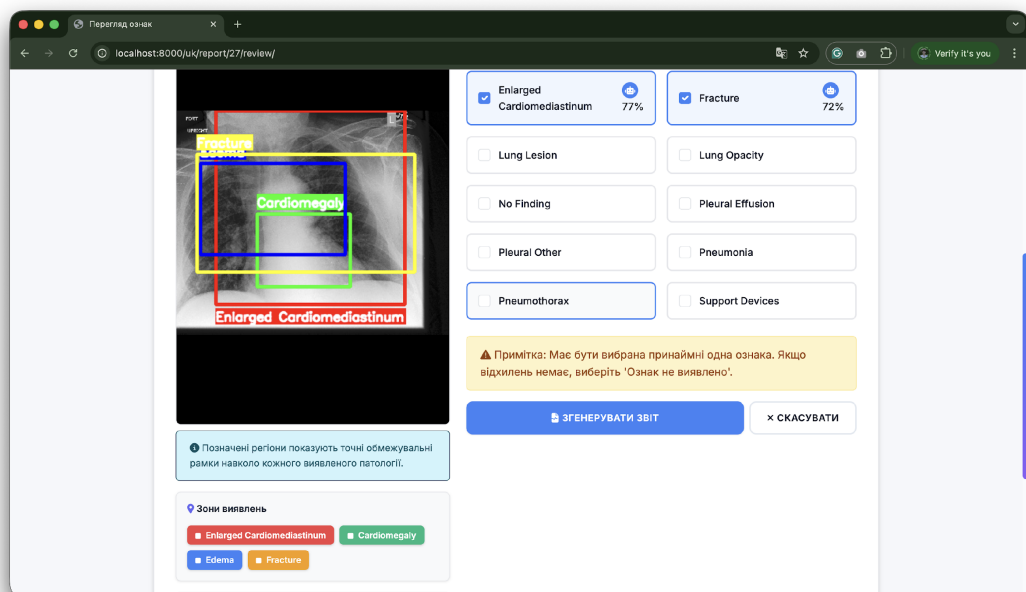


Рис. 3.25 – Локалізація зон виявлених патологій

Для детального вивчення дрібних структур реалізовано режим поглибленого перегляду (рис. 3.26). Інтерфейс надає інструменти для панорамування, зміни масштабу (Zoom) та регулювання контрастності зображення, що допомагає лікарю детальніше роздивитися спірні ділянки.

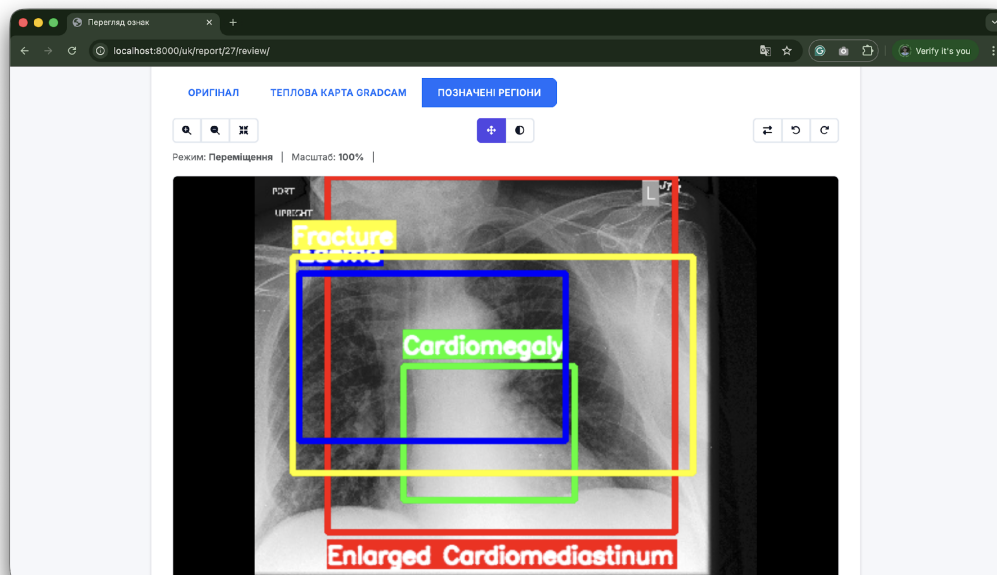


Рис. 3.26 – Режим огляду знімку на весь екран

На основі верифікованих ознак мультимодальна модель генерує текстовий опис. На сторінці результатів (рис. 3.27) представлено фінальний радіологічний звіт, структурований на секції Findings та Impression. Лікар має доступ до текстового редактора для внесення фінальних правок у згенерований текст.

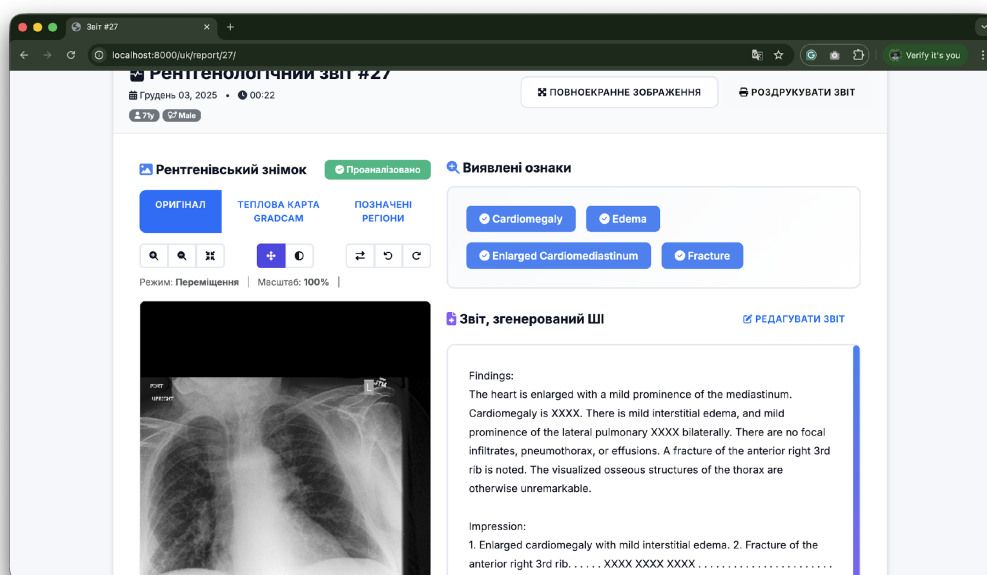


Рис. 3.27 – Сторінка радіологічного звіту

Система підтримує експорт результатів у формат PDF для друку або збереження в електронній картці пацієнта. Вікно попереднього перегляду друку (рис. 3.28) формує стандартизований документ, що містить інформацію про пацієнта, оригінальний знімок та текст висновку.

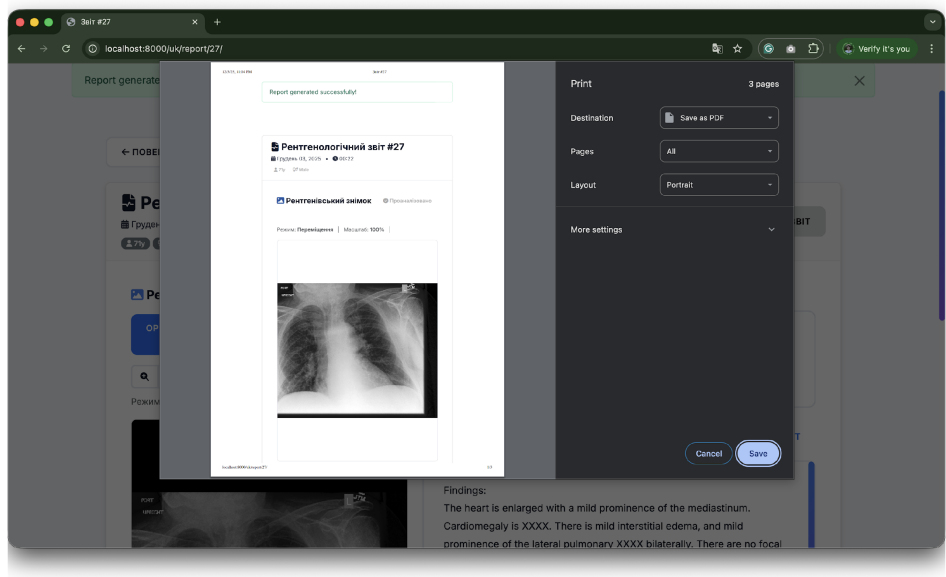


Рис. 3.28 – Друк звіту

Доступ до історії всіх проведених аналізів здійснюється через сторінку «Архів звітів» (рис. 3.29). Таблиця містить дані про ID дослідження, дату завантаження, статус обробки та ключові виявлені ознаки. Інтерфейс дозволяє повернутися до будь-якого попереднього запису для перегляду або доопрацювання звіту.

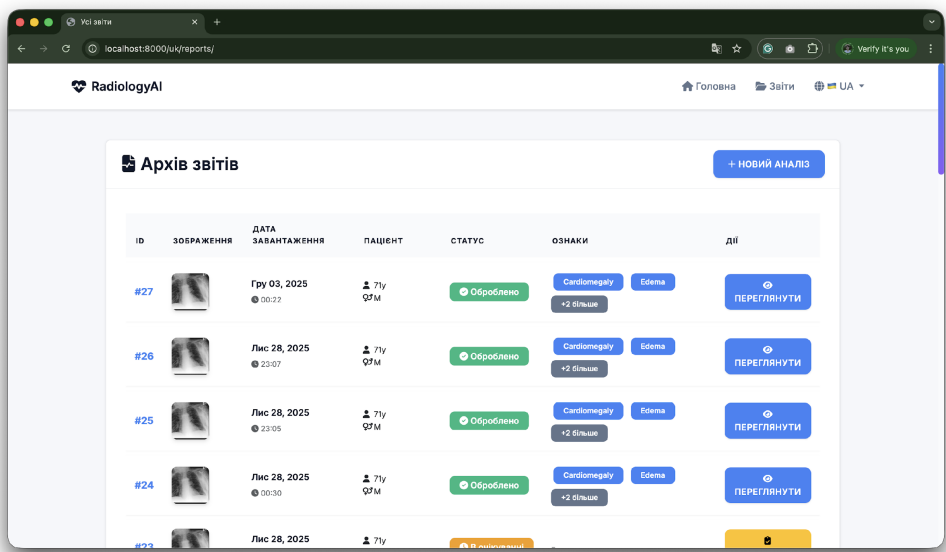


Рис. 3.29 – Сторінка історії та архіву звітів

Реалізований графічний інтерфейс забезпечує повний цикл роботи з радіологічними даними. Ключовою особливістю системи є інтеграція інструментів пояснюваного штучного інтелекту (Grad-CAM, візуалізація регіонів інтересу), що дозволяє лікарю не просто отримувати результат класифікації, а й верифікувати логіку прийняття рішень моделлю. Такий підхід реалізує концепцію Human-in-the-Loop, де неймережа виступає в ролі інтелектуального асистента, автоматизуючи рутинний опис знахідок, тоді як критичний контроль та остаточне затвердження діагнозу залишаються за фахівцем. Це мінімізує ризик медичних помилок та суттєво прискорює процес опису рентгенограм.

3.3 Експериментальне дослідження якості роботи

Для перевірки ефективності розробленої системи було проведено серію експериментів на тестових вибірках датасетів CheXpert (для візуального модуля) та Indiana University Reports (для текстового модуля). Метою експериментів було не лише отримання кількісних показників точності, а й аналіз поведінки системи в умовах клінічної невизначеності та дисбалансу класів.

Навчання моделі EfficientNet-B0 з використанням архітектури Late Fusion продемонструвало стабільну збіжність, незважаючи на складність мультиміткової задачі.

Аналіз кривих функції втрат (Loss) показує, що модель швидко вчиться на ранніх епохах. Training Loss стабільно знижується, тоді як Validation Loss досягає свого мінімуму (0.957) на 5-й епосі, після чого починає повільно зростати, що свідчить про початок перенавчання (рис. 3.30).

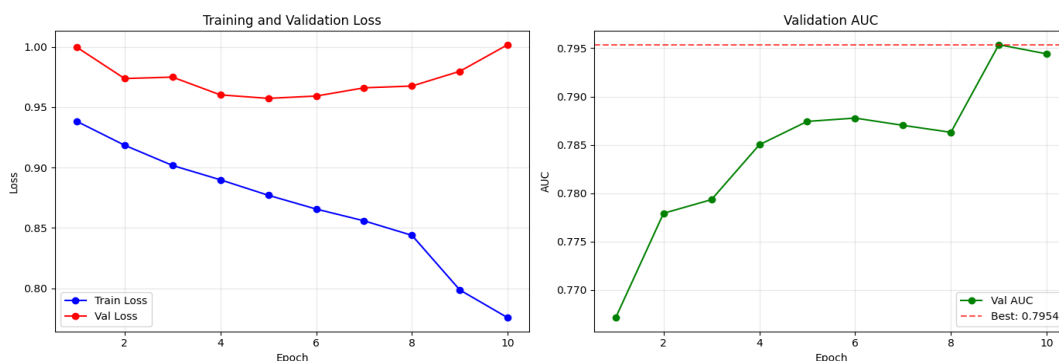


Рис. 3.30 – Крива функції втрат та AUC

Оскільки для незбалансованих медичних даних точність (Accuracy) не є показовою, основною метрикою було обрано площу під ROC-кривою (AUC). Максимальне значення Validation AUC склало 0.7954 на 9-й епісі. Це свідчить про високу здатність моделі ранжувати пацієнтів за ймовірністю наявності патології.

Детальний аналіз точності по окремих класах (Per-Class AUC) виявив суттєву варіативність, яка корелює з візуальною вираженістю патологій (рис. 3.31).

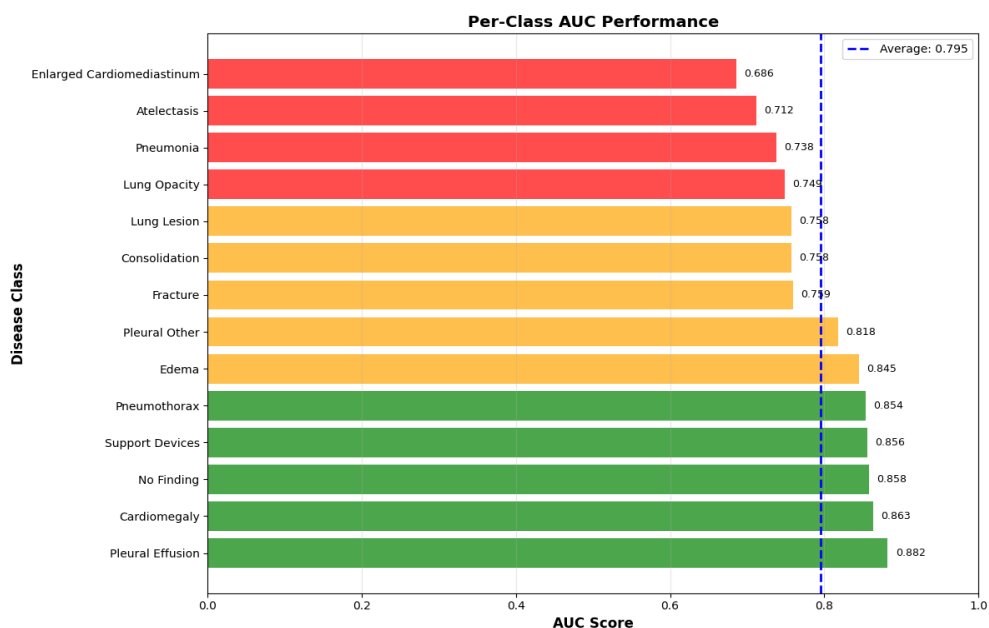


Рис. 3.31 – AUC по класам

Pleural Effusion (Випіт) – AUC 0.882. Ця патологія має чіткі геометричні ознаки (згладження реберно-діафрагмального синуса), які легко детектуються згортковою мережею.

Cardiomegaly (Збільшення серця) – AUC 0.863. Оцінка кардіоторакального індексу є класичною задачею для CNN.

No Finding (Норма) – AUC 0.858. Модель надійно відрізняє здорові знімки від патологічних, що є критично важливим для сортування (triage) пацієнтів.

Pneumonia (Пневмонія) – AUC 0.738. Інфільтрація при пневмонії часто має розмиті межі та візуально схожа на інші типи затемнень, що ускладнює класифікацію.

Enlarged Cardiomediatinum – AUC 0.686. Цей клас виявився найскладнішим, ймовірно через залежність візуалізації середостіння від позиціонування пацієнта (лежачи/стоячи) та якості вдиху.

Окрім дискримінаційної здатності, було проаналізовано калібрування моделі. Діаграма надійності (Reliability Diagram) показує, що крива калібрування проходить нижче ідеальної діагоналі у діапазоні ймовірностей (рис. 3.32). Це вказує на те, що модель є дещо самовпевненою (overconfident): прогнозовані ймовірності часто перевищують реальну частоту події. Такий ефект є типовим для глибоких нейромереж і може бути скоригований у майбутньому методами Temperature Scaling.

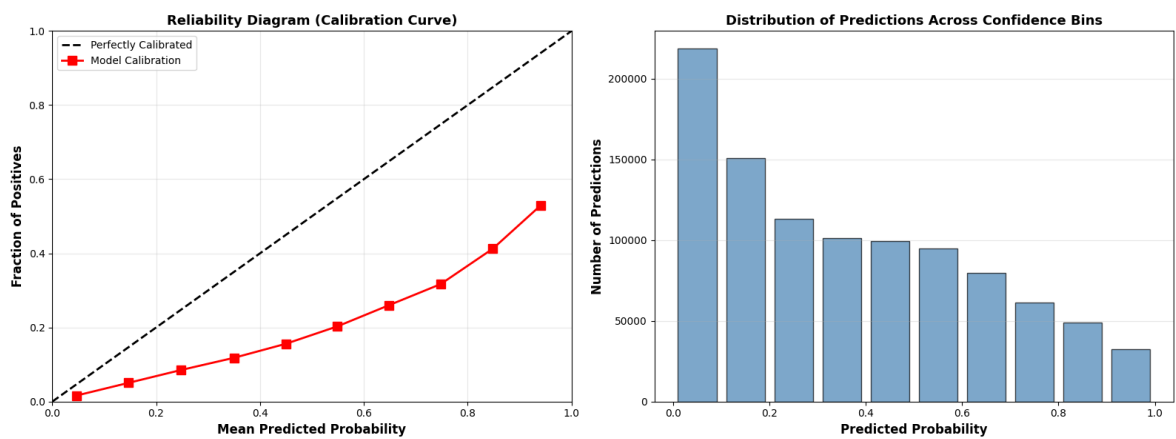


Рис. 3.32 – Діаграма надійності

Оцінка якості генерації радіологічних звітів проводилася на відкладеній вибірці з 100 досліджень з використанням автоматичних метрик обробки природної мови (табл. 3.1).

Таблиця 3.1

Метрики якості генерації тексту

Категорія меторик	Метрика	Значення (Mean ± Std)	Інтерпретація
Точність термінології	Keyword F1	0.8215 ± 0.32	Висока. Модель дуже точно використовує ключові медичні терміни (наприклад, «opacity», «effusion»), які були передані їй у вхідному промпті. Це підтверджує ефективність архітектури Concept Bottleneck.

Продовження таблиці 3.1

Семантична близькість	BERT Score F1	0.8752 ± 0.01	Висока. Глибокі семантичні ембедінги показують, що зміст згенерованого звіту майже ідентичний до еталонного, навіть якщо використано інші слова.
Структурна повнота	Section Completeness	39.0%	Середня. Модель не завжди генерує обидві секції ("Findings" та "Impression"), у 61% випадків обмежуючись лише детальним описом ("Findings"). Це вказує на необхідність більш жорсткого промпт-інжинірингу.
N-грамні перекриття	ROUGE-L	0.1979 ± 0.07	Низька.
	BLEU-4	0.0339 ± 0.04	Дуже низька.

Спостерігається значна розбіжність між високими показниками семантичних метрик (BERT Score, Keyword Accuracy) та низькими показниками N-грамних метрик (BLEU, ROUGE).

Низький BLEU (0.0339) є очікуваним і не свідчить про погану якість. BLEU вимагає дослівного збігу фраз. У медицині одну й ту ж патологію можна описати десятками способів (наприклад, «small effusion», «minimal fluid», «trace pleural liquid»). Модель генерує валідні, але лексично відмінні описи.

Високий Keyword F1 (0.82) є найважливішим показником для клінічної безпеки. Він гарантує фактичну точність: якщо візуальний модуль детектував «Пневмонію», текстовий модуль обов'язково включить цей термін у звіт, не «загубивши» діагноз.

Результати експериментів підтверджують життєздатність запропонованого гібридного підходу. Система демонструє високу діагностичну точність (AUC ~0.80) для критичних станів та здатність генерувати семантично коректні звіти. Виявлені обмеження (перенавчання класифікатора, неповна структура тексту) є технічними проблемами, що можуть бути вирішені шляхом

розширення навчальної вибірки та оптимізації промптів, і не спростовують загальну концепцію роботи.

Висновки до третього розділу

У третьому розділі магістерської роботи виконано програмну реалізацію та експериментальне дослідження розробленої системи автоматизованої генерації радіологічних звітів.

Систему реалізовано як вебдодаток на базі Django та сервісу машинного навчання BentoML, з використанням датасетів CheXpert та Indiana University Reports.

Експериментальне дослідження підтвердило ефективність гібридного методу (Concept Bottleneck). Модуль візуального аналізу (EfficientNet-B0 Late Fusion) досяг максимального AUC 0.7954, показавши високу точність для плеврального випоту (AUC 0.882) та кардіомегалії (AUC 0.863).

Модуль генерації тексту (BioMistral-7B) забезпечив високу семантичну точність (Keyword F1 0.82, BERT Score F1 0.87), гарантуючи коректність медичних термінів у звітах.

Реалізація механізму Human-in-the-loop та оптимізація моделі до Q5_K_M дозволили створити прототип, готовий до впровадження в клінічні умови.

ВИСНОВКИ

У магістерській роботі вирішено завдання автоматизації опису рентгенівських знімків грудної клітки шляхом розробки гібридної системи. Встановлено, що критичною проблемою сучасних генеративних моделей є «галюцинації», що робить End-to-End підходи непридатними для клінічної практики.

Для вирішення проблеми розроблено метод на основі архітектури Concept Bottleneck, який розділяє процес на виявлення ознак та генерацію тексту. Впровадження проміжного шару тегів створило «фільтр», що гарантує потрапляння у звіт лише підтверджених діагнозів.

Удосконалено метод візуального аналізу на базі EfficientNet-B0 з механізмом Late Fusion, що дозволило враховувати метадані пацієнта. Застосування CLANЕ та зваженої функції втрат забезпечило високу точність виявлення (AUC 0.882 для плеврального випоту, 0.863 для кардіомегалії).

Адаптацію мовної моделі для радіології виконано на базі BioMistral-7B методом QLoRA. Розроблена стратегія декодування дозволила досягти високої семантичної точності (BERT Score F1 0.87) та коректності термінології (Keyword F1 0.82).

Практична цінність роботи полягає у реалізації вебдодатку (Django + BentoML) з інтерфейсом Human-in-the-loop, що надає лікарю інструменти для верифікації патологій. Оптимізація моделі до формату Q5_K_M дозволила зменшити споживання відеопам'яті, зробивши можливим розгортання системи на стандартному обладнанні. Отримані результати підтверджують ефективність запропонованого підходу для підвищення продуктивності роботи радіолога.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Скаковський В.О., Савіцький Р.С., Фант М.О. Методи та алгоритми для виявлення аномалій. *Computer Science and Applied Mathematics*. 2025. №1, 2025. С. 81-88. <https://doi.org/10.26661/2786-6254-2025-1>.
2. V.O. Skakovskiy, R.S. Savitskiy, A.M. Kontsydaylo. Comprehensive testing strategies for anomaly detection machine learning models: performance, robustness, and interpretability. Тези XVIII Міжнародна науково-практична конференція магістрантів та аспірантів "Теоретичні та практичні дослідження молодих вчених", 19 – 22 листопада 2024. Харків: "Харківський політехнічний інститут", 2024. С. 13-14.
3. Brady A. P. Error and discrepancy in radiology: inevitable or avoidable? // *Insights into Imaging*. 2017. Vol. 8, no. 1. P. 171–182. <https://doi.org/10.1007/s13244-016-0534-1>.
4. Benson T., Grieve G. Principles of Health Interoperability: SNOMED CT, HL7 and FHIR. 4th ed. Springer International Publishing, 2021. <https://doi.org/10.1007/978-3-030-56883-2>.
5. Huang S.-C., Pareek A., Seyyedi S. [et al.]. Fusion of medical imaging and electronic health records using deep learning: a systematic review and implementation guidelines // *npj Digital Medicine*. 2020. Vol. 3. Article number: 136. <https://doi.org/10.1038/s41746-020-00341-z>.
6. Pillai A. S. Multi-Label Chest X-Ray Classification via Deep Learning // *Journal of Intelligent & Fuzzy Systems*. 2022. Vol. 43, no. 4. P. 1–15. <https://doi.org/10.3233/JIFS-213348>.
7. Huang G., Liu Z., Van Der Maaten L., Weinberger K. Q. Densely connected convolutional networks // *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 2017. P. 4700–4708. <https://doi.org/10.1109/CVPR.2017.243>.
8. Rajpurkar P., Irvin J., Zhu K. [et al.]. CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. 2017. <https://doi.org/10.48550/arXiv.1711.05225>.

9. Tan M., Le Q. V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks // International Conference on Machine Learning (ICML). PMLR, 2019. P. 6105–6114. URL: <http://proceedings.mlr.press/v97/tan19a.html>.
10. Shamshad F. [et al.]. Transformers in medical imaging: A survey // Medical Image Analysis. 2023. Vol. 88. Article 102859. <https://doi.org/10.1016/j.media.2023.102859>.
11. Saha A. [et al.]. A review on multimodal medical image fusion: Methods, applications, and challenges // Information Fusion. 2024. Vol. 101. Article 102001. <https://doi.org/10.1016/j.inffus.2023.102001>.
12. Ji Z. [et al.]. Survey of Hallucination in Natural Language Generation // ACM Computing Surveys. 2023. Vol. 55, iss. 12. P. 1–38. <https://doi.org/10.1145/3571730>.
13. Labrak Y., Bazoge A., Morin E. [et al.]. BioMistral: A Collection of Open-Source Pretrained Large Language Models for Medical Domains. 2024. <https://doi.org/10.48550/arXiv.2402.1037>.
14. Yuksekgonul M. [et al.]. Post-hoc Concept Bottleneck Models // International Conference on Learning Representations (ICLR). 2023. URL: <https://openreview.net/forum?id=s6Xq94xH1s>.
15. Johnson A. E. W., Pollard T. J., Berkowitz S. J. [et al.]. MIMIC-CXR, a de-identified publicly available database of chest radiographs with free-text reports // Scientific Data. 2019. Vol. 6. Article 317. <https://doi.org/10.1038/s41597-019-0322-0>.
16. Tu T. [et al.]. Towards Generalist Biomedical AI // NEJM AI. 2024. Vol. 1, iss. 3. <https://doi.org/10.1056/AIoa2300138>.
17. Koh P. W. [et al.]. Concept Bottleneck Models // International Conference on Machine Learning (ICML). PMLR, 2020. P. 5338–5348. URL: <http://proceedings.mlr.press/v119/koh20a.html>.
18. Selvaraju R. R., Cogswell M., Das A. [et al.]. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization // Proceedings of

the IEEE International Conference on Computer Vision (ICCV). 2017. P. 618–626.
<https://doi.org/10.1109/ICCV.2017.74>.

19. Hu E. J. [et al.]. LoRA: Low-Rank Adaptation of Large Language Models // International Conference on Learning Representations (ICLR). 2022. URL: <https://openreview.net/forum?id=nZeVKeeFYf9>.

20. Kreuzberger D., Kühl N., Koll S. Machine Learning Operations (MLOps): Overview, Definition, and Architecture // IEEE Access. 2023. Vol. 11. P. 31866–31879. <https://doi.org/10.1109/ACCESS.2023.3262138>.

21. Budd S., Robinson E. C., Kainz B. A survey on active learning and human-in-the-loop deep learning for medical image analysis // Medical Image Analysis. 2021. Vol. 71. Article 102062. <https://doi.org/10.1016/j.media.2021.102062>.

22. Jain S., Agrawal A., Saporta A. [et al.]. RadGraph: Extracting Clinical Entities and Relations from Radiology Reports // Proceedings of the Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track. 2021. Vol. 1. URL: https://openreview.net/forum?id=P4T7_4XzX8.

23. Kline A., Wang H., Li Y. [et al.]. Multimodal machine learning in precision health. 2022. <https://doi.org/10.48550/arXiv.2204.04777>.

24. Tan M., Le Q. V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Proceedings of the 36th International Conference on Machine Learning (ICML). 2019. Vol. 97. P. 6105–6114. URL: <http://proceedings.mlr.press/v97/tan19a.html>

25. Al-Waisy A. S., Al-Fahdawi S., Qahwaji R. [et al.]. A new deep learning framework for COVID-19 diagnosis and lesion segmentation in chest CT images // Multimedia Tools and Applications. 2023. Vol. 82. P. 13865–13888. <https://doi.org/10.1007/s11042-023-14923-x>.

26. Smit A., Jain S., Rajpurkar P. [et al.]. CheXbert: Combining Automatic Labelers and Expert Annotations for Accurate Radiology Report Labeling Using BERT // Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2020. P. 1500–1519. <https://doi.org/10.18653/v1/2020.emnlp-main.118>.

27. Agrawal M., Hegselmann S., Lang H. [et al.]. Large Language Models are Few-Shot Clinical Information Extractors // Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing. 2022. P. 1998–2022. <https://doi.org/10.18653/v1/2022.emnlp-main.130>.
28. Detrmers T., Pagnoni A., Holtzman A., Zettlemoyer L. QLoRA: Efficient Finetuning of Quantized LLMs // Advances in Neural Information Processing Systems (NeurIPS). 2023. Vol. 36. P. 10088–10115. <https://doi.org/10.48550/arXiv.2305.14314>.
29. Detrmers T., Pagnier A., Holtzman A., Zettlemoyer L. QLoRA: Efficient Finetuning of Quantized LLMs. // Advances in Neural Information Processing Systems (NeurIPS). 2024. Vol. 36. <https://doi.org/10.48550/arXiv.2305.14314>.
30. Demner-Fushman D., Kohli M. D., Rosenman M. B. et al. Preparing a collection of radiology examinations for distribution and retrieval. Journal of the American Medical Informatics Association. 2016. Vol. 23, Iss. 2. P. 304–310. <https://doi.org/10.1093/jamia/ocv080>.

ДОДАТКИ

Лістинг вихідного коду

```
import bentoml
import numpy as np
import torch
from torchvision import transforms
from PIL import Image
import ast
import cv2
import io
import base64

# Import the GGUF loader
from llama_cpp import Llama

import torch.serialization
from collections import OrderedDict
import cloudpickle.cloudpickle

# Keep standard layers safe
from torch.nn.modules.container import Sequential
from torch.nn.modules.conv import Conv2d
from torch.nn.modules.batchnorm import BatchNorm2d
from torch.nn.modules.activation import ReLU, SiLU, Sigmoid # Added SiLU (Swish)
from torch.nn.modules.pooling import MaxPool2d, AvgPool2d, AdaptiveAvgPool2d
from torch.nn.modules.linear import Linear
from torch.nn.modules.dropout import Dropout
from torch.nn.modules.module import Module
import torch.nn.functional as F

# Add all classes to the safe list including cloudpickle functions
torch.serialization.add_safe_globals(
    [
        # PyTorch modules
        Sequential,
        Conv2d,
        BatchNorm2d,
        ReLU,
        SiLU, # Used by EfficientNet
        Sigmoid,
        MaxPool2d,
        AvgPool2d,
        AdaptiveAvgPool2d,
        Linear,
        Dropout,
        Module, # Base Module class
        # Python built-ins
        OrderedDict,
        type,
```

```

dict,
list,
tuple,
set,
frozenset,
bytes,
bytearray,
str,
int,
float,
bool,
# Cloudpickle functions
cloudpickle.cloudpickle._make_skeleton_class,
cloudpickle.cloudpickle._make_skeleton_enum,
cloudpickle.cloudpickle._make_function,
cloudpickle.cloudpickle._class_setstate,
cloudpickle.cloudpickle._function_setstate,
cloudpickle.cloudpickle._builtin_type,
cloudpickle.cloudpickle._get_cell_contents,
cloudpickle.cloudpickle._make_cell,
]
)

class ApplyCLAHE:
    def __init__(self, clip_limit=2.0, tile_grid_size=(8, 8)):
        self.clip_limit = clip_limit
        self.tile_grid_size = tile_grid_size
        self.clahe = cv2.createCLAHE(
            clipLimit=self.clip_limit, tileGridSize=self.tile_grid_size
        )

    def __call__(self, img: Image.Image) -> Image.Image:
        img_ycbcr = img.convert("YCbCr")
        img_np = np.array(img_ycbcr)

        y_channel = img_np[:, :, 0]
        y_channel_clahe = self.clahe.apply(y_channel)
        img_np[:, :, 0] = y_channel_clahe

        size = (img_np.shape[1], img_np.shape[0])
        img_clahe = Image.new("YCbCr", size)
        img_clahe.frombytes(img_np.tobytes())

        return img_clahe.convert("RGB")

class GradCAM:
    def __init__(self, model, target_layer):
        self.model = model
        self.target_layer = target_layer

```

```

self.gradients = None
self.activations = None

self.target_layer.register_forward_hook(self.save_activation)
self.target_layer.register_full_backward_hook(self.save_gradient)

def save_activation(self, module, input, output):
    self.activations = output.detach()

def save_gradient(self, module, grad_input, grad_output):
    self.gradients = grad_output[0].detach()

def generate_cam(self, input_tensor, target_class):
    self.model.eval()
    # Use stored meta_tensor if available
    meta_tensor = getattr(self, "meta_tensor", None)
    if meta_tensor is not None:
        output = self.model(input_tensor, meta_tensor)
    else:
        output = self.model(input_tensor)
    self.model.zero_grad()

    target = output[0, target_class]
    target.backward()

    weights = torch.mean(self.gradients, dim=[2, 3], keepdim=True)
    cam = torch.sum(weights * self.activations, dim=1, keepdim=True)
    cam = F.relu(cam)

    cam = cam.squeeze().cpu().numpy()
    cam = (cam - cam.min()) / (cam.max() - cam.min() + 1e-8)
    return cam

def generate_multi_class_cam(self, input_tensor, target_classes, class_names):
    class_cams = {}
    combined_cam = np.zeros((input_tensor.shape[2], input_tensor.shape[3]))

    for class_idx in target_classes:
        cam = self.generate_cam(input_tensor.clone(), class_idx)
        cam_resized = cv2.resize(
            cam, (input_tensor.shape[3], input_tensor.shape[2])
        )
        class_cams[class_names[class_idx]] = cam_resized
        combined_cam = np.maximum(combined_cam, cam_resized)

    return combined_cam, class_cams

def get_bounding_box_from_cam(cam, threshold=0.5):
    mask = (cam > threshold).astype(np.uint8)
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

```

```

    if not contours:
        return None
    largest_contour = max(contours, key=cv2.contourArea)
    return cv2.boundingRect(largest_contour)

def apply_heatmap_only(img, class_cams, class_names, alpha=0.4):
    img_array = np.array(img.convert("RGB"))
    h, w = img_array.shape[:2]
    combined_cam = np.zeros((h, w), dtype=np.float32)

    for idx, class_name in enumerate(class_names):
        if class_name not in class_cams:
            continue
        cam = class_cams[class_name]
        cam_resized = cv2.resize(cam, (w, h))
        combined_cam = np.maximum(combined_cam, cam_resized)

    heatmap = cv2.applyColorMap(np.uint8(255 * combined_cam), cv2.COLORMAP_JET)
    heatmap = cv2.cvtColor(heatmap, cv2.COLOR_BGR2RGB)
    overlaid = (alpha * heatmap + (1 - alpha) * img_array).astype(np.uint8)
    return Image.fromarray(overlaid)

def apply_boxes_with_labels(img, class_cams, class_names, threshold=0.5):
    img_array = np.array(img.convert("RGB"))
    h, w = img_array.shape[:2]
    overlay = img_array.copy()
    colors = [
        (255, 0, 0),
        (0, 255, 0),
        (0, 0, 255),
        (255, 255, 0),
        (255, 0, 255),
        (0, 255, 255),
    ]

    for idx, class_name in enumerate(class_names):
        if class_name not in class_cams:
            continue
        cam = class_cams[class_name]
        cam_resized = cv2.resize(cam, (w, h))
        bbox = get_bounding_box_from_cam(cam_resized, threshold)

        if bbox:
            x, y, box_w, box_h = bbox
            color = colors[idx % len(colors)]
            cv2.rectangle(overlay, (x, y), (x + box_w, y + box_h), color, 3)

            label = class_name
            font = cv2.FONT_HERSHEY_SIMPLEX

```

```

        font_scale = 0.6
        thickness = 2
        (text_w, text_h), _ = cv2.getTextSize(label, font, font_scale, thickness)

        label_y = y - 10 if y - 10 > text_h + 10 else y + box_h + text_h + 10
        cv2.rectangle(
            overlay, (x, label_y - text_h - 4), (x + text_w, label_y + 4), color, -1
        )
        cv2.putText(
            overlay,
            label,
            (x, label_y),
            font,
            font_scale,
            (255, 255, 255),
            thickness,
            cv2.LINE_AA,
        )

    return Image.fromarray(overlay)

@bentoml.service(
    name="chexpert_runner_service",
    resources={"cpu": 1, "memory": "2Gi"},
    traffic={"timeout": 60},
)
class CheXpertRunner:
    def __init__(self) -> None:
        self.device = "cpu"
        print(f"CheXpertRunner forced to use device: {self.device}")

        bento_model = bentoml.models.get("chexpert_vision_fusion:latest")
        class_names_str = bento_model.info.labels["class_names"]
        self.class_names = ast.literal_eval(class_names_str)
        print("Classes loaded:", self.class_names)

        model_path = bento_model.path_of("saved_model.pt")
        self.model = torch.load(
            model_path, map_location=self.device, weights_only=False
        )
        self.model.to(self.device)
        self.model.eval()

        self.transform = transforms.Compose(
            [
                transforms.Grayscale(num_output_channels=3),
                ApplyCLAHE(clip_limit=2.0, tile_grid_size=(8, 8)),
                transforms.Resize((240, 240)),
                transforms.ToTensor(),
                transforms.Normalize(

```

```

        mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]
    ),
]
)

target_layer = None
if hasattr(self.model, "conv_head"):
    target_layer = self.model.conv_head

if target_layer is None:
    for module in self.model.modules():
        if isinstance(module, torch.nn.Conv2d):
            target_layer = module # Will grab the last one

self.gradcam = GradCAM(self.model, target_layer) if target_layer else None
print(f"CheXpertRunner loaded. GradCAM ready: {self.gradcam is not None}")

@bentoml.api(batchable=True, max_batch_size=16)
def classify(
    self,
    images: list[Image.Image],
    ages: list[float] = None,
    sexes: list[str] = None,
) -> list[dict]:
    batch_tensor = torch.stack([self.transform(img) for img in images]).to(
        self.device
    )

    # Prepare metadata batch
    batch_size = len(images)
    if ages is None:
        ages = [None] * batch_size
    if sexes is None:
        sexes = [None] * batch_size

    meta_list = []
    for age, sex in zip(ages, sexes):
        age_norm = (age / 100.0) if age is not None else 0.5
        sex_enc = {"Male": 0.0, "Female": 1.0}.get(sex, 0.5)
        meta_list.append([age_norm, sex_enc])

    meta_tensor = torch.tensor(meta_list, dtype=torch.float32).to(self.device)

    with torch.no_grad():
        logits = self.model(batch_tensor, meta_tensor)
        probs = torch.sigmoid(logits).cpu().numpy()
        binary_preds = (probs > 0.5).astype(int)

    result = [
        {
            "predictions": binary_preds[i].tolist(),

```

```

        "probabilities": probs[i].tolist(),
    }
    for i in range(len(binary_preds))
]
return result

@bentoml.api
def classify_with_gradcam(
    self, image: Image.Image, age: float = None, sex: str = None
) -> dict:
    input_tensor = self.transform(image).unsqueeze(0).to(self.device)

    age_norm = (age / 100.0) if age is not None else 0.5 # Default to median
    sex_enc = {"Male": 0.0, "Female": 1.0}.get(sex, 0.5)

    meta_tensor = torch.tensor([[age_norm, sex_enc]], dtype=torch.float32).to(
        self.device
    )

    with torch.no_grad():
        logits = self.model(input_tensor, meta_tensor)
        probs = torch.sigmoid(logits).cpu().numpy()[0]
        binary_preds = (probs > 0.5).astype(int)

    gradcam_base64 = None
    if self.gradcam is not None:
        detected_classes = [i for i, pred in enumerate(binary_preds) if pred == 1]
        detected_names = [self.class_names[i] for i in detected_classes]

        if detected_classes:
            input_tensor.requires_grad = True
            # Store meta_tensor for GradCAM (it doesn't need gradients)
            self.gradcam.meta_tensor = meta_tensor
            combined_cam, class_cams = self.gradcam.generate_multi_class_cam(
                input_tensor, detected_classes, self.class_names
            )

            gradcam_heatmap = apply_heatmap_only(
                image, class_cams, detected_names, alpha=0.4
            )
            gradcam_boxes = apply_boxes_with_labels(
                image, class_cams, detected_names, threshold=0.5
            )

            buffered_heatmap = io.BytesIO()
            gradcam_heatmap.save(buffered_heatmap, format="PNG")
            gradcam_heatmap_base64 = base64.b64encode(
                buffered_heatmap.getvalue()
            ).decode("utf-8")

            buffered_boxes = io.BytesIO()

```

```

        gradcam_boxes.save(buffered_boxes, format="PNG")
        gradcam_boxes_base64 = base64.b64encode(
            buffered_boxes.getvalue()
        ).decode("utf-8")

        gradcam_base64 = {
            "heatmap": gradcam_heatmap_base64,
            "boxes": gradcam_boxes_base64,
        }

    return {
        "predictions": binary_preds.tolist(),
        "probabilities": probs.tolist(),
        "gradcam_image": gradcam_base64,
    }

@bentoml.service(
    name="biomistral_runner_service",
    resources={"cpu": "4", "memory": "16Gi"},
    traffic={"timeout": 300},
)

class BioMistralRunner:
    def __init__(self) -> None:
        MODEL_PATH = "models/biomistral-Q5_K_M.gguf"
        print(f>Loading GGUF model from {MODEL_PATH}...")
        try:
            self.model = Llama(
                model_path=MODEL_PATH,
                n_ctx=2048,
                n_gpu_layers=-1, # Metal GPU offloading
                verbose=True,
            )
        except Exception as e:
            print("---- FAILED TO LOAD GGUF MODEL ----")
            raise

        vision_model_info = bentoml.models.get("chexpert_vision_fusion:latest").info
        class_names_str = vision_model_info.labels["class_names"]
        self.class_names = ast.literal_eval(class_names_str)
        print("BioMistralRunner loaded.")

    @bentoml.api(batchable=False)
    def generate(self, binary_preds: np.ndarray) -> str:
        chexpert_labels = [
            name for pred, name in zip(binary_preds, self.class_names) if pred == 1
        ]
        if not chexpert_labels:
            chexpert_labels = ["No Finding"]
        labels_str = "; ".join(chexpert_labels)

```

```
        prompt = f"""Given the following medical labels, generate a professional
radiological report with 'Findings' and 'Impression' sections.
```

```
### Labels:
{labels_str}
```

```
### Report:
"""
```

```
        output = self.model(
            prompt,
            max_tokens=512,
            temperature=0.7,
            top_k=50,
            top_p=0.9,
            stop=["###"],
        )
        return output["choices"][0]["text"].strip()
```

```
@bentoml.service(
    name="radiology_report_service",
    traffic={"timeout": 300},
)
```

```
class RadiologyReportService:
```

```
    vision_runner = bentoml.depends(CheXpertRunner)
    llm_runner = bentoml.depends(BioMistralRunner)
```

```
    def __init__(self):
        model_info = bentoml.models.get("chexpert_vision_fusion:latest").info
        class_names_str = model_info.labels["class_names"]
        self.class_names = ast.literal_eval(class_names_str)
```

```
@bentoml.api
```

```
    def create_report(
        self, image: Image.Image, age: float = None, sex: str = None
    ) -> dict:
```

```
        result_list = self.vision_runner.classify(
            [image],
            ages=[age] if age is not None else None,
            sexes=[sex] if sex else None,
        )
```

```
        result = result_list[0]
        binary_preds = result["predictions"]
        probabilities = result["probabilities"]
```

```
        report_text = self.llm_runner.generate(binary_preds)
```

```
        findings = [
            name for pred, name in zip(binary_preds, self.class_names) if pred == 1
        ]
        findings_confidence = {
```

```

        name: float(prob)
        for pred, name, prob in zip(binary_preds, self.class_names, probabilities)
        if pred == 1
    }

    return {
        "findings_detected": findings if findings else ["No Finding"],
        "findings_confidence": findings_confidence,
        "report": report_text,
    }

@bentoml.api
def analyze_vision(
    self, image: Image.Image, age: float = None, sex: str = None
) -> dict:
    result = self.vision_runner.classify_with_gradcam(image, age=age, sex=sex)
    binary_preds = result["predictions"]
    probabilities = result["probabilities"]
    gradcam_image = result["gradcam_image"]

    findings = [
        name for pred, name in zip(binary_preds, self.class_names) if pred == 1
    ]
    findings_confidence = {
        name: float(prob)
        for pred, name, prob in zip(binary_preds, self.class_names, probabilities)
        if pred == 1
    }

    return {
        "findings_detected": findings if findings else ["No Finding"],
        "findings_confidence": findings_confidence,
        "gradcam_image": gradcam_image,
    }

@bentoml.api
def generate_report(self, findings: list[str]) -> dict:
    binary_preds = np.zeros(len(self.class_names), dtype=int)
    for finding in findings:
        if finding in self.class_names:
            idx = self.class_names.index(finding)
            binary_preds[idx] = 1

    report_text = self.llm_runner.generate(binary_preds)
    return {"report": report_text}

```