Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелект
Кафедра математичного моделювання та аналізу даних

До захисту допущено
Кафедрою ММАД протокол №___ від_____

завідувач кафедри _____ Володимир СТРУКОВ
(підпис) (ім'я, прізвище)
«___»_____ 2026 р.

Кваліфікаційна робота
здобувача першого (бакалаврського) рівня вищої освіти

Розробка програмного забезпечення медичного моніторингу із інтеграцією методів машинного навчання

(назва роботи)

Спеціальність (спеціалізація) 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки

Виконавець _____ Даніїл Павлусенко
(підпис) (ім'я, прізвище)
Науковий керівник _____ Володимир Донець
(підпис) (ім'я, прізвище)

Харків – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Кафедра математичного моделювання та аналізу даних

ЗАТВЕРДЖУЮ

Завідувач кафедри ММАД

_____ Володимир Струков

підпис ім’я, прізвище

“____” _____2026 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувача першого (бакалаврського) рівня вищої освіти

(першого (бакалаврського) / другого (магістерського))

Павлусенко Данііл Віталійович

(прізвище, ім’я, по батькові здобувача)

Спеціальність (спеціалізація) 122 Комп'ютерні науки

(код та найменування спеціальності; спеціалізації спеціальності)

Освітня програма Комп'ютерні науки

(назва освітньої програми)

1. Тема роботи: Розробка програмного забезпечення медичного моніторингу із інтеграцією методів машинного навчання

керівник роботи Донець Володимир Віталійович, доктор філософії

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по Університету від _____2026 року №_____

2. Строк здобувачем роботи “____” _____2026 року

3. Вихідні дані до роботи: Набір анотованих рентгенограм грудної клітки; технічне завдання на розробку медичного клієнт-серверного вебзастосунку; вимоги до архітектури (REST API, JSON) та базового технологічного стека (Python, FastAPI, React, PostgreSQL); вимоги щодо

інтеграції методів пояснюваного ШІ (Grad-CAM) і проведення автоматизованого тестування (Selenium, Postman).

4. Перелік питань, які потрібно розробити:

1. ознайомитись з особливостями автоматизованого аналізу медичних зображень та вимогами до програмного забезпечення у сфері охорони здоров'я;
2. провести огляд існуючих методів глибокого навчання для класифікації рентгенограм та сучасних архітектурних рішень для медичних вебзастосунків;
3. познайомитися з алгоритмом роботи і методиками пояснюваного штучного інтелекту (зокрема алгоритмом Grad-CAM), створити архітектурну та інформаційну модель програмного комплексу;
4. провести програмну реалізацію та експериментальне дослідження розробленого діагностичного рішення, оцінити його швидкодію та надійність механізмів захисту.

5. План роботи

№ з/п	Назви етапів роботи	Строк виконання етапів роботи	Примітка
1	Визначення проблематики та аналіз існуючих систем діагностики захворювань легень	01.03 – 07.03	
2	Дослідження архітектур згорткових нейронних мереж та методів Transfer Learning	07.03 – 14.03	
3	Аналіз методів інтерпретованості моделей машинного навчання (XAI) та обґрунтування вибору Grad-CAM	14.03 – 18.03	
4	Формування вимог до програмного забезпечення та проєктування архітектури системи	18.03 – 22.03	
5	Формування та підготовка набору даних рентгенологічних знімків легень	22.03 – 25.03	
6	Порівняльний аналіз архітектур VGG16, ResNet50 та EfficientNetB3	25.03 – 01.04	
7	Розробка та навчання моделі класифікації із застосуванням Fine-tuning	01.04 – 07.04	
8	Реалізація методу Grad-CAM для візуальної інтерпретації рішень моделі	07.04 – 10.04	

9	Проєктування та реалізація бази даних системи	10.04 – 14.04	
10	Розробка серверної частини системи на базі FastAPI	14.04 – 21.04	
11	Розробка клієнтської частини на базі React	21.04 – 30.04	
12	Реалізація системи генерації медичних звітів у форматах PDF та JSON з метаданими моделі	30.04 – 05.05	
13	Тестування системи та оцінка якості моделі на тестовій вибірці	05.05 – 10.05	
14	Порівняння результатів з існуючими аналогами	10.05 – 13.05	
15	Підбиття підсумків, визначення напрямків для подальшого покращення системи та оформлення звіту.	13.05 – 30.05	

6. Дата видачі завдання ____ _____ 2026 р.

Здобувач вищої освіти _____ Д. В. Павлусенко

(підпис) (ініціали, прізвище)

Керівник роботи _____ В. В. Донець

АНОТАЦІЯ

Кваліфікаційна робота містить: 116 с., 4 розділи, 47 рис., 12 табл., 3 додаток, 75 джерел.

Мета роботи – поліпшення точності та надійності автоматизованої ідентифікації захворювань легень на основі аналізу рентгенівських знімків шляхом розробки програмного комплексу із застосуванням методів глибокого навчання та засобів пояснюваного штучного інтелекту.

Методи дослідження і перелік використаних програмно-апаратних рішень: методи комп'ютерного зору, технологія Transfer Learning, алгоритм Grad-CAM. Використано стек технологій: мови програмування Python, фреймворки FastAPI та React, бібліотеку машинного навчання PyTorch, СКБД PostgreSQL.

Результати та їх новизна: створено клієнт-серверний вебзастосунок для скринінгу патологій легень. Навчено модель EfficientNetB3 з точністю класифікації 92% та інтегровано модуля генерації теплових карт Grad-CAM. Новизна полягає у комплексному поєднанні високої точності розпізнавання з автоматичною генерацією звітів у форматах PDF та машиннозчитуваному JSON згідно з принципом прозорості штучного інтелекту.

Рекомендації щодо використання результатів роботи: програмний комплекс рекомендовано для впровадження у радіологічних відділеннях медичних закладів як надійний інструмент підтримки прийняття лікарських рішень.

КЛЮЧОВІ СЛОВА: МАШИННЕ НАВЧАННЯ, КОМП'ЮТЕРНИЙ ЗІР, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, РЕНТГЕНОГРАМА, ПНЕВМОНІЯ, ТУБЕРКУЛЬОЗ, ДІАГНОСТИКА, АЛГОРИТМ, ТЕПЛОВА КАРТА.

ANNOTATION

Qualifying work contains: 116 p., 4 chapters, 47 fig., 12 tab., 3 appendix, 75 sources.

Purpose of the work is to improve the accuracy and reliability of automated identification of lung diseases based on the analysis of X-ray images by developing a software system using deep learning methods and explainable artificial intelligence.

Research methods and hardware/software solutions: computer vision methods, Transfer Learning technology, Grad-CAM algorithm. Technology stack: Python programming language, FastAPI and React frameworks, PyTorch machine learning library, PostgreSQL DBMS.

Results and novelty: a client-server web application for lung pathology screening was created. The EfficientNetB3 model was trained with a classification accuracy of 92%, and the Grad-CAM heatmap generation subsystem was integrated. The novelty lies in the complex combination of high recognition accuracy with the automatic generation reports in PDF and machine-readable JSON formats according to the AI transparency principle.

Recommendations for use: the software complex is recommended for implementation in radiological departments of medical institutions as a reliable medical decision support tool.

KEYWORDS: MACHINE LEARNING, COMPUTER VISION, CONVOLUTIONAL NEURAL NETWORK, RADIOGRAPH, PNEUMONIA, TUBERCULOSIS, DIAGNOSIS, ALGORITHM, HEATMAP.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

БД – база даних

МІС – медична інформаційна система

ПЗ – програмне забезпечення

ШІ – штучний інтелект

API (Application Programming Interface) – прикладний програмний інтерфейс

AUC (Area Under the Curve) – площа під кривою (показник точності)

CAD (Computer-Aided Diagnosis) – система комп'ютерної підтримки діагностики

CNN (Convolutional Neural Network) – згорткова нейронна мережа

DOM (Document Object Model) – об'єктна модель документа

E2E (End-to-End) – наскрізне тестування

EDA (Exploratory Data Analysis) – розвідувальний аналіз даних

FHIR (Fast Healthcare Interoperability Resources) – стандарт обміну медичними даними

FN (False Negative) – хибнонегативний результат

FP (False Positive) – хибнопозитивний результат

Grad-CAM (Gradient-weighted Class Activation Mapping) – градієнтне зважування карт активації

JSON (JavaScript Object Notation) – текстовий формат обміну даними

JWT (JSON Web Token) – вебтокен формату JSON

PDF (Portable Document Format) – портативний формат документа

TN (True Negative) – істинно негативний результат

TP (True Positive) – істинно позитивний результат

UI (User Interface) – інтерфейс користувача

XAI (Explainable Artificial Intelligence) – пояснюваний штучний інтелект

ВСТУП

Актуальність теми. У сучасному світі стрімкий розвиток інформаційних технологій сприяє їх активному впровадженню в медичну галузь, зокрема для автоматизації процесів діагностики захворювань. Концепція цифрової медицини (e-Health) вимагає створення інтелектуальних інструментів, здатних швидко та якісно обробляти великі масиви клінічних даних. Одним із найбільш перспективних напрямків у цій сфері є застосування методів машинного та глибокого навчання (Deep Learning) для аналізу медичних зображень. Це дозволяє не лише підвищити точність діагностики, але й суттєво зменшити когнітивне навантаження на медичний персонал.

Захворювання органів дихання, такі як вірусна та бактеріальна пневмонія, туберкульоз, а також ускладнення, викликані COVID-19, залишаються глобальною проблемою для систем охорони здоров’я у всьому світі. Вони характеризуються високим рівнем поширеності, швидким перебігом та значною смертністю, особливо серед вразливих груп населення. За даними Всесвітньої організації охорони здоров'я (ВООЗ) [1], респіраторні захворювання щорічно призводять до мільйонів випадків госпіталізації. Несвоєчасна або помилкова діагностика на ранніх стадіях значно ускладнює процес лікування та збільшує економічний тягар на медичні установи.

Основним інструментом первинної діагностики, що характеризується швидкістю та доступністю, є рентгенографія грудної клітки. Проте аналіз рентгенівських знімків є складним завданням, яке значною мірою залежить від кваліфікації лікаря-рентгенолога, його досвіду та поточного функціонального стану. В умовах великого потоку пацієнтів лікарі часто стикаються з явищем «зорової втоми», що призводить до зростання відсотка суб’єктивних помилок. З огляду на це, створення автоматизованих систем підтримки прийняття рішень (Computer-Aided Diagnosis, CAD) стає об'єктивною необхідністю [2].

У цьому контексті революційні результати демонструють методи комп’ютерного зору на базі згорткових нейронних мереж (CNN). Завдяки

своїй здатності автоматично вилучати ієрархічні ознаки безпосередньо з пікселів зображення, такі архітектури, як ResNet [3], VGG [4] та EfficientNet [5], здатні розпізнавати тонкі патологічні зміни, які можуть бути непомітними для людського ока. Використання технології перенесення навчання (Transfer Learning) дозволяє ефективно адаптувати ці потужні моделі для медичних задач навіть за умов обмеженої кількості розмічених клінічних даних.

Однак, незважаючи на високу точність, впровадження алгоритмів глибокого навчання у реальну клінічну практику гальмується суттєвим недоліком – їх низькою інтерпретованістю. Ця проблема, відома як феномен «чорної скрині» (black box), означає, що процес формування висновку моделлю є прихованим від користувача. З медичної та юридичної точок зору лікар не може призначати лікування, покладаючись виключно на математичний алгоритм без розуміння логіки його роботи.

Для вирішення цієї проблеми активно розвивається напрям пояснюваного штучного інтелекту (Explainable AI, XAI). Застосування градієнтних методів локалізації, зокрема алгоритму Grad-CAM (Gradient-weighted Class Activation Mapping) [6], дозволяє генерувати теплові карти активації, що візуально підсвічують ділянки рентгенограми, які стали вирішальними для постановки діагнозу. Це дозволяє підвищити прозорість роботи моделі та робить її надійним інструментом підтримки прийняття рішень.

Окрім математичної складової, критично важливою є інженерна реалізація програмної архітектури. Сучасне медичне програмне забезпечення повинно мати клієнт-серверну архітектуру, надійну базу даних та можливості генерації структурованих звітів для їх подальшої інтеграції у медичні системи. Таким чином, розробка комплексного веб-застосунку для автоматизованої діагностики захворювань легень із використанням методів глибокого навчання та засобів XAI є актуальним науково-практичним завданням.

Мета і завдання дослідження. Метою роботи є поліпшення точності та надійності автоматизованої ідентифікації захворювань легень на основі

аналізу рентгенівських знімків шляхом розробки програмного комплексу із застосуванням методів глибокого навчання та засобів пояснюваного штучного інтелекту.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати предметну область та існуючі підходи до автоматизованої діагностики захворювань органів дихання;
- дослідити сучасні архітектури згорткових нейронних мереж та методи забезпечення їх інтерпретованості;
- виконати підготовку, аугментацію та попередню обробку набору медичних даних;
- реалізувати моделі класифікації на базі архітектур ResNet, VGG та EfficientNet та провести їх порівняльний аналіз;
- здійснити донавчання (fine-tuning) та оптимізацію найбільш ефективної моделі;
- інтегрувати метод Grad-CAM для генерації візуальних пояснень (теплових карт);
- спроєктувати базу даних та розробити серверну і клієнтську частини вебзастосунку;
- реалізувати модуль генерації результатів у форматах PDF та JSON;
- провести експериментальне тестування розробленого програмного забезпечення та перевірити точність ML-моделей і ефективність методів інтерпретації.

Засоби та критерії досягнення мети. Засобами досягнення поставленої мети є використання сучасних варіацій моделей згорткових нейронних мереж (зокрема EfficientNet) у поєднанні з технологією Transfer Learning та алгоритмом локалізації Grad-CAM. Критеріями досягнення мети є:

1. отримання високих показників точності класифікації (Accuracy) на тестовій вибірці;

2. успішна генерація візуально інтерпретованих теплових карт, що обґрунтовують рішення моделі;
3. коректне функціонування розробленого клієнт-серверного вебзастосунку та модуль генерації звітів.

Об’єкт дослідження – процес автоматизованої діагностики захворювань легень на основі рентгенологічних зображень.

Предмет дослідження – методи та алгоритми глибокого навчання, інструменти пояснюваного штучного інтелекту (XAI) та програмні засоби для їх інтеграції у веб-орієнтовані продукти.

Методи дослідження. У роботі використано: методи комп’ютерного зору та глибокого навчання (для класифікації знімків); технологію Transfer Learning (для оптимізації навчання моделей); методи Explainable AI (для візуалізації прийняття рішень); статистичні методи (для оцінки якості класифікації).

Практичне значення роботи полягає у створенні повноцінного програмного комплексу, який може бути розгорнутий у закладах охорони здоров’я як інструмент підтримки прийняття рішень лікаря-радіолога. Наявність модуля генерації звітів та візуалізації через Grad-CAM підвищує довіру медичного персоналу до алгоритмів ШІ, а реалізоване API дозволяє інтегрувати розроблений програмний комплекс з існуючими медичними інформаційними системами.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР НАПРЯМКУ ДОСЛІДЖЕНЬ

1.1 Проблематика діагностики захворювань легень та ціна лікарської помилки

Захворювання органів дихання залишаються однією з найактуальніших проблем сучасної медицини, що обумовлено їх широким розповсюдженням, швидким перебігом та значним рівнем смертності. До найбільш небезпечних патологій належать вірусна та бактеріальна пневмонія, туберкульоз, а також ускладнення, викликані COVID-19. За даними Всесвітньої організації охорони здоров'я (ВООЗ) [1], респіраторні інфекції є однією з провідних причин госпіталізації та летальних наслідків у світі, створюючи колосальне навантаження на національні системи охорони здоров’я.

Основним інструментом первинного скринінгу та діагностики патологій легень залишається рентгенографія грудної клітки, що зумовлено її доступністю, відносною дешевизною та швидкістю отримання результатів. Проте інтерпретація отриманих зображень є надзвичайно складним завданням, оскільки на перший погляд рентгенограми здорової людини та пацієнта з пневмонією можуть виглядати майже ідентичними. Цей ефект візуальної схожості проілюстровано на рисунку 1.1, де навіть за наявності патології знімки важко відрізнити від норми без спеціалізованої підготовки. Особливо критичною ця проблема стає на початкових стадіях захворювання, коли патологічні зміни є субміліметровими та майже невидимими для неозброєного ока.

Саме ця візуальна неоднозначність є першопричиною виникнення діагностичних помилок, які повністю лягають на плечі лікаря-рентгенолога. Головною проблемою сучасної радіології є стрімке зростання обсягів медичних даних на тлі дефіциту кваліфікованих кадрів. За даними досліджень у галузі медичної візуалізації, середній рівень діагностичних помилок (discrepancy rate) під час ручного аналізу рентгенограм становить близько

3–5% у щоденній практиці, але під час ретроспективного аналізу складних патологій може сягати 20–30% [7].

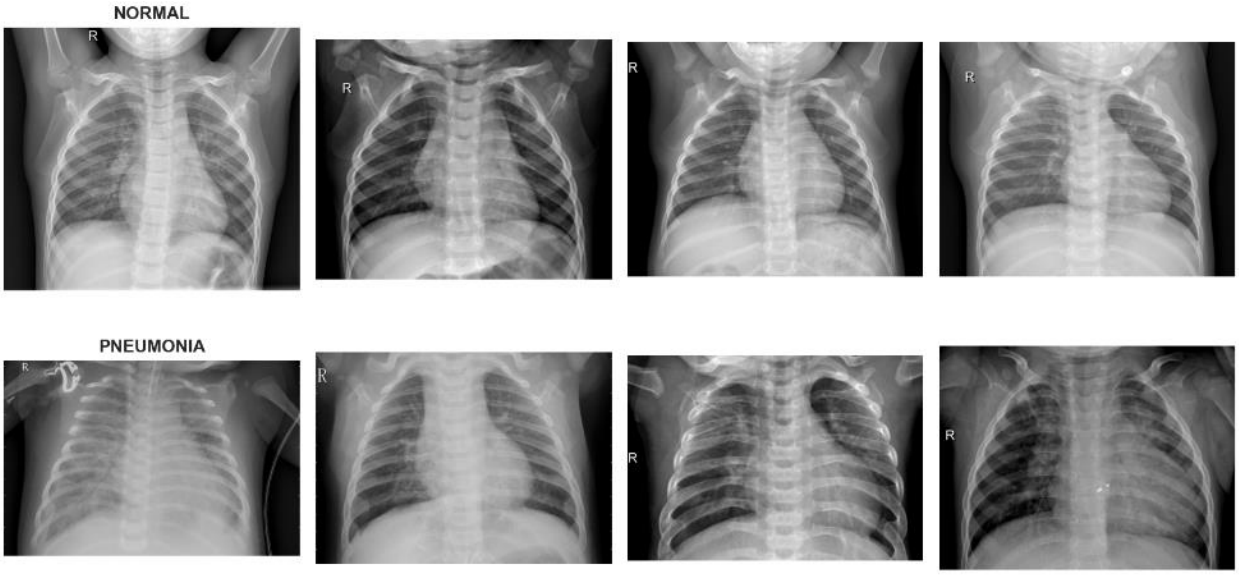

Рисунок 1.1 – Приклади рентгенівських знімків легень у нормі та при патології

Згідно з дослідженнями [8], лікарські помилки в радіології фундаментально поділяються на дві основні категорії: когнітивні (коли патологічна зона була виявлена лікарем на знімку, але неправильно інтерпретована) та перцептивні (коли лікар взагалі не помітив наявну патологію). Статистика свідчить, що більшість хибних діагнозів (до 80%) припадає саме на перцептивні помилки. Їх головною причиною є явище «зорової втоми», деконцентрація та надмірне когнітивне навантаження.

В умовах щільного потоку пацієнтів лікарі змушені аналізувати сотні рентгенограм протягом однієї робочої зміни. Дослідження ергономіки роботи медичного персоналу показують пряму кореляцію між часом доби та точністю діагностики, що особливо критично для фахівців із меншим досвідом роботи. Доведено, що ефективність візуального пошуку стрімко знижується після кількох годин безперервного аналізу знімків [9]. Як показано на рисунку 1.2, показник загальної діагностичної точності (AUC) у молодих спеціалістів є найвищим на початку робочої зміни, проте він стрімко та статистично значущо

знижується наприкінці робочого дня. Натомість показники більш досвідчених фахівців залишаються стабільними завдяки виробленим патернам візуального пошуку.

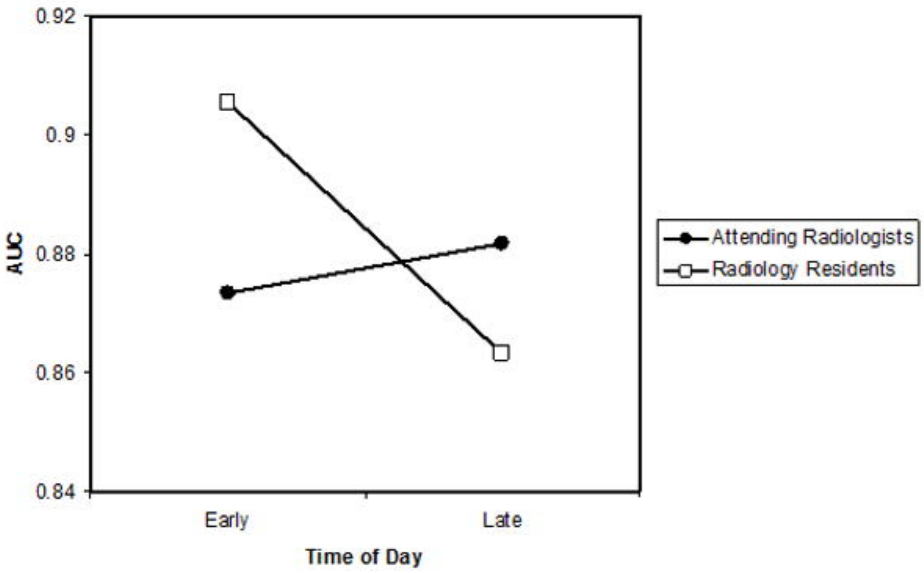

Рисунок 1.2 – Вплив фактора втоми на діагностичну точність (AUC) залежно від часу робочої зміни та кваліфікації лікаря [4]

Ціна лікарської помилки в контексті гострих респіраторних захворювань є критично високою. Хибнонегативний результат (пропуск патології) на ранній стадії призводить до затримки у призначенні терапії. Для таких захворювань, як пневмонія, запізніле введення антибактеріальної терапії експоненціально збільшує ризик розвитку дихальної недостатності, сепсису та летального результату. Крім того, недіагностовані інфекційні патології (наприклад, туберкульоз) становлять епідеміологічну загрозу для оточуючих.

З іншого боку, хибнопозитивний результат (гіпердіагностика) веде до призначення непотрібних додаткових обстежень (наприклад, дороговартісної комп'ютерної томографії), необґрунтованого використання антибіотиків та створення додаткового психологічного стресу для пацієнта. Це, у свою чергу, призводить до значних економічних втрат для медичного закладу. Додатковою проблемою є міжперсональна варіативність (inter-reader variability) – явище,

коли кілька фахівців роблять відмінні висновки на основі одного й того ж знімка, що підтверджує високий ступінь суб'єктивності ручного аналізу [7].

Таким чином, основними проблемами існуючого процесу діагностики захворювань легень є:

- висока залежність результатів від суб'єктивного сприйняття та кваліфікації лікаря;
- домінування перцептивних помилок через явище «зорової втоми»;
- пряма кореляція між обсягом робочого навантаження та зниженням якості діагностики;
- критично висока ціна помилки, що впливає на тривалість лікування, виживання пацієнтів та фінансові витрати лікарень.

Наявність цих невід'ємних недоліків ручного аналізу формує гостру об'єктивну необхідність у розробці автоматизованих систем, здатних мінімізувати вплив людського фактору та виступати в ролі «другої думки» (second opinion) для медичного фахівця.

1.2 Огляд еволюції та існуючих медичних систем на базі комп’ютерного зору

Історично процес діагностики респіраторних захворювань базувався виключно на візуальному аналізі плівкових рентгенограм та використанні паперових медичних карток. Для прийняття рішень у складних або атипових випадках лікарі-радіологи спиралися на друковані рентгенологічні атласи – спеціалізовані довідники, що містили еталонні знімки різних патологій. Такий підхід вимагав значних витрат часу на пошук аналогічних випадків у літературі, а фізичні медичні архіви ускладнювали відстеження динаміки захворювання протягом тривалого часу.

Першим кроком до автоматизації став перехід медичних закладів на цифрову рентгенографію та впровадження систем передачі й архівації зображень (Picture Archiving and Communication Systems, PACS). Це вирішило

проблему зберігання та доступу до знімків, проте сам процес діагностики залишався повністю ручним. Варто зазначити, що рентгенографія грудної клітки і сьогодні залишається найчастіше призначуваним видом медичної візуалізації: щороку у світі виконується понад 2 мільярди рентгенологічних досліджень [10].

З розвитком обчислювальної техніки наприкінці XX століття почали з'являтися перші системи комп'ютерної підтримки (Computer-Aided Diagnosis, CAD). Ранні CAD-системи базувалися на традиційних алгоритмах комп'ютерного зору: застосуванні математичних фільтрів, виділенні контурів (edge detection) та ручному конструюванні ознак (feature engineering). Вони дозволяли автоматично виявляти підозрілі затемнення на знімках, однак мали суттєвий недолік – низьку специфічність. Як зазначається у дослідженнях, високий рівень хибнопозитивних спрацьовувань призводив до виникнення ефекту «сигнальної втоми» (alert fatigue), унаслідок чого лікарі поступово починали ігнорувати підказки системи. Систематичний огляд показав, що класичні CAD-системи не мали статистично значущого впливу на загальний рівень виявлення патологій [11]. Узагальнену архітектуру роботи CAD-систем наведено на рисунку 1.4.

Як видно з наведеної схеми, типовий конвеєр обробки даних у сучасних системах комп'ютерної діагностики складається з кількох послідовних етапів. На початковому етапі (A) відбувається апаратне отримання цифрового медичного знімка. Наступним кроком є попередня обробка зображення (B: Preprocessing), яка включає усунення апаратних шумів, нормалізацію контрастності та приведення даних до єдиного формату. Центральною частиною класичних CAD-систем є етапи просторової сегментації патологічних ділянок (C) та вилучення ключових математичних ознак (D: Feature extraction). На основі отриманого вектора ознак модуль класифікації (E) приймає рішення щодо наявності або відсутності захворювання. Важливою тенденцією розвитку сучасних систем є інтеграція додаткових модулів просторової реконструкції (F) та засобів пояснюваного штучного інтелекту (G:

Decision making and Explainability), які візуалізують логіку прийняття рішень для лікаря.

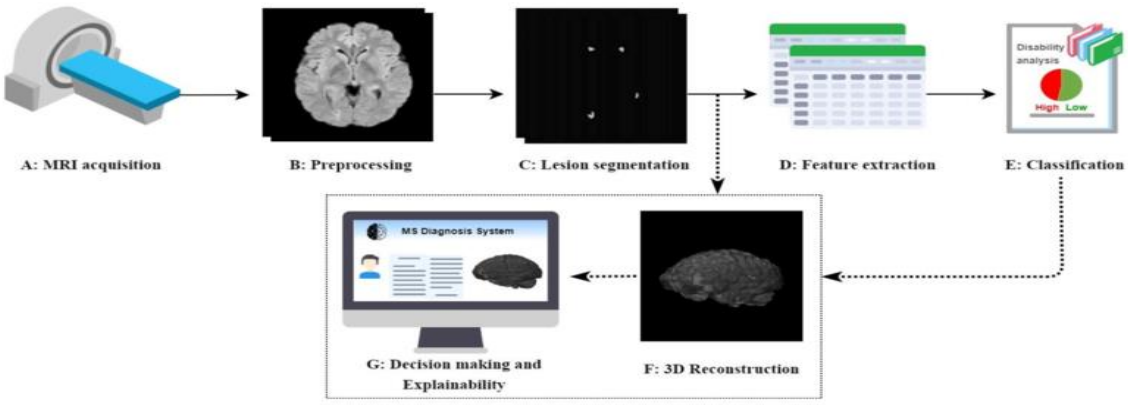

Рисунок 1.4 – Узагальнена архітектура та етапи роботи комплексної системи комп'ютерної діагностики на прикладі аналізу медичних зображень [12]

Справжня революція в розвитку CAD-систем відбулася завдяки масовому впровадженню методів глибокого навчання після 2012 року. Сучасні автоматизовані системи активно використовують згорткові нейронні мережі (CNN), які забезпечують безпрецедентну точність розпізнавання патернів. На відміну від класичних підходів, що потребували ручного конструювання ознак, такі моделі автоматично вилучають ієрархічні представлення безпосередньо з піксельних даних. Дослідження, що охопило 111 радіологів різного рівня кваліфікації, показало, що використання алгоритмів глибокого навчання як допоміжного інструменту статистично значущо підвищує точність та знижує час аналізу рентгенограм у всіх категоріях спеціалістів [13].

Серед найбільш відомих практичних та дослідницьких рішень варто відзначити системи від провідних університетів та технологічних компаній. Значним проривом стала розробка моделі CheXNet (Stanford University), навченої на масивному наборі даних NIH ChestX-ray14, що містить понад 112 000 знімків, для виявлення пневмонії з точністю, що перевищує показники

практикуючих радіологів [14]. Іншим вагомим прикладом є спеціалізована архітектура COVID-Net (University of Waterloo), оптимізована для диференціальної діагностики вірусних пневмоній, яка досягла точності 91.0% у трикласовій класифікації [15]. У галузі скринінгу туберкульозу активно використовуються комерційні CAD-системи – CAD4TB, Lunit INSIGHT та qXR. Міжнародне багатоцентрове дослідження показало, що система Lunit досягає AUC 0.85 при виявленні туберкульозу на рентгенограмах, що є порівнянним з точністю досвідчених радіологів [16].

Комерційні технологічні гіганти також мають власні розробки. Зокрема, підрозділ Google Health активно досліджує застосування штучного інтелекту для скринінгу захворювань легень, демонструючи високу точність моделей на базі архітектури EfficientNet. Важливим драйвером розвитку цього напрямку є відкриті набори даних: платформи на кшталт Kaggle надають доступ до великих анотованих медичних наборів даних, що дозволяє дослідникам застосовувати технологію Transfer Learning на базі перевірених архітектур без необхідності збору клінічних даних з нуля.

Попри значні досягнення, існуючі системи мають ряд критичних обмежень:

- **Закритість екосистем:** більшість комерційних продуктів мають закритий вихідний код та складні механізми ліцензування, що ускладнює їх адаптацію для невеликих клінік та дослідницьких центрів.
- **Проблема «чорної скрині»**: алгоритми надають кінцевий результат класифікації без пояснення логіки його отримання. Відсутність візуальної інтерпретації критично знижує рівень довіри медичних працівників до системи.
- **Відсутність комплексної інфраструктури:** багато академічних моделей існують лише у вигляді дослідницьких скриптів і не інтегровані у повноцінні клієнт-серверні застосунки з можливістю збереження історії в

реляційних базах даних та генерації структурованих медичних звітів у форматах PDF та JSON.

Отже, аналіз існуючих медичних систем на базі комп'ютерного зору показав, що, незважаючи на високий рівень розвитку алгоритмів, існує гостра потреба у створенні комплексного програмного рішення. Таке рішення повинно поєднувати високу точність класифікації, інтерпретованість результатів та зручну інтеграцію у вигляді повноцінного вебзастосунку.

1.3 Аналіз методів машинного навчання та інтерпретованості (XAI)

При вирішенні задачі автоматизованого пошуку патологій на рентгенологічних знімках концептуально існують два основні підходи: моделі семантичної сегментації та моделі класифікації. Моделі семантичної сегментації (наприклад, архітектури типу U-Net) здатні з високою точністю виділяти контури патології. Проте головною перешкодою для їх масового впровадження є критична залежність від якості навчальних даних – навчання таких моделей вимагає набору даних, розміченого на рівні окремих пікселів (pixel-level ground-truth masks), що передбачає прецизійне ручне обведення контурів патологій консиліумом лікарів-радіологів. Враховуючи високу вартість часу медичних спеціалістів, формування такого масиву для тисяч знімків є фінансово та часово нереалістичним завданням.

Зважаючи на ці обмеження, значно ефективнішим, масштабованим та економічно доцільним підходом є використання згорткових нейронних мереж (CNN) для задачі класифікації. Архітектури типу ResNet [3], VGG [4] та EfficientNet [5] здатні виявляти складні нелінійні взаємозв'язки у піксельних даних та досягають високої точності у задачах виявлення патологій на рентгенограмах. Проте зростання глибини та складності цих моделей призвело до виникнення фундаментальної проблеми, відомої як проблема «чорної скрині» (black box problem).

Проблема «чорної скрині» полягає в тому, що багатошарова структура нейронної мережі з мільйонами вагових коефіцієнтів робить внутрішню логіку

прийняття рішень абсолютно непрозорою для кінцевого користувача: система приймає зображення на вхід і видає діагноз на вихід, не надаючи жодних пояснень щодо того, які саме візуальні ознаки стали вирішальними. У контексті охорони здоров'я такий підхід є неприпустимим з двох причин. По-перше, лікар не має права приймати клінічне рішення, спираючись виключно на вердикт алгоритму, який він не може верифікувати. По-друге, відсутність прозорості суперечить принципам відповідальності штучного інтелекту (AI Accountability) та ускладнює сертифікацію медичного програмного забезпечення відповідно до стандартів IEC 62304 та FDA SaMD.

Для подолання цього бар'єру активно розвивається науковий напрям пояснюваного штучного інтелекту (Explainable AI, XAI), метою якого є створення методів, здатних перетворити непрозорі моделі на зрозумілі інструменти шляхом генерації теплових карт активацій (saliency maps). Серед існуючих підходів виділяють моделе-незалежні методи та методи, специфічні для CNN. Порівняльний аналіз основних методів за ключовими критеріями наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика методів XAI для задачі інтерпретації CNN

Метод	Тип	Зміна архі-тектури	Обчислювальна складність	Просторова роздільність	Придатність для RT*
LIME	Моделе-незалежний	Ні	Висока	Середня	Ні
SHAP	Моделе-незалежний	Ні	Висока	Середня	Ні
CAM	CNN-специфічний	Так (GAP-шар)	Низька	Середня	Так
Grad-CAM	CNN-специфічний	Ні	Низька	Середня	Так
Grad-CAM++	CNN-специфічний	Ні	Низька	Висока	Так
LayerCAM	CNN-специфічний	Ні	Середня	Висока	Частково

**Примітка: RT — придатність для використання у режимі реального часу (real-time).*

Методи LIME та SHAP є універсальними (моделе-незалежними), але принципово не придатні для систем із вимогами до часу відгуку, оскільки потребують від сотень до тисяч повторних прямих проходів через мережу для генерації одного пояснення.

Базовий метод CAM (Class Activation Mapping) позбавлений цього недоліку, однак вимагає жорсткої архітектурної обмеженості: наявності шару глобального усереднення (Global Average Pooling, GAP) безпосередньо перед єдиним повнозв’язним класифікаційним шаром. Це унеможливлює його застосування до сучасних архітектур типу EfficientNet без їх структурної модифікації та повного перенавчання.

Серед градієнтних CNN-специфічних методів найбільш практично значущими є три підходи: Grad-CAM [6], Grad-CAM++ [17] та LayerCAM [18]. Алгоритм Grad-CAM++ є розширенням базового методу і теоретично забезпечує кращу локалізацію у випадках, коли зображення містить множинні екземпляри одного класу об'єктів. Проте дослідження Lerma та Lucas [19] математично та емпірично доводять, що класичний алгоритм Grad-CAM (за умови використання лише позитивних градієнтів) генерує теплові карти, які є практично ідентичними до результатів Grad-CAM++. Візуальне підтвердження цієї еквівалентності наведено на рисунку 1.5.

Як видно з наведеного порівняння, центральна теплова карта (Grad-CAM із позитивними градієнтами) та права теплова карта (Grad-CAM++) не мають суттєвих відмінностей у точності локалізації цільового об'єкта. Оскільки для рентгенограм грудної клітки при пневмонії характерна наявність переважно одиночних або масивних однорідних вогнищ патології, використання математично складнішого алгоритму Grad-CAM++ не дає клінічної переваги. Своєю чергою, метод LayerCAM [18] забезпечує вищу просторову роздільність за рахунок зваженої агрегації карт активацій з різних шарів мережі, однак вимагає більших обчислювальних ресурсів і збільшує час генерації пояснення – критичний параметр для клієнт-серверного застосунку.

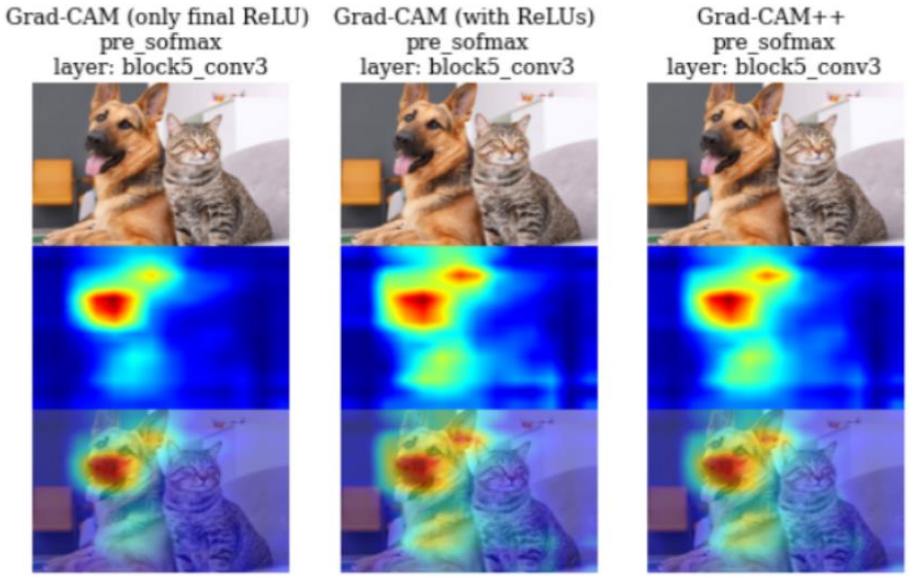

Рисунок 1.5 – Візуальне порівняння теплових карт Grad-CAM та Grad-CAM++, що демонструє їхню практичну еквівалентність [19]

Таким чином, класичний Grad-CAM є оптимальним вибором для задачі інтерпретації класифікації пневмонії за сукупністю критеріїв: відсутність вимог до модифікації архітектури, мінімальні обчислювальні вимоги, достатня просторова роздільність для локалізації одиночного підсвічення ураження та широка підтримка у виробничих бібліотеках.

Алгоритм Grad-CAM використовує градієнти цільового класу c (наприклад, логіт класу «Пневмонія»), які поширюються у зворотному напрямку до останнього згорткового шару мережі. Саме цей шар містить найбагатшу комбінацію просторової та семантичної інформації. Вага важливості k-ї карти ознак для класу c визначається операцією глобального усереднення градієнтів за просторовими розмірами (формула 1.1):

$$\alpha_k^c = \frac{1}{Z}\sum_i\sum_j \frac{\partial Y^c}{\partial A_{i,j}^k} \quad , \tag{1.1}$$

де Y^c – вихідна оцінка (логіт) класу c до застосування функції активації;

$A^k_{i,j}$– значення активації k-ї карти ознак у просторових координатах (i, j);

Z – загальна кількість пікселів у карті ознак.

Отримана вагова константа α^c_k відображає часткову лінійну залежність активацій від цільового класу. Наступним кроком є обчислення самої теплової карти $L^c_{Grad-CAM}$ як зваженої суми карт ознак останнього згорткового шару з подальшим застосуванням функції активації ReLU, що описується формулою (1.2):

$$L^c_{Grad-CAM} = ReLU\left(\sum_k \alpha^c_k A^k\right) \quad (1.2)$$

Застосування ReLU відіграє принципову роль: функція відсікає від'ємні значення, що відповідають ознакам, які знижують ймовірність цільового класу, залишаючи лише ті просторові ділянки, що мають позитивний вплив на прийняте рішення. Загальна архітектура алгоритму та математичний принцип формування теплової карти від вхідного зображення до фінальної візуалізації проілюстровано на рисунку 1.6.

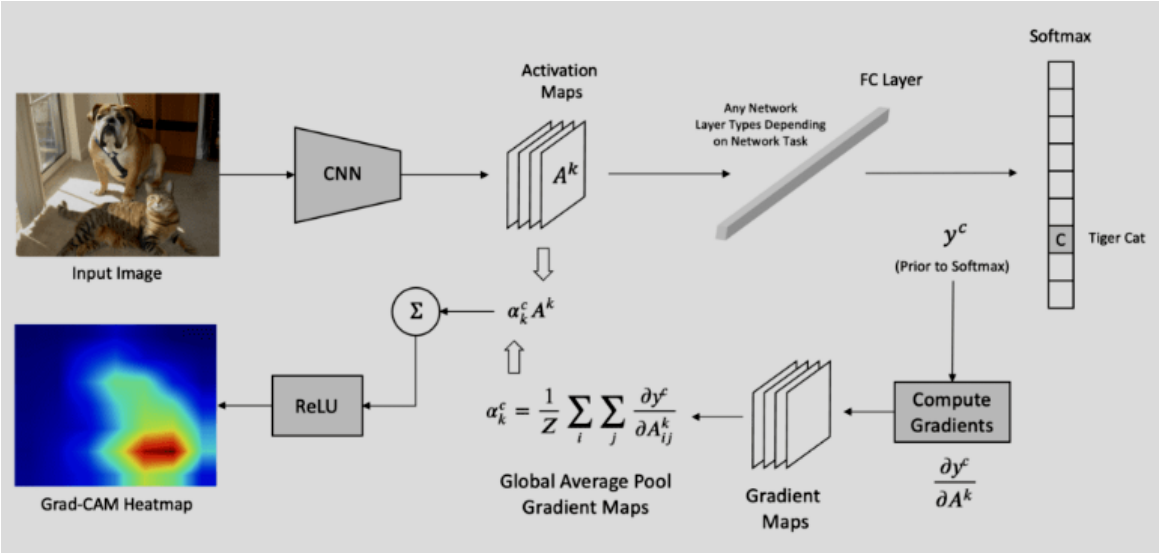

Рисунок 1.6 – Архітектура та математичний принцип роботи конвеєра Grad-CAM [20]

Отримана теплова карта після білінійної інтерполяції до розміру вхідного зображення накладається на оригінальну рентгенограму, підсвічуючи саме ті анатомічні ділянки (наприклад, вогнища інфільтрації при пневмонії), які стали визначальними для класифікаційного рішення моделі.

Інтеграція алгоритму Grad-CAM у проєктовану систему діагностики є необхідною умовою для створення прозорого, клінічно верифікованого та інтерпретованого інструменту комп'ютерної підтримки клінічних рішень, що відповідає сучасним вимогам до відповідальних систем штучного інтелекту у медичній галузі.

1.4 Формування вимог та постановка задачі на розробку програмного комплексу

На основі проведеного аналізу предметної області, існуючих програмних рішень та методів машинного навчання детерміновано необхідність проєктування спеціалізованого програмного комплексу. Для забезпечення коректної інтеграції розроблюваного програмного комплексу в клінічне середовище та регламентації процесу розробки сформовано специфікацію функціональних і нефункціональних вимог.

Функціональні вимоги визначають перелік операцій, які комплекс повинен виконувати для реалізації заявленого функціоналу. Їх склад безпосередньо випливає з виявлених у підрозділах 1.1-1.3 потреб: необхідності автоматизованої обробки медичних зображень, забезпечення інтерпретованості результатів та підтримки документообігу в клінічному середовищі. До функціональних вимог розроблюваного комплексу віднесено:

- забезпечення імпорту та первинної валідації вхідних медичних зображень у форматах цифрової графіки;
- виконання автоматизованої препроцесингової обробки та тензорної трансформації даних для подальшого аналізу нейромережевою моделлю;

- реалізація модуля класифікації патологічних змін (інференс моделі) за допомогою донавченої згорткової нейронної мережі на базі архітектури EfficientNet;
- генерування карт просторової активації на основі алгоритму Grad-CAM для забезпечення візуальної інтерпретованості результатів класифікації;
- забезпечення персистентного зберігання результатів діагностики, метаданих знімків та облікових даних користувачів у реляційній базі даних;
- автоматизоване формування медичних висновків у форматі PDF для підтримки документообігу;
- експорт структурованих результатів аналізу у форматі JSON для потенційної інтеграції з існуючими медичними інформаційними системами (MIC).

Нефункціональні вимоги регламентують архітектурні, технологічні та експлуатаційні характеристики програмного продукту:

- *Архітектура:* система проєктується на базі клієнт-серверної архітектури з ізоляцією бізнес-логіки та ML-компонентів на боці сервера і реалізацією взаємодії через REST API. Даний підхід усуває необхідність встановлення ресурсоємного програмного забезпечення на робочих станціях медичного персоналу.
- *Технологічний стек:* серверна маршрутизація та обробка запитів реалізуються засобами бібліотеки FastAPI (Python), клієнтський інтерфейс на базі бібліотеки React, управління даними – засобами СКБД PostgreSQL.
- *Продуктивність:* часова затримка (latency) виведення результату та генерації теплової карти Grad-CAM не повинна перевищувати 3–5 секунд на одне зображення. Дана вимога обґрунтована ергономічними нормами взаємодії людини з інформаційними системами, згідно з якими затримка відгуку понад 10 секунд призводить до втрати уваги оператора та переривання робочого процесу [21]. Вибір алгоритму Grad-CAM

замість обчислювально затратних методів LIME або SHAP безпосередньо забезпечує виконання цієї вимоги.

- *Кросплатформність:* взаємодія з системою здійснюється через інтерфейс веб-браузера за принципом «тонкого клієнта» без необхідності розгортання додаткового ПЗ на робочих станціях.

Постановка задачі. У рамках даної кваліфікаційної роботи ставиться задача проєктування та програмної реалізації веб застосунку автоматизованої діагностики захворювань органів дихання за рентгенологічними знімками. Відповідно до сформованих вимог необхідно вирішити такі підзадачі:

1. Провести експериментальне порівняльне дослідження сучасних згорткових архітектур (зокрема ResNet, VGG та EfficientNet) для задачі аналізу медичних зображень.
2. Розробити фінальний модуль класифікації на базі визначеної оптимальної моделі із застосуванням технології Transfer Learning.
3. Імплементувати модуль пояснюваного штучного інтелекту на основі алгоритму Grad-CAM для візуальної верифікації результатів.
4. Спроєктувати логічну та фізичну моделі реляційної бази даних.
5. Розробити серверну частину програмного комплексу з наданням REST API на базі фреймворку FastAPI.
6. Розробити клієнтський веб-інтерфейс із підтримкою візуалізації результатів діагностики та генерації звітних документів у форматі PDF.

Висновки до розділу 1

У першому розділі виконано комплексний аналіз проблематики автоматизованої діагностики захворювань органів дихання та обґрунтовано концепцію розробки спеціалізованого медичного програмного забезпечення. За результатами дослідження зроблено такі висновки:

1. Встановлено, що рентгенографія є базовим методом первинного скринінгу патологій легень, проте високе когнітивне навантаження на лікарів-радіологів зумовлює необхідність впровадження систем

комп'ютерної підтримки прийняття рішень (CAD) з метою мінімізації діагностичних помилок.

2. Доведено ефективність застосування згорткових нейронних мереж (CNN) та технології перенесення навчання (Transfer Learning) для екстракції високорівневих ознак із медичних зображень. Даний підхід дозволяє досягти високої точності класифікації за умов використання обмежених наборів клінічних даних.
3. Виявлено критичний недолік сучасних моделей глибокого навчання, що полягає у відсутності інтерпретованості результатів («чорна скриня»). Обґрунтовано доцільність імплементації градієнтного алгоритму Grad-CAM як інструменту пояснюваного штучного інтелекту (XAI) для візуальної локалізації патологічних патернів.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Проєктування загальної клієнт-серверної архітектури

Архітектурне рішення є фундаментальним проєктним рішенням, що визначає структуру системи, характер взаємодії між її компонентами та закладає основу для подальшої розробки і масштабування. Вибір архітектурного підходу безпосередньо впливає на такі якісні характеристики програмного забезпечення, як надійність, продуктивність, безпека та зручність супроводу [22].

Для реалізації розроблюваного програмного комплексу обрано трирівневу клієнт-серверну архітектуру, що є загальновизнаним стандартом для вебзастосунків корпоративного класу [23]. Ця архітектура чітко розмежовує відповідальність між трьома незалежними рівнями: рівнем представлення (Presentation Tier), рівнем бізнес-логіки (Logic Tier) та рівнем даних (Data Tier). Така організація забезпечує слабку зв'язаність компонентів – зміни в одному рівні не потребують модифікації інших, що суттєво спрощує тестування та розширення системи.

Загальну архітектуру програмного комплексу наведено на рисунку 2.1.

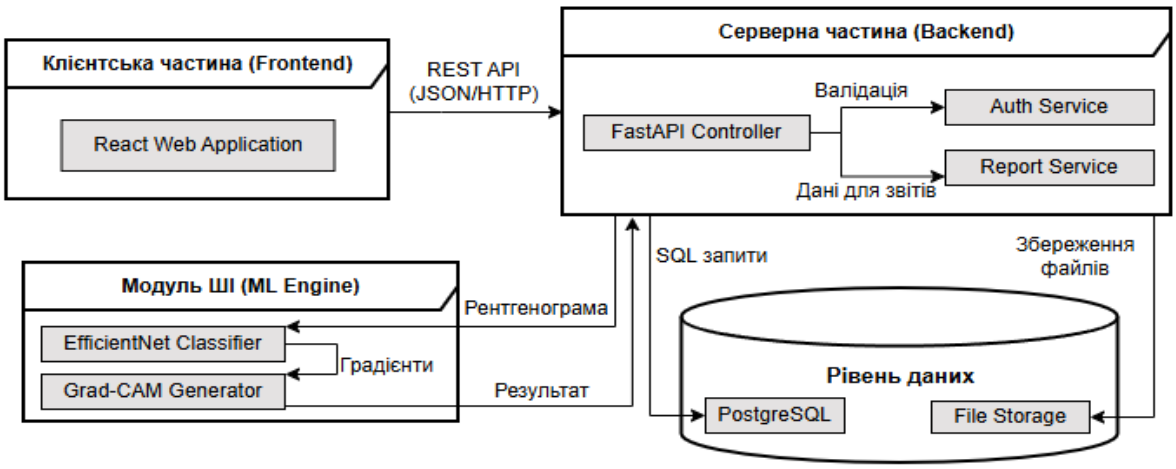

Рисунок 2.1 – UML-діаграма компонентів розроблюваної медичної системи

Як видно з рисунку 2.1, система складається з чотирьох основних підсистем, що взаємодіють між собою через чітко визначені інтерфейси.

Клієнтська частина (Frontend) реалізована у вигляді односторінкового вебзастосунку (Single Page Application, SPA) на базі бібліотеки React. Клієнт відповідає виключно за відображення даних та взаємодію з користувачем – лікарем-радіологом. Вся комунікація із серверною частиною здійснюється через стандартизований REST API з використанням протоколу HTTP та формату обміну даними JSON. Така архітектура забезпечує повну незалежність клієнта від серверної реалізації – у разі потреби React-клієнт може бути замінений мобільним застосунком або будь-яким іншим клієнтом, що підтримує HTTP, без жодних змін на боці сервера.

Серверна частина (Backend) побудована на базі фреймворку FastAPI (Python) та виконує роль центрального координатора усіх бізнес-процесів системи. У її складі виділено два спеціалізовані сервіси. Auth Service відповідає за автентифікацію та авторизацію користувачів з використанням механізму JWT-токенів (JSON Web Token) – stateless підходу, що не потребує збереження сесій на сервері та легко масштабується. Report Service реалізує генерацію структурованих медичних звітів у форматах PDF та JSON з повними метаданими ML-моделі відповідно до принципу AI Accountability.

Модуль штучного інтелекту (ML Engine) функціонує як внутрішній сервіс серверної частини та інкапсулює всю логіку машинного навчання. Він складається з двох взаємопов'язаних компонентів: класифікатора на базі архітектури EfficientNetB3, що виконує інференс над вхідним зображенням, та генератора теплових карт Grad-CAM, що забезпечує інтерпретованість прийнятих рішень. Передача зображення від Backend до ML Engine здійснюється через шлях до файлу або байтовий потік – тензорне перетворення виконується безпосередньо всередині ML Engine у процесі препроцесингу.

Рівень даних складається з двох компонентів, що доповнюють один одного. Реляційна база даних PostgreSQL зберігає структуровані дані: облікові

записи лікарів, інформацію про пацієнтів та метадані результатів аналізів. Файлове сховище (File Storage) використовується для зберігання бінарних об'єктів: оригінальних рентгенограм, згенерованих Grad-CAM теплових карт та готових PDF і JSON звітів. Розмежування структурованих та неструктурованих даних є поширеною практикою, що дозволяє оптимізувати продуктивність кожного типу сховища окремо [24].

Детальну послідовність взаємодії компонентів системи при виконанні основного сценарію використання – аналізу рентгенограми – наведено на рисунку 2.2.

Як показано на рисунку 2.2, повний цикл обробки запиту включає 14 послідовних кроків. Лікар завантажує рентгенограму через вебінтерфейс (крок 1), після чого React-клієнт формує HTTP POST запит до ендпоінту /api/analyze із вкладеним зображенням (крок 2). FastAPI сервер зберігає оригінал у File Storage та отримує його URL (крок 3), після чого передає шлях до зображення у ML Engine (крок 4).

ML Engine виконує інференс моделі EfficientNetB3 (крок 5) та генерує Grad-CAM теплову карту (крок 6). Результат – клас захворювання, вектор імовірностей та шлях до теплової карти – повертається до Backend (крок 7). Сервер зберігає теплову карту у File Storage (крок 8) та записує всі результати аналізу до PostgreSQL через CRUD-операції (крок 9).

Після успішного збереження Backend звертається до Report Service із запитом на генерацію звіту (крок 10). Report Service компілює PDF та JSON документи (крок 11), зберігає їх у File Storage (крок 12) та повертає URL готових документів. Завершальним кроком (13) є формування відповіді 200 OK з повним JSON-об'єктом, що містить результати діагностики та URL усіх згенерованих ресурсів. Клієнт відображає діагноз із тепловою картою та надає лікарю можливість завантажити PDF-звіт (крок 14).

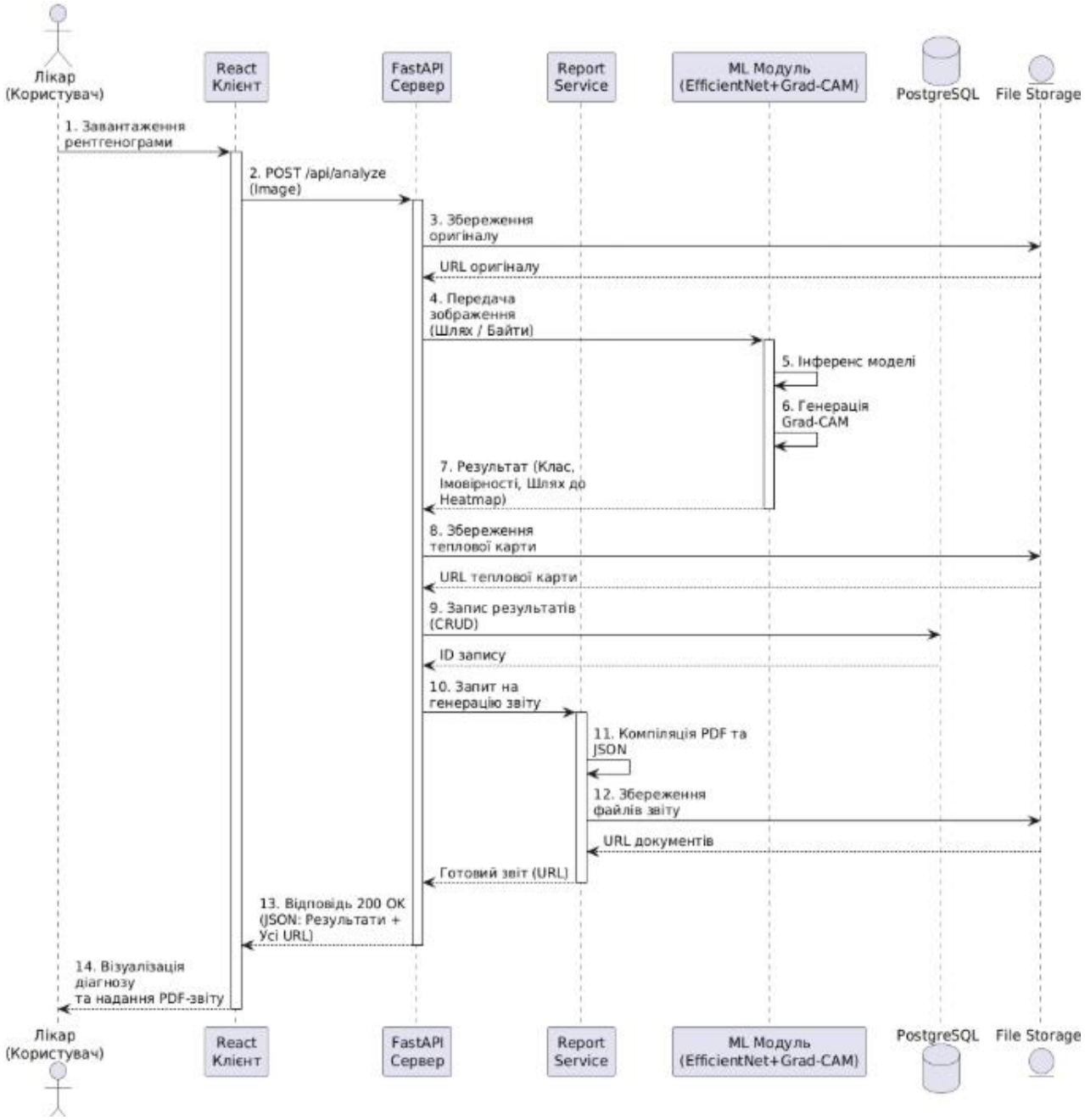

Рисунок 2.2 – UML-діаграма послідовностей процесу обробки рентгенограм

Описана архітектура забезпечує чітке розмежування відповідальності між компонентами, можливість незалежного масштабування кожного рівня та відповідність сучасним стандартам розробки медичного програмного забезпечення.

2.2 Проєктування логічної та фізичної моделі бази даних (PostgreSQL)

Проєктування бази даних є критично важливим етапом розробки будь-якої інформаційної системи, що працює з персональними медичними даними. Коректно спроєктована схема бази даних забезпечує цілісність даних, ефективність запитів та відповідність вимогам нормалізації – фундаментальному принципу реляційних баз даних, що мінімізує надмірність та аномалії при операціях вставки, оновлення та видалення [25].

Для зберігання структурованих даних розроблюваного програмного комплексу обрано реляційну систему керування базами даних (СКБД) PostgreSQL. Цей вибір обумовлений рядом технічних переваг: підтримкою складних типів даних та JSON-полів, розвиненою системою обмежень цілісності, транзакційністю відповідно до стандарту ACID, а також широкою підтримкою з боку Python-екосистеми через бібліотеку SQLAlchemy [26].

На етапі концептуального проєктування визначено три основні сутності, що відображають предметну область системи: лікар (users), пацієнт (patients) та результат аналізу (analyses). Між сутностями встановлено такі семантичні зв'язки:

- лікар лікує пацієнтів (один лікар може вести картки багатьох пацієнтів – зв'язок 1:M);
- лікар проводить аналізи (один лікар може виконати багато аналізів – зв'язок 1:M);
- пацієнт має історію аналізів (один пацієнт може мати історію з багатьох аналізів – зв'язок 1:M).

Логічну модель бази даних наведено на рисунку 2.3.

На етапі фізичного проєктування концептуальна модель трансформована у конкретну схему таблиць PostgreSQL. Схема пройшла нормалізацію до третьої нормальної форми (3NF) – кожен неключовий атрибут залежить виключно від первинного ключа і не має транзитивних залежностей [27].

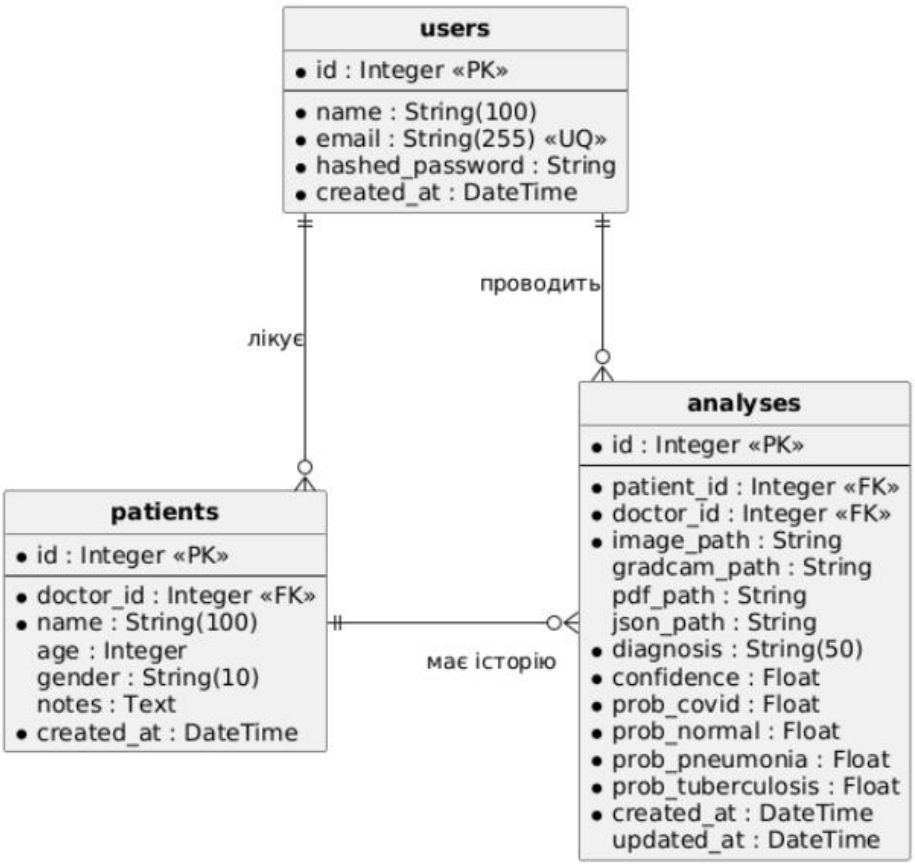

Рисунок 2.3 – ER-діаграма логічної моделі бази даних розроблюваної системи

- **Таблиця users (облікові записи лікарів).** Центральна сутність системи, що зберігає дані авторизації медичного персоналу. Поле email оголошено як UNIQUE – система не допускає реєстрації двох облікових записів з однаковою електронною адресою. Пароль зберігається виключно у вигляді хешу (алгоритм bcrypt) – оригінальний пароль ніколи не зберігається у базі даних, що відповідає стандартам інформаційної безпеки у медичних системах [28]. Поле created_at із часовою зоною (DateTime with timezone) забезпечує коректне відстеження часу реєстрації незалежно від географічного розташування сервера.
- **Таблиця patients (дані пацієнтів).** Зберігає демографічну інформацію про пацієнтів, прив'язану до конкретного лікаря через зовнішній ключ doctor_id. Поля age та gender є необов'язковими (nullable) – система не вимагає повних даних пацієнта для виконання діагностики, що підвищує

гнучкість використання в ургентних ситуаціях. Поле notes типу Text надає лікарю можливість додавати довільні клінічні примітки до картки пацієнта. Каскадне видалення (ON DELETE CASCADE) на зовнішньому ключі забезпечує автоматичне видалення всіх пов'язаних записів при видаленні облікового запису лікаря – це гарантує відсутність «осиротілих» записів у базі даних.

- **Таблиця analyses (результати діагностичних аналізів).** Є центральною таблицею системи, що акумулює всі результати ML-діагностики. Містить два зовнішніх ключі (patient_id та doctor_id), що дозволяє будувати запити як по пацієнту, так і по лікарю незалежно. Шляхи до файлів (image_path, gradcam_path, pdf_path, json_path) зберігаються як текстові рядки – самі бінарні файли зберігаються в об'єктному файловому сховищі (File Storage), а БД містить лише посилання на них. Такий підхід є загальновизнаною практикою, що дозволяє уникнути надмірного навантаження на СКБД під час роботи з великими бінарними об'єктами [29].

Результати ML-класифікації представлені двома групами полів. Поле diagnosis (String(50)) зберігає назву класу з найвищою імовірністю. Поля prob_covid, prob_normal, prob_pneumonia та prob_tuberculosis типу Float зберігають повний вектор імовірностей по всіх чотирьох класах – це дозволяє у майбутньому будувати аналітичні звіти та відстежувати розподіл діагнозів без повторного звернення до ML-моделі. Поле confidence містить максимальне значення з вектора імовірностей і слугує швидким індикатором впевненості моделі без необхідності додаткових обчислень на боці клієнта. Поля created_at та updated_at забезпечують повний аудиторський слід, що є обов'язковою вимогою для медичних інформаційних систем відповідно до принципу AI Accountability.

Цілісність даних забезпечується на кількох архітектурних рівнях. На рівні СКБД використовуються обмеження NOT NULL для критичних полів, UNIQUE для ідентифікаторів користувачів, та каскадне видалення для

підтримки референційної цілісності. На рівні ORM (SQLAlchemy) додатково визначені програмні валідатори, що перевіряють коректність даних до їх передачі до СКБД. На рівні REST API (схеми Pydantic) вхідні дані проходять сувору типізацію та валідацію перед виконанням будь-якої операції з базою даних.

Доступ до даних реалізовано через структурний патерн Repository, що інкапсулює всі SQL-запити та надає бізнес-логіці уніфікований інтерфейс, незалежний від конкретної СКБД. Це забезпечує гнучкість системи та можливість потенційної міграції на іншу СКБД без зміни коду бізнес-логіки [25].

2.3 Обґрунтування вибору технологічного стека розробки

Вибір технологічного стека є одним з найбільш відповідальних архітектурних рішень, що визначає продуктивність розробки, якість кінцевого продукту та довгостроковий потенціал супроводу системи. Під час формування стека для розроблюваного програмного комплексу враховувались такі критерії: зрілість технології та розмір спільноти, продуктивність у задачах обробки зображень та машинного навчання, швидкість розробки, наявність відкритого вихідного коду та безкоштовних ліцензій.

Узагальнений технологічний стек системи наведено у таблиці 2.1.

Таблиця 2.1 – Технологічний стек розроблюваної системи

Рівень	Технологія	Версія	Призначення
Frontend	React	18.x	SPA вебінтерфейс
	React Router	6.x	Клієнтська маршрутизація
	Recharts	2.x	Візуалізація даних
Backend	Python	3.10	Мова програмування
	FastAPI	0.115	REST API фреймворк
	SQLAlchemy	2.0	ORM для роботи з БД

Продовження таблиці 2.1

	Pydantic	2.x	Валідація даних
	python-jose	3.x	JWT автентифікація
ML	PyTorch	2.5.1	Фреймворк глибокого навчання
	TorchVision	0.20	Комп'ютерний зір
	OpenCV	4.x	Обробка зображень
	ReportLab	4.x	Генерація PDF
БД	PostgreSQL	18	Реляційна СКБД
	File Storage	–	Зберігання файлів

2.3.1 React як технологія клієнтської частини

Для реалізації клієнтської частини обрано бібліотеку React (Meta, 2013), яка є найпопулярнішим інструментом для розробки SPA-застосунків – станом на 2024 рік її використовують понад 40% веб-розробників у світі згідно зі щорічним опитуванням Stack Overflow Developer Survey [29]. Ключовою концепцією React є компонентна модель розробки – інтерфейс будується з незалежних перевикористовуваних компонентів, кожен з яких керує власним станом. Це забезпечує високу модульність коду та спрощує тестування окремих частин інтерфейсу.

Важливою перевагою React є Virtual DOM – механізм, що мінімізує кількість операцій з реальним DOM браузера шляхом порівняння поточного та попереднього стану інтерфейсу й оновлення лише тих вузлів, що фактично змінились. Це забезпечує високу продуктивність інтерфейсу навіть при частих оновленнях даних, що є критичним для сторінки результатів діагностики, де одночасно відображаються рентгенограма, теплова карта Grad-CAM та вектор імовірностей [30].

Для реалізації навігації між розділами застосунку використовується React Router – стандартна бібліотека маршрутизації для React, що реалізує

концепцію клієнтської маршрутизації. Для побудови графіків та візуалізації даних застосовується бібліотека Recharts – декларативна бібліотека, побудована поверх SVG, що забезпечує адаптивне відображення діаграм без залежностей від зовнішніх сервісів.

Альтернативами React під час вибору були Vue.js та Angular. Vue.js має нижчий поріг входження, однак меншу екосистему та менш розвинений ринок праці. Angular є потужним enterprise-фреймворком, однак його надмірна складність та жорстка структура є надлишковими для застосунку даного масштабу. React забезпечує оптимальний баланс між гнучкістю, продуктивністю та доступністю кваліфікованих розробників [31].

2.3.2 FastAPI як технологія серверної частини

Серверна частина системи реалізована на базі фреймворку FastAPI (Sebastián Ramírez, 2018) – сучасного асинхронного Python-фреймворку для побудови REST API. За даними офіційної документації, FastAPI є одним з найшвидших Python-фреймворків – його продуктивність порівнянна з NodeJS та Go завдяки використанню асинхронної моделі виконання на базі ASGI (Asynchronous Server Gateway Interface) [32].

Ключовою перевагою FastAPI у контексті даної роботи є нативна інтеграція з Pydantic – бібліотекою валідації даних, що використовує анотації типів Python для автоматичної перевірки вхідних даних. Це дозволяє декларативно описати схему кожного API-ендпоінту і отримати автоматичну валідацію, серіалізацію та десеріалізацію даних без написання додаткового коду. Критично важливою особливістю є автоматична генерація інтерактивної документації API у форматі OpenAPI (Swagger UI) – це суттєво спрощує тестування та інтеграцію з зовнішніми системами [32].

Для роботи з базою даних використовується SQLAlchemy 2.0 – найпоширеніший ORM для Python, що надає декларативний інтерфейс для визначення схеми БД та виконання запитів без написання сирого SQL. Автентифікація реалізована через JWT-токени з використанням бібліотеки

python-jose – stateless підхід, що не потребує збереження сесій на сервері та природно масштабується при горизонтальному розширенні.

Альтернативами FastAPI розглядались Django REST Framework та Flask. Django REST Framework є зрілим та функціональним рішенням, однак його монолітна архітектура та значна кількість абстракцій є надлишковими для API-орієнтованого мікросервісного застосунку. Flask є мінімалістичним фреймворком, що потребує значного обсягу ручної конфігурації для реалізації валідації, документації та асинхронності – функціональності, що у FastAPI доступна «з коробки» [33].

2.3.3 PyTorch як фреймворк машинного навчання

Для реалізації ML-компонента системи обрано фреймворк PyTorch (Meta AI, 2016) – один з двох домінуючих фреймворків глибокого навчання поряд з TensorFlow. Ключовою перевагою PyTorch є динамічний обчислювальний граф (define-by-run), що дозволяє будувати та модифікувати архітектуру нейронної мережі під час виконання – на відміну від статичного графа TensorFlow 1.x. Це суттєво спрощує налагодження моделей та реалізацію нестандартних архітектурних рішень [34].

Критично важливою для даної роботи є підтримка CUDA – технології паралельних обчислень на GPU від NVIDIA. Використання відеокарти RTX 4060 Ti дозволило скоротити час навчання моделі у 15–20 разів порівняно з CPU, що є принциповим при проведенні багаторазових експериментів з різними архітектурами та гіперпараметрами. Бібліотека TorchVision надає готові реалізації архітектур VGG16, ResNet50 та EfficientNetB3 з попередньо навченими вагами ImageNet, що є основою для підходу Transfer Learning, застосованого у даній роботі.

Для генерації медичних PDF-звітів використовується бібліотека ReportLab – зрілий та функціональний інструмент, що надає низькорівневий контроль над розташуванням елементів документа та підтримує вставку

зображень, що є необхідним для включення рентгенограм та Grad-CAM карт у звіт.

2.3.4 PostgreSQL як СКБД

Серед реляційних СКБД з відкритим вихідним кодом PostgreSQL є найбільш функціонально повним рішенням, що підтримує складні типи даних, повнотекстовий пошук, JSON-поля та розширений набір індексів [35]. На відміну від MySQL, PostgreSQL суворо дотримується стандарту SQL та має більш розвинену систему транзакцій та блокувань. Легковагова альтернатива SQLite була відхилена через відсутність підтримки конкурентного запису – критичної вимоги для медичної системи, де кілька лікарів можуть одночасно завантажувати рентгенограми.

2.4 Проєктування модуля генерації та експорту медичних звітів

Модуль генерації медичних звітів є одним з ключових компонентів, що принципово відрізняє розроблювану програмний комплекс від існуючих дослідницьких рішень. Якщо більшість академічних моделей обмежуються поверненням числового вектору імовірностей, розроблений інструментарій формує повноцінну медичну документацію у двох форматах – структурованому PDF-документі для клінічного використання та машинозчитуваному JSON-пакеті для інтеграції із зовнішніми медичними інформаційними системами (MIC). Така архітектура відповідає принципу AI Accountability – кожне рішення алгоритму супроводжується повними метаданими моделі, що забезпечує аудитованість та юридичну простежуваність діагностичного висновку [36].

2.4.1 Архітектура модуля

Модуль реалізований як окремий Report Service у складі серверної частини, що отримує запити від FastAPI Controller після успішного завершення ML-аналізу. Така архітектурна ізоляція забезпечує незалежність логіки

генерації звітів від решти бізнес-логіки та спрощує подальше розширення – наприклад, додавання нових форматів експорту (DICOM SR, HL7 FHIR) без модифікації основного контролера.

Повний алгоритм роботи модуля у вигляді UML-діаграми діяльностей (Activity Diagram) наведено на рисунку 2.4.

Як видно з рисунка 2.4, процес генерації звіту включає три послідовні фази та одну паралельну.

- **Фаза підготовки даних** охоплює три послідовні кроки. Спочатку Report Service отримує запит з ідентифікатором обстеження (ID аналізу), після чого виконує запит до PostgreSQL для вилучення клінічних метаданих – інформації про пацієнта, лікаря, діагноз, вектор імовірностей та метадані ML-моделі. Завершальним кроком підготовки є завантаження бінарних файлів зображень зі сховища – оригінальної рентгенограми та згенерованої Grad-CAM теплової карти. Ці зображення є обов'язковими компонентами PDF-документа, що унаочнює для лікаря, як саме модель інтерпретувала знімок.
- **Фаза паралельної генерації** є ключовою з архітектурної точки зору. Після завершення підготовки потік виконання розгалужується на дві паралельні гілки (fork), що виконуються незалежно одна від одної. Перша гілка реалізує компіляцію PDF-документа засобами бібліотеки ReportLab. Процес включає послідовне формування розділів документа: титульний блок з даними пацієнта та лікаря, блок діагностичного висновку з відсотком впевненості моделі, розворот з оригінальним рентгеном та Grad-CAM тепловою картою поруч та блок метаданих моделі. Критично важливим кроком є додавання юридичного застереження про допоміжну роль ШІ – цей елемент є обов'язковим відповідно до вимог регуляторів медичних AI-систем [37]. Друга гілка формує JSON-пакет засобами стандартної бібліотеки Python. JSON-документ містить структуровані дані у машинозчитуваному форматі,

придатному для програмної обробки зовнішніми МІС без необхідності парсингу PDF.

Рисунок 2.4 – UML-діаграма діяльностей підсистеми генерації медичних звітів

- **Фаза фіналізації** виконується після злиття паралельних гілок (join). Обидва згенеровані файли зберігаються у файловому сховищі (File Storage) з унікальними іменами, що включають ідентифікатор аналізу. Після успішного збереження виконується оновлення запису в PostgreSQL – поля pdf_path та json_path таблиці analyses заповнюються актуальними шляхами до файлів. Завершальним кроком є повернення

URL-посилань на обидва документи до FastAPI Controller, який включає їх у фінальну відповідь клієнту.

2.4.2 Обґрунтування вибору форматів звітності

Вибір PDF як формату клінічного звіту обумовлений його універсальністю – документ коректно відображається на будь-якому пристрої без залежності від встановленого програмного забезпечення, зберігає фіксоване форматування та може бути роздрукований або підписаний електронним підписом. PDF є де-факто стандартом медичної документації у більшості країн [38].

Вибір JSON як формату для інтеграції з МІС обумовлений його статусом як домінуючого формату обміну даних у сучасних REST API. JSON є основою стандарту HL7 FHIR (Fast Healthcare Interoperability Resources) – міжнародного стандарту обміну медичними даними, що активно впроваджується у системах охорони здоров'я [39]. Це означає, що JSON-звіти розроблюваного програмного комплексу можуть бути інтегровані з FHIR-сумісними МІС без додаткових конверторів.

Висновки до розділу 2

У другому розділі виконано повний цикл проєктування розроблюваного програмного комплексу – від визначення архітектурної концепції до детального опрацювання окремих функціональних модулів.

За результатами проєктування прийнято та обґрунтовано такі ключові рішення:

- **Архітектурне рішення.** Для веб застосунку обрано трирівневу клієнт-серверну архітектуру з чітким розмежуванням рівня представлення (React SPA), рівня бізнес-логіки (FastAPI) та рівня даних (PostgreSQL і File Storage). Розроблено компонентну діаграму програмного комплексу та детальну UML-діаграму послідовностей (Sequence Diagram), що

охоплює повний цикл обробки запиту із 14 послідовними кроками — від завантаження рентгенограми лікарем до отримання готового PDF-звіту.

- **Рішення щодо бази даних.** Спроєктовано реляційну схему бази даних, що складається з трьох взаємопов'язаних таблиць: users, patients та analyses. Схема нормалізована до третьої нормальної форми (3NF) та реалізує каскадне видалення для забезпечення референційної цілісності. Як СКБД обрано PostgreSQL завдяки повній підтримці ACID-транзакцій, розвиненій системі обмежень цілісності та безшовній інтеграції з Python-екосистемою через ORM SQLAlchemy.
- **Рішення щодо технологічного стека.** Обґрунтовано вибір кожної технології на основі порівняльного аналізу альтернатив. Бібліотека React забезпечує компонентну модель розробки та Virtual DOM для високої продуктивності клієнтського інтерфейсу. Фреймворк FastAPI надає асинхронну обробку запитів та автоматичну валідацію даних через Pydantic. PyTorch обрано як базовий ML-фреймворк завдяки підтримці паралельних обчислень CUDA та динамічному обчислювальному графу.
- **Рішення щодо модуля звітності.** Спроєктовано сервіс генерації медичних звітів із паралельним формуванням документів у форматах PDF (засобами ReportLab) та JSON. Забезпечено відповідність процесу експорту даних принципу AI Accountability та закладено архітектурне підґрунтя для сумісності JSON-формату з міжнародним стандартом HL7 FHIR, що дозволяє інтегрувати розроблений комплекс із зовнішніми медичними інформаційними системами (MIC).

Сукупність прийнятих проєктних рішень формує цілісну архітектуру, що забезпечує виконання всіх функціональних та нефункціональних вимог, визначених у першому розділі, і створює надійне підґрунтя для розробки модуля машинного навчання та безпосередньої програмної реалізації комплексу, що будуть описані у подальших розділах.

РОЗДІЛ 3. РОЗРОБКА МОДУЛЯ МАШИННОГО НАВЧАННЯ ТА ІНТЕРПРЕТАЦІЇ

3.1 Аналіз та попередня обробка набору медичних знімків

3.1.1 Опис та характеристика набору даних

Якість та репрезентативність навчальних даних є визначальним фактором, що безпосередньо впливає на точність та узагальнюючу здатність моделі машинного навчання. У медичних застосуваннях це набуває особливої критичності – зміщення у навчальній вибірці (dataset bias) може призвести до систематичних діагностичних помилок на реальних клінічних даних [40].

Для навчання та оцінки моделі використано публічний набір даних «Chest X-Ray: Pneumonia, COVID-19, Tuberculosis», розміщений на платформі Kaggle [41]. Датасет є агрегованим – він об'єднує рентгенограми грудної клітки з кількох верифікованих медичних джерел та охоплює чотири клінічно значущі класи, що є основними об'єктами диференційної діагностики. Детальний опис набору даних наведено у таблиці 3.1.

Таблиця 3.1 – Характеристика консолідованого набору даних

Клас	Тренувальна вибірка (Train)	Тестова вибірка (Test)	Всього	Частка
COVID-19	460	106	566	8.9%
NORMAL	1341	234	1575	24.9%
PNEUMONIA	3875	390	4265	67.4%
TUBERCULOSIS	650	40	690	10.9%
Всього	6326	770	7096	100%

Набір даних попередньо розділений авторами на тренувальну та тестову вибірки у співвідношенні приблизно 89% до 11%, що є стандартною

практикою для задач класифікації медичних зображень [42]. Розподіл класів за вибірками наведено на рисунку 3.1.

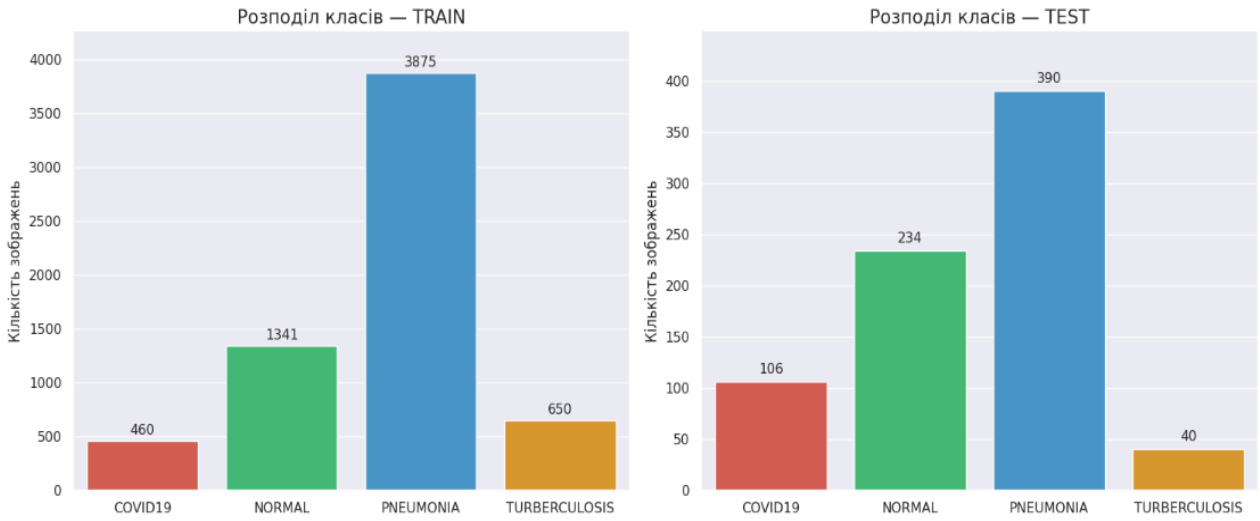

Рисунок 3.1 – Розподіл кількості зображень за класами у тренувальній та тестовій вибірка

3.1.2 Розвідувальний аналіз даних (EDA)

Перед побудовою моделі проведено розвідувальний аналіз даних (Exploratory Data Analysis, EDA), метою якого є виявлення характерних властивостей набору даних, що можуть впливати на процес навчання.

- **Візуальний аналіз знімків.** На рисунку 3.2 наведено по п'ять репрезентативних прикладів рентгенограм кожного класу. Аналіз зображень виявляє суттєву візуальну схожість між класами – особливо між NORMAL та PNEUMONIA на ранніх стадіях захворювання, що підтверджує складність задачі та необхідність застосування потужних архітектур глибокого навчання. Рентгенограми класу COVID-19 характеризуються двосторонніми інфільтратами, переважно у нижніх частках легень, тоді як TUBERCULOSIS виявляється специфічними вогнищевими змінами у верхніх частках.

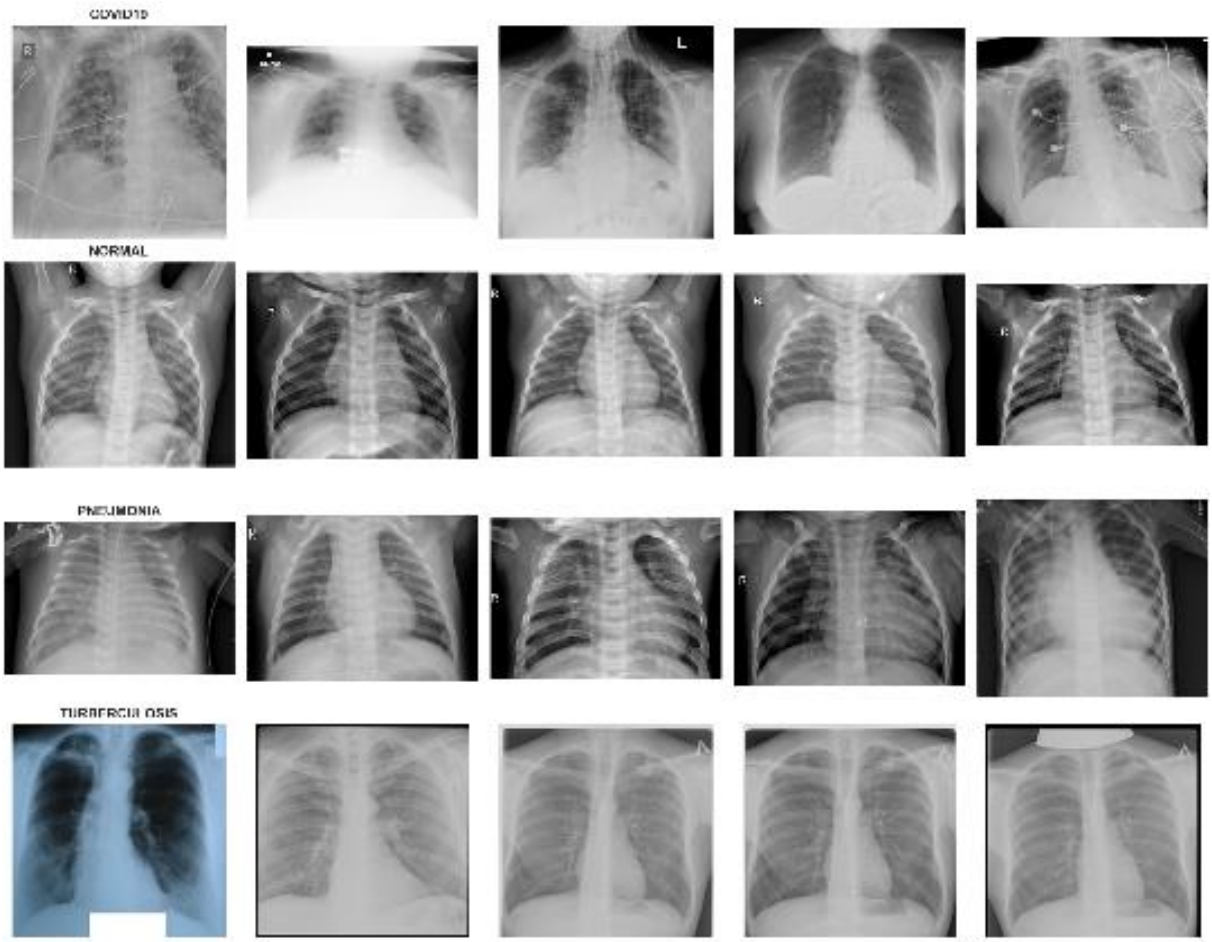

Рисунок 3.2 – Приклади рентгенограм грудної клітки за класами

- **Аналіз розмірів зображень.** Як показано на рисунку 3.3, оригінальні рентгенограми мають суттєво різні розміри – ширина варіюється від 500 до понад 4000 пікселів, а висота від 500 до 3500 пікселів. Така неоднорідність є типовою для реальних клінічних даних, отриманих з різних рентгенографічних апаратів, і безпосередньо обумовлює необхідність стандартизації розміру (resize) на етапі препроцесингу.
- **Аналіз яскравості пікселів.** Рисунок 3.4 демонструє розподіл середньої яскравості пікселів для кожного класу. Виявлено статистично значущі відмінності: рентгенограми класів COVID-19 та TUBERCULOSIS мають вищу медіанну яскравість (близько 135–145 одиниць) порівняно з NORMAL та PNEUMONIA (близько 120 одиниць). Ця відмінність пояснюється патологічними змінами – інфільтрати та ущільнення легеневої тканини відображаються як світліші ділянки. Наявність

вимірних відмінностей на рівні базових пікселів є позитивним індикатором, який свідчить про те, що класи є роздільними не лише за семантичними ознаками, але й за низькорівневими статистиками [43].

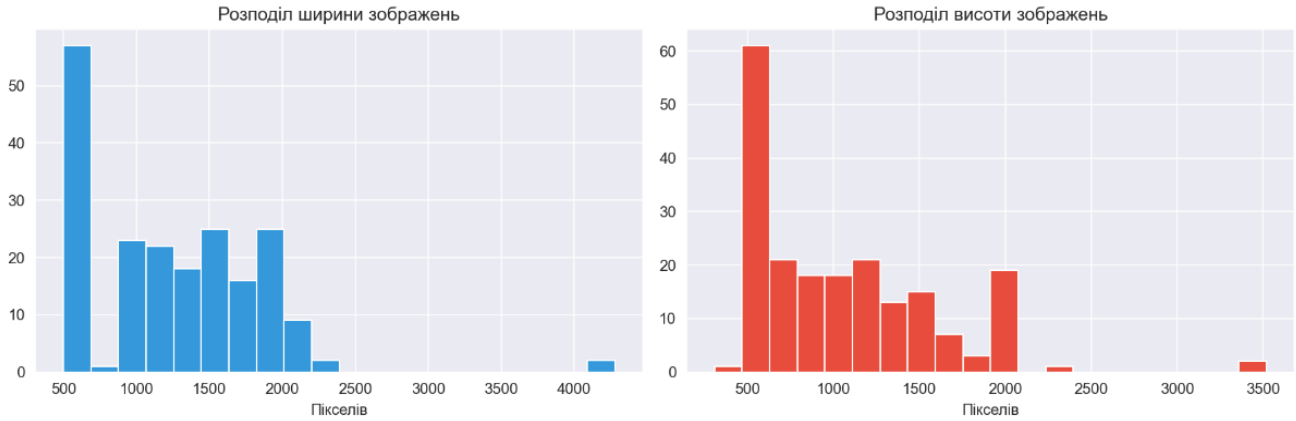

Рисунок 3.3 – Розподіл ширини та висоти оригінальних рентгенограм

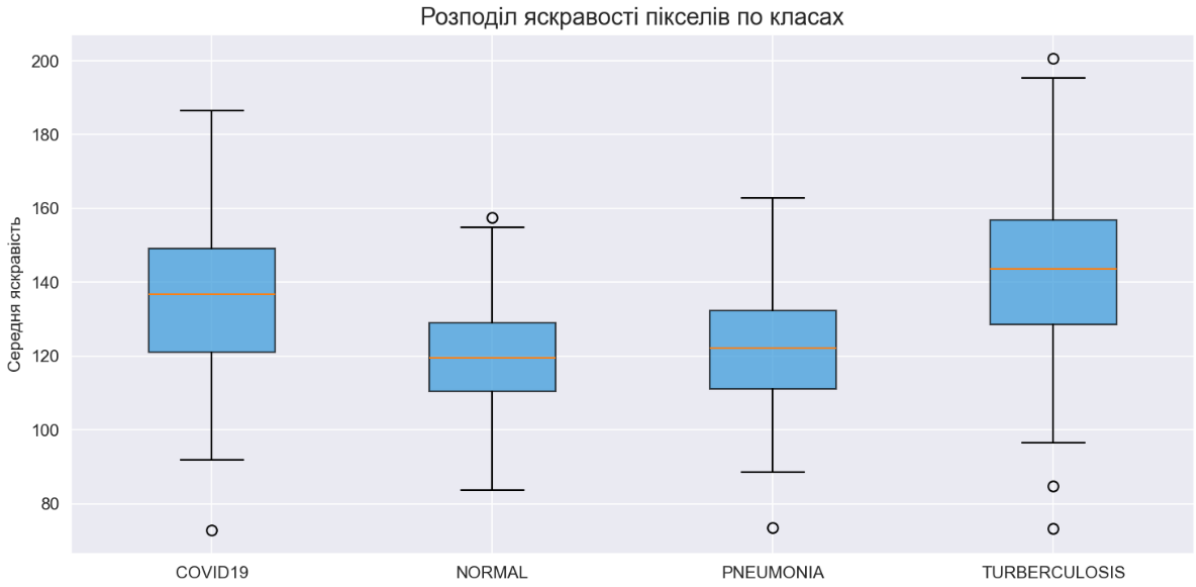

Рисунок 3.4 – Розподіл середньої яскравості пікселів за класами

- **Аналіз дисбалансу класів.** Критичною проблемою, виявленою під час EDA, є суттєвий класовий дисбаланс. Клас PNEUMONIA містить 3875 тренувальних прикладів, що у 8.4 раза перевищує кількість прикладів класу COVID-19 (460). Навчання моделі на незбалансованих даних без застосування коригувальних заходів призводить до зміщення

передбачень у бік домінуючого класу – модель досягає прийнятної загальної точності, але демонструє критично низьку чутливість для рідкісних класів [44]. Стратегія розв'язання цієї проблеми описана у підрозділі 3.1.3.

3.1.3 Препроцесинг та аугментація даних

Повний алгоритм препроцесингу наведено у вигляді UML-діаграми діяльностей на рисунку 3.5. Конвеєр обробки включає кілька послідовних етапів та розгалужується залежно від призначення вибірки.

1. **Стандартизація розміру.** На першому етапі всі зображення незалежно від оригінального розміру конвертуються у формат RGB (3 канали) та масштабуються до розміру 224×224 пікселів за допомогою білінійної інтерполяції. Розмір 224×224 обраний як стандартний вхідний вимір для архітектур сімейства EfficientNet [45].
2. **Аугментація тренувальної вибірки.** Для тренувальної вибірки застосовано набір геометричних та фотометричних аугментацій, що є одним з ключових інструментів боротьби з перенавчанням та покращення узагальнювальної здатності моделі [46]:
   - `RandomHorizontalFlip` (p=0.5) – випадкове горизонтальне відображення з імовірністю 50%. Клінічно коректна аугментація, оскільки патологічні зміни легень можуть бути симетричними;
   - `RandomRotation` (±15°) – випадковий поворот у діапазоні ±15 градусів, що імітує варіацію позиціонування пацієнта під час знімання;
   - `ColorJitter` – випадкова зміна яскравості та контрасту у межах ±20%, що імітує варіації налаштувань рентгенографічного апарату;
   - `RandomAffine` – випадковий паралельний зсув до 10% розміру зображення, що імітує варіації центрування знімка.

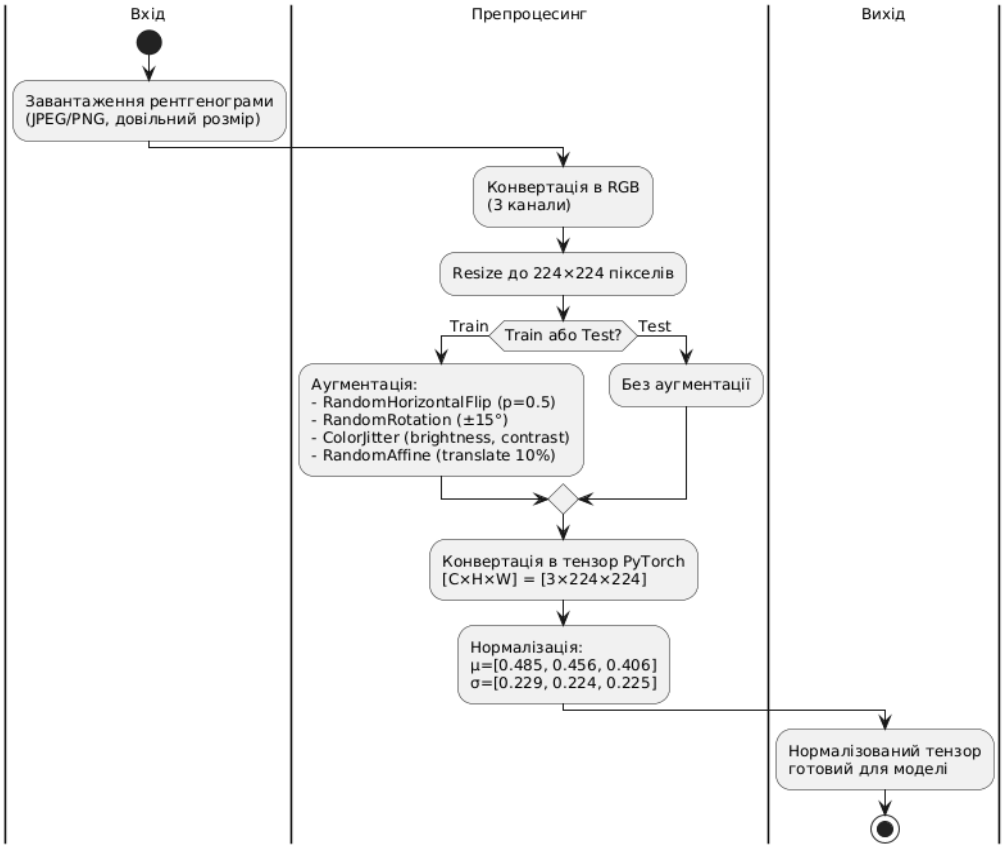

Рисунок 3.5 – UML-діаграма діяльностей конвеєра препроцесингу та аугментації даних

На рисунку 3.6 наведено дев'ять варіантів одного знімка після застосування аугментацій, що наочно демонструє різноманітність трансформацій. Тестова вибірка не піддається аугментації – для об'єктивної оцінки використовуються оригінальні зображення.

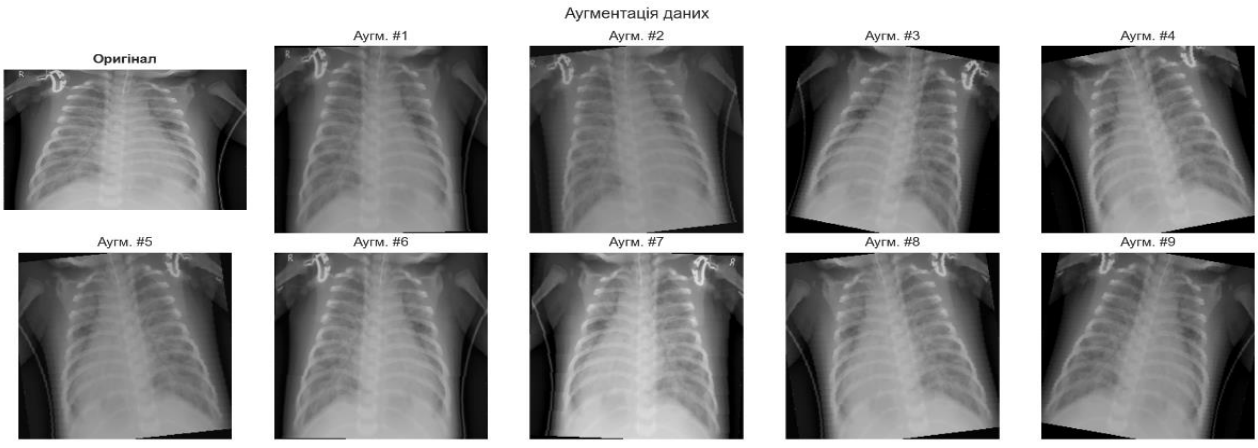

Рисунок 3.6 – Приклади аугментованих варіантів однієї рентгенограми

3. **Нормалізація.** Завершальним етапом є конвертація зображення у тензор PyTorch формату [3, 224, 224] та нормалізація значень пікселів за формулою 3.1:

$$x_{norm} = \frac{x - \mu}{\sigma} \quad , \tag{3.1}$$

де $\mu = [0.485, 0.456, 0.406]$ та $\sigma = [0.229, 0.224, 0.225]$ – значення математичного очікування та стандартного відхилення каналів RGB, обчислені на наборі даних ImageNet.

Використання цих статистик є стандартом при Transfer Learning, оскільки попередньо навчена модель очікує дані в тому ж діапазоні, на якому вона навчалась [45].

4. **Балансування класів.** Для розв'язання проблеми дисбалансу класів застосовано механізм зваженої вибірки (WeightedRandomSampler). Кожному тренувальному прикладу присвоюється вага, обернено пропорційна кількості прикладів його класу , згідно з формулою (3.2):

$$w_i = \frac{1}{N_c} \quad , \tag{3.2}$$

де N_c – кількість прикладів класу c.

Під час формування кожного тренувального мінібатчу зразки вибираються стохастично, відповідно до розрахованих ваг, що гарантує збалансоване та рівномірне представлення всіх класів у процесі навчання. Це дозволяє ефективно уникнути зміщення моделі у бік найбільш представленого класу. Крім того, на відміну від методів фізичного дублювання (oversampling), такий підхід не потребує збільшення обсягу даних на диску та знижує ризик перенавчання (overfitting), оскільки не вносить штучних артефактів у вибірку [44].

3.2 Порівняльний аналіз архітектур нейромереж із застосуванням Transfer Learning

3.2.1 Концепція Transfer Learning та метрики оцінювання

Навчання глибоких нейронних мереж з нуля вимагає мільйонів розмічених прикладів та значних обчислювальних ресурсів. У медичній візуалізації збір та розмітка клінічних даних є надзвичайно трудомістким процесом, що потребує участі кваліфікованих спеціалістів та суворого дотримання вимог конфіденційності пацієнтів. Технологія перенесення навчання (Transfer Learning) розв'язує цю проблему шляхом використання знань, отриманих моделлю під час вирішення однієї задачі, для ефективнішого навчання на іншій, спорідненій задачі [47]. Дослідження Raghu et al. підтвердило, що Transfer Learning з набору даних ImageNet забезпечує кращу початкову точність та швидшу збіжність порівняно з навчанням з нуля навіть для медичних зображень, які суттєво відрізняються від природних фотографій [48].

У даній роботі застосовано стратегію виділення ознак (Feature Extraction). Базові шари моделі, що була попередньо навчена на наборі даних ImageNet (1.2 мільйона зображень, 1000 класів), повністю заморожуються задля збереження сформованих ваг та оптимізації обчислювальних ресурсів. Навчається лише нова класифікаційна голівка, що складається з шарів `Dense(256) → ReLU → Dropout(0.3) → Dense(4) → Softmax`, яка адаптована до чотирьох цільових класів. При цьому шар регуляризації Dropout інтегровано для запобігання ефекту коадаптації ознак та перенавчання на фінальних етапах класифікації. Обґрунтування застосування ваг ImageNet для медичних зображень полягає у тому, що низькорівневі ознаки – детектори країв, текстур та базових форм – є універсальними та транспортабельними між різними доменами [49]. Для забезпечення об'єктивного порівняння всі три архітектури навчались в ідентичних умовах, наведених у таблиці 3.2.

Таблиця 3.2 – Конфігурація навчання для порівняльного аналізу

Параметр	Значення
Розмір мінібатчу (Batch Size)	32
Оптимізатор	Adam
Швидкість навчання (Learning rate)	0.001
Кількість епох	10
Функція втрат	`CrossEntropyLoss`
Механізм балансування	`WeightedRandomSampler`
Розмір вхідного зображення	224×224 пікселів
Апаратне забезпечення	NVIDIA RTX 4060 Ti (CUDA)

Зважаючи на виявлений раніше суттєвий дисбаланс класів (від 460 прикладів COVID-19 до 3875 прикладів пневмонії), використання виключно загальної точності (Accuracy) є методологічно некоректним – модель, що завжди передбачає мажоритарний клас пневмонії, матиме Accuracy близько 50%, не навчившись нічого корисного. Для комплексного оцінювання використано розширений набір метрик на основі матриці похибок [50].

Загальна точність класифікації обчислюється за формулою (3.3):

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad , \tag{3.3}$$

де TP (True Positive) – кількість істинно позитивних результатів;

TN (True Negative) – кількість істинно негативних результатів;

FP (False Positive) – кількість хибно позитивних результатів;

FN (False Negative) – кількість хибно негативних результатів.

Точність (Precision) алгоритму розраховується за формулою (3.4):

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad , \tag{3.4}$$

де Precision(точність) характеризує здатність моделі уникати хибних тривог. Низька точність означає, що алгоритм генерує надмірну кількість хибнопозитивних діагнозів, що призводить до невиправданих додаткових обстежень та психологічного стресу пацієнтів.

Повнота (Recall) моделі обчислюється за формулою (3.5):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad , \tag{3.5}$$

де Recall (чутливість, повнота) характеризує здатність моделі виявляти всіх хворих пацієнтів, мінімізуючи пропуски захворювань. У контексті медичної діагностики ця метрика є критично важливішою, ніж Precision, оскільки медична помилка другого роду (хибнонегативний результат, FN), коли модель пропускає реально хвору людину, несе пряму загрозу її життю та здоров'ю [51].

Для збалансованого оцінювання застосовується F1-міра, що розраховується за формулою (3.6):

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad , \tag{3.6}$$

де F1-score є гармонійним середнім між точністю та повнотою, що забезпечує збалансовану оцінку при дисбалансі класів. На відміну від арифметичного середнього, ця метрика жорстко штрафує моделі з різко асиметричними значеннями.

Для загальної оцінки моделей застосовано макро-усереднення (Macro-Average), що обчислює метрики незалежно для кожного класу з подальшим

знаходженням їхнього незваженого середнього. Цей підхід надає однакову вагу кожному класу незалежно від його частоти, що є коректним для медичної задачі, де рідкісний випадок туберкульозу є не менш важливим за поширену пневмонію [50].

3.2.2 Архітектура VGG16

Архітектура VGG16, розроблена групою Visual Geometry Group Оксфордського університету у 2014 році, продемонструвала практичну цінність збільшення глибини мережі за рахунок використання виключно малих згорткових фільтрів 3×3 [52]. Архітектуру моделі наведено на рисунку 3.7.

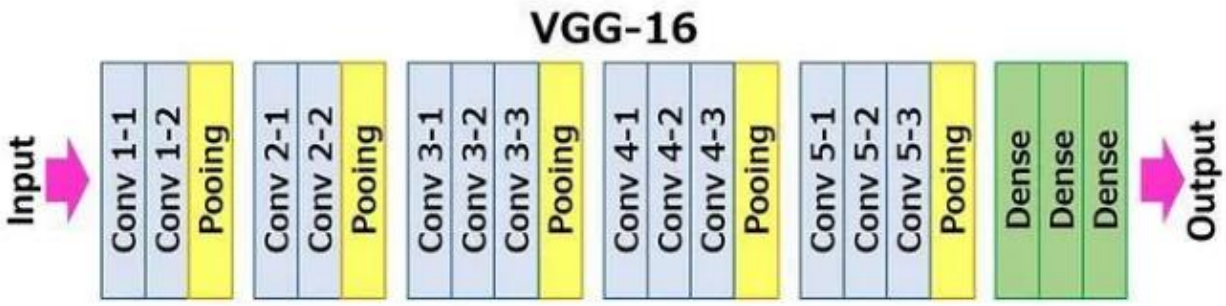

Рисунок 3.7 – Архітектура згорткової нейронної мережі VGG16 [53]

Мережа складається з п'яти блоків згорткових шарів із пулінгом (MaxPooling) між ними та трьох повнозв'язних шарів на виході. Стек із двох фільтрів 3×3 має те саме рецептивне поле, що й один фільтр 5×5, але містить менше параметрів та забезпечує більше нелінійностей. Попри концептуальну простоту, архітектура має близько 138 мільйонів параметрів, що робить її надзвичайно ресурсомісткою. Криві навчання та матрицю похибок наведено на рисунках 3.8 та 3.9.

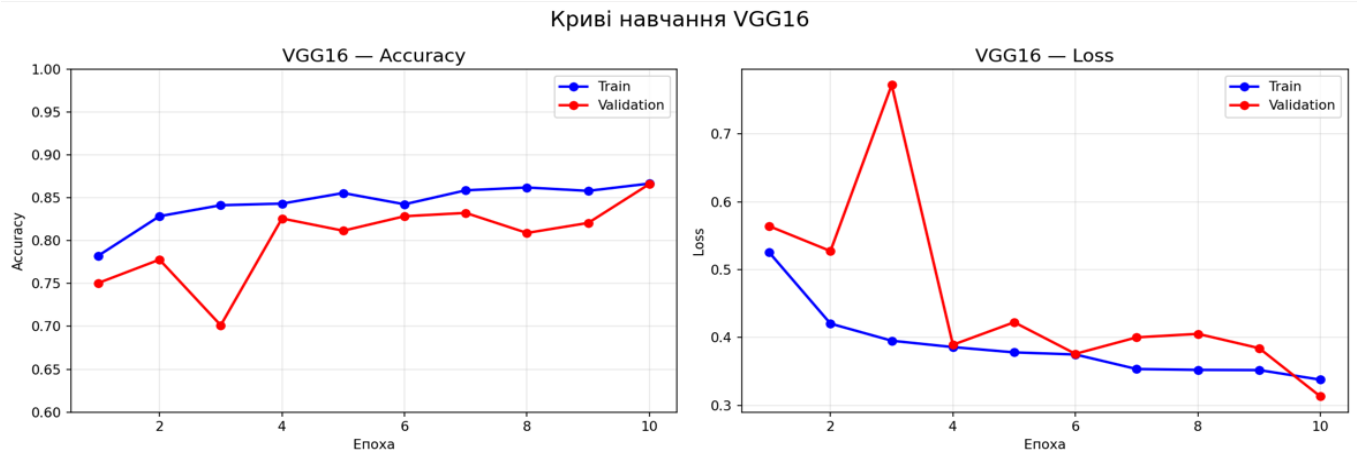

Рисунок 3.8 – Криві навчання архітектури VGG16 (Accuracy та Loss)

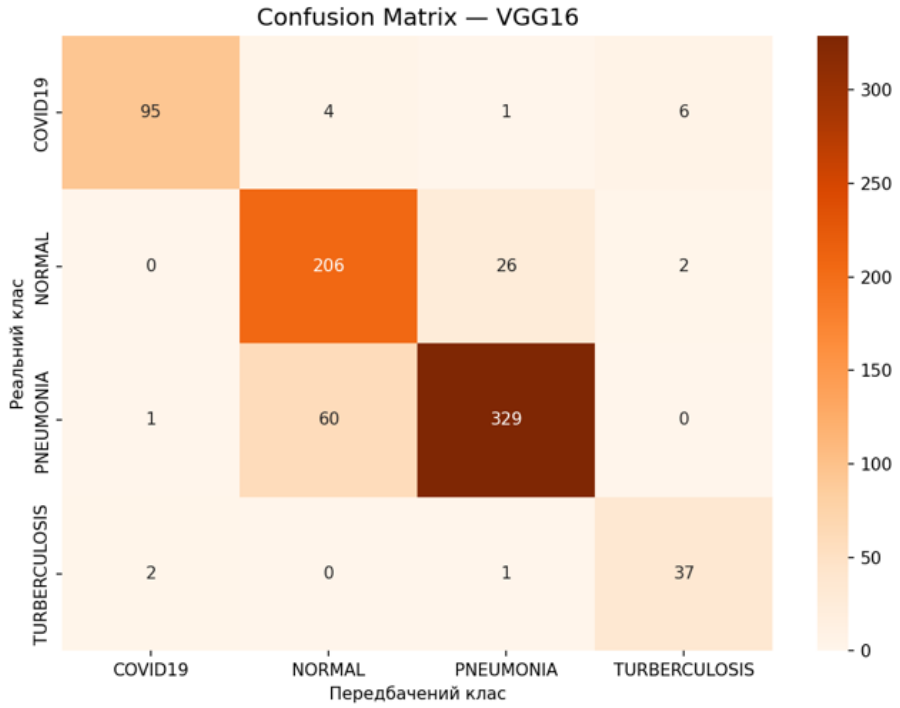

Рисунок 3.9 – Матриця похибок (Confusion Matrix) моделі VGG16

Характерною особливістю кривих навчання є різкий стрибок валідаційного Loss на третій епосі до значення 0.77, після чого відбувається поступова стабілізація. Така нестабільність пояснюється відсутністю залишкових з'єднань, що ускладнює оптимізацію при заморожених базових шарах. Загальна точність моделі на тестовій вибірці склала 87%. Макро-усереднені показники: Precision – 87%, Recall – 89%, F1-score – 88%. Попри

прийнятні загальні показники, модель демонструє посередню точність (Precision = 0.76) для класу NORMAL, хибно ідентифікуючи значну кількість здорових знімків як патологічні – 60 знімків пневмонії класифіковано як норму, що є клінічно небезпечною помилкою (хибнонегативний результат).

3.2.3 Архітектура ResNet50

Мережа ResNet50, розроблена командою Microsoft Research у 2015 році, розв'язала фундаментальну проблему деградації точності при збільшенні глибини за допомогою залишкових з'єднань (shortcut connections), що передають вхідний сигнал безпосередньо до виходу блоку, оминаючи кілька шарів [54]. Архітектуру наведено на рисунку 3.10.

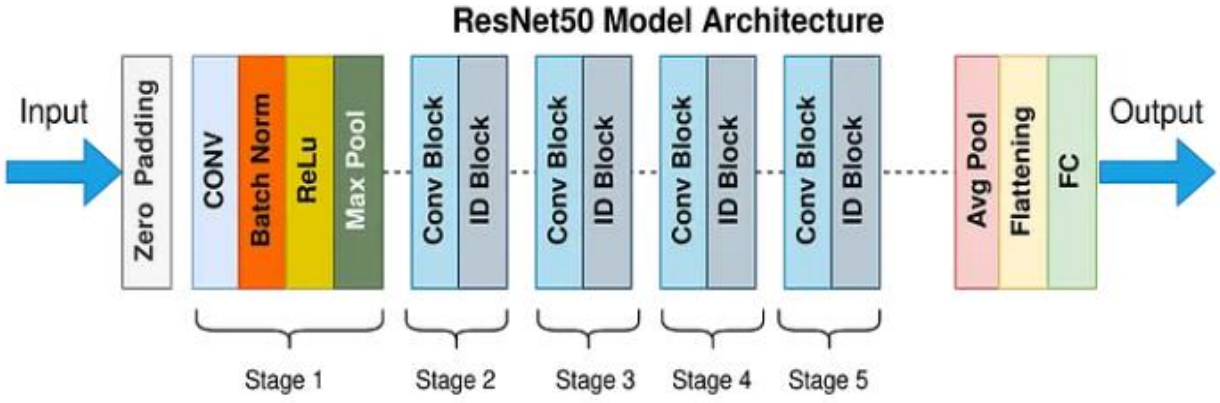

Рисунок 3.10 – Архітектура згорткової нейронної мережі ResNet50 [55]

Якщо блок не навчився нічому корисному, градієнт може безперешкодно протікати через залишкові з'єднання, що розв'язує проблему зникаючих градієнтів у глибоких мережах. ResNet50 містить 25 мільйонів параметрів – у 5.5 раза менше, ніж VGG16, що суттєво пришвидшує швидкодію. Криві навчання та матрицю похибок наведено на рисунках 3.11 та 3.12.

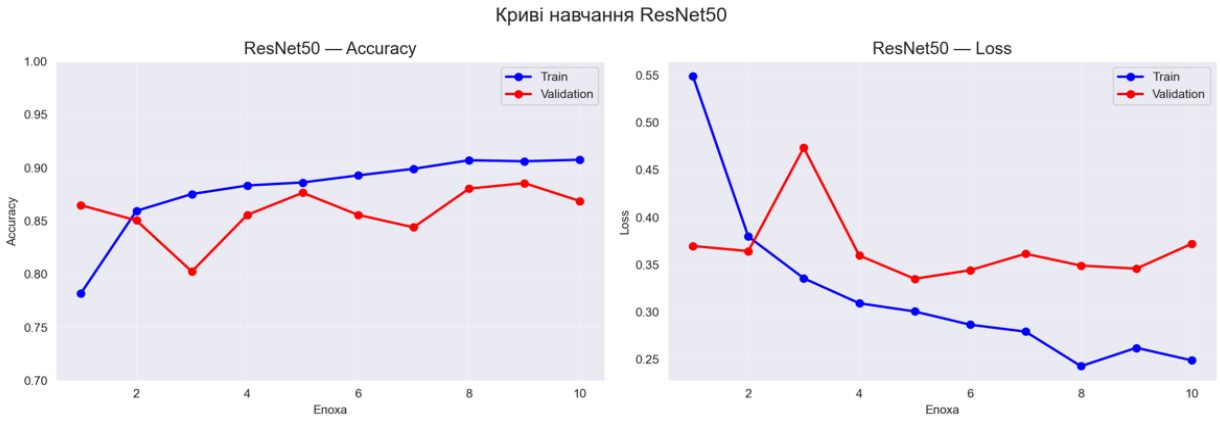

Рисунок 3.11 – Криві навчання архітектури ResNet50 (Accuracy та Loss)

Порівняно з VGG16, криві навчання демонструють значно вищу стабільність – відсутні різкі стрибки функції втрат. Загальна точність становить 87%. Макро-усереднені метрики: Precision – 85%, Recall – 89%, F1-score – 86%. Аналіз детальних метрик виявляє специфічну проблему – низький показник Precision для класу TUBERCULOSIS (лише 0.69). Це означає, що модель надмірно часто хибно класифікує інші патології як туберкульоз, що може призвести до зайвого навантаження на етап додаткової діагностики та невиправданого психологічного стресу пацієнтів.

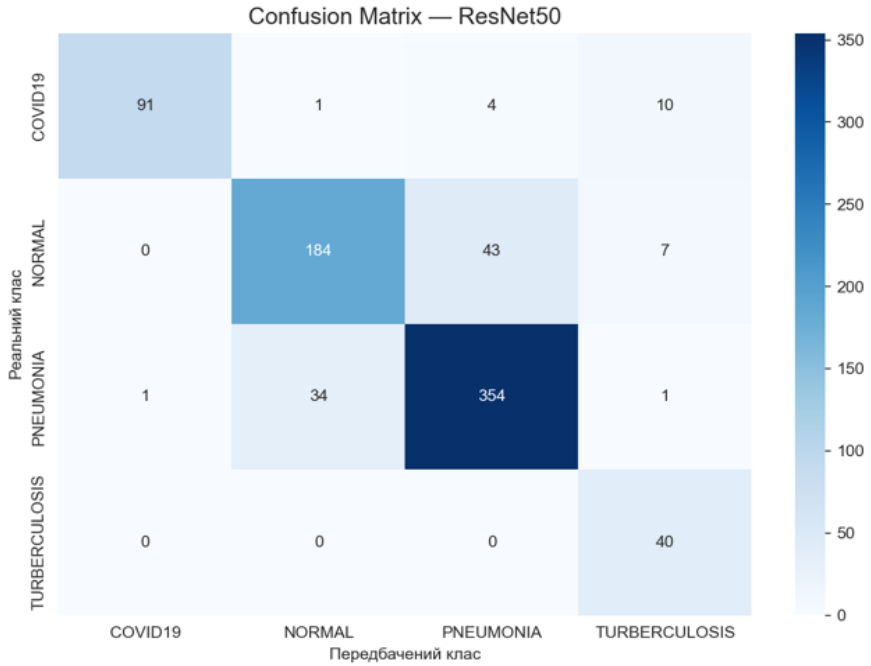

Рисунок 3.12 – Матриця похибок (Confusion Matrix) моделі ResNet50

3.2.4 Архітектура EfficientNetB3

Архітектура EfficientNet, розроблена командою Google Brain у 2019 році, запропонувала принципово новий підхід до масштабування нейронних мереж – метод комплексного масштабування (compound scaling), що одночасно збільшує глибину d, ширину w та роздільну здатність r вхідного зображення за єдиним коефіцієнтом φ [56], згідно з формулою (3.7):

$$d = \alpha^{\varphi}, \quad w = \beta^{\varphi}, \quad r = \gamma^{\varphi} \quad , \tag{3.7}$$

де α, β, γ – константи, що визначаються співвідношенням $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$;

φ – визначений користувачем коефіцієнт (для варіанта B3 $\varphi = 3$).

Архітектуру моделі наведено на рисунку 3.13.

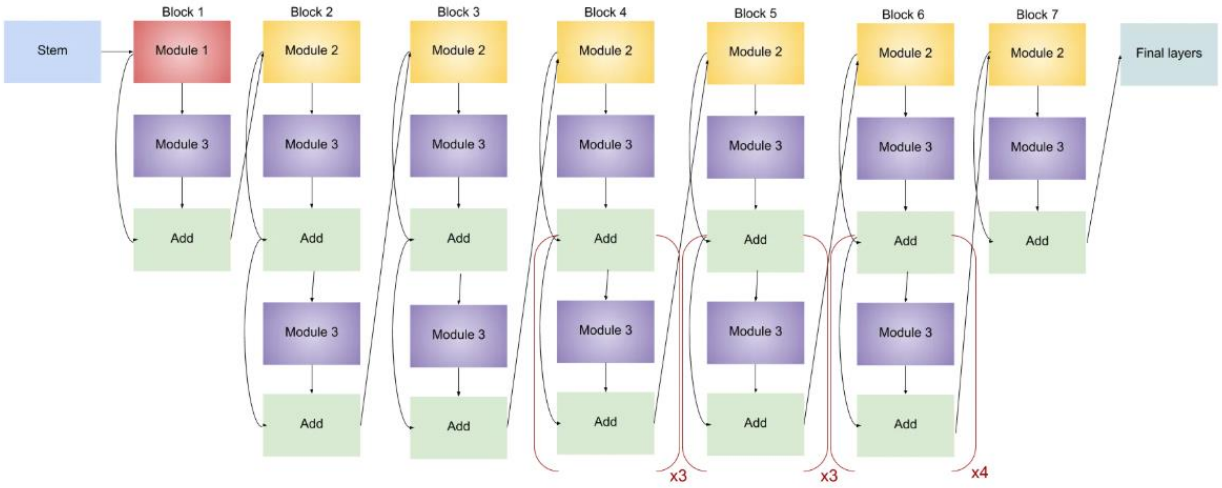

Рисунок 3.13 – Архітектура EfficientNetB3 з блоками MBConv [57]

Мережа складається з початкового шару (Stem) та семи блоків мобільних інвертованих залишкових блоків (MBConv) з наявністю skip connections між ними. EfficientNetB3 містить лише 12 мільйонів параметрів, перевершуючи VGG16 за точністю на ImageNet (81.6%), маючи при цьому в 11.5 рази меншу ресурсомісткість. Криві навчання та матрицю похибок наведено на рисунках 3.14 та 3.15.

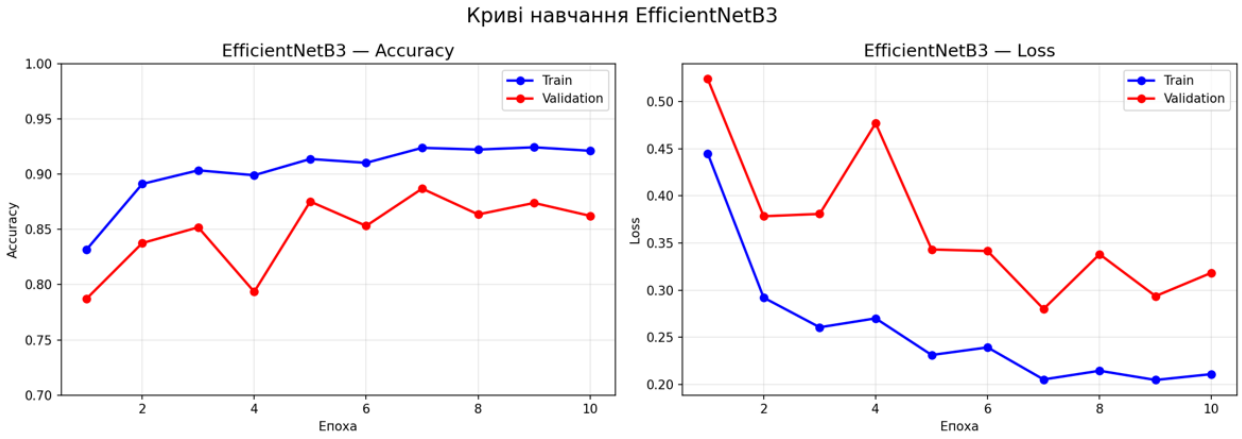

Рисунок 3.14 – Криві навчання архітектури EfficientNetB3 (Accuracy та Loss)

EfficientNetB3 демонструє найкращу динаміку навчання – тренувальна точність стрімко зростає до 92% вже на третій епосі, а валідаційний Loss монотонно спадає без різких стрибків, що свідчить про кращу узагальнювальну здатність. Загальна точність на тестовій вибірці склала 86%. Проте ключовою перевагою є макро-усереднені показники: Precision – 86%, Recall – 91%, F1-score – 88%. Модель досягла бездоганної повноти (Recall = 1.00) для класу TUBERCULOSIS та видатного показника (Recall = 0.97) для COVID-19 – двох найнебезпечніших з епідеміологічної точки зору патологій.

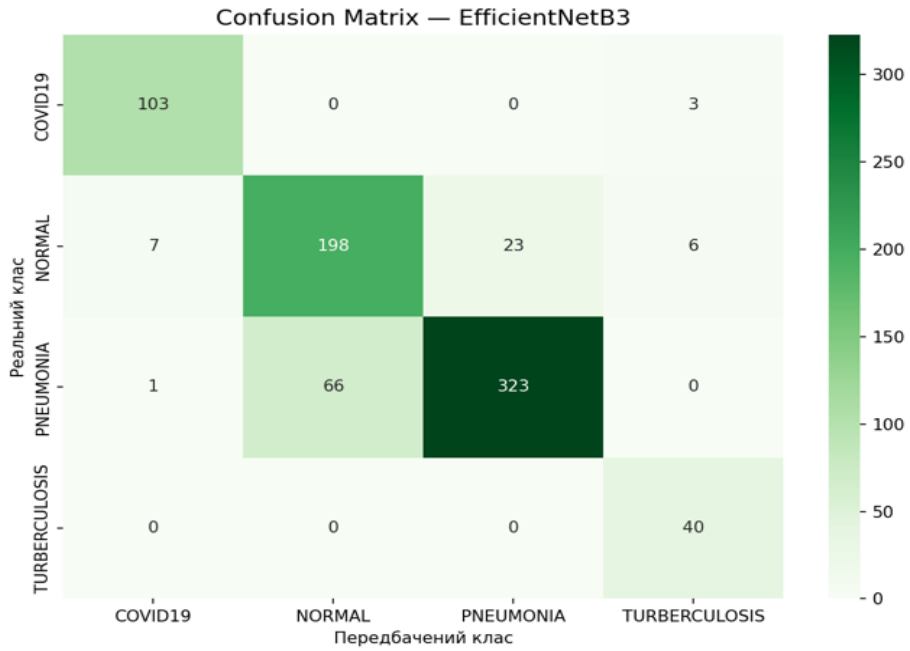

Рисунок 3.15 – Матриця похибок (Confusion Matrix) моделі EfficientNetB3

3.2.5 Порівняльний аналіз результатів

Зведені результати порівняльного аналізу наведено на рисунку 3.16 та у таблиці 3.3.

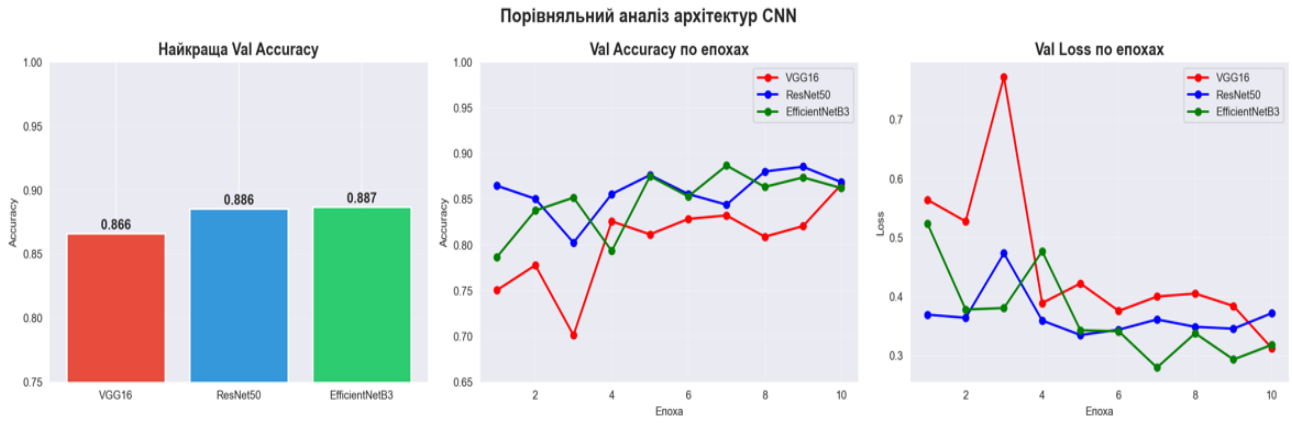

Рисунок 3.16 – Порівняльний аналіз трьох архітектур CNN

Таблиця 3.3 – Комплексні результати порівняльного аналізу архітектур CNN

Архітектура	Кількість параметрів	Accuracy	Precision (Macro)	Recall (Macro)	F1-score (Macro)
VGG16	138 млн	87%	87%	89%	88%
ResNet50	25 млн	87%	85%	89%	86%
EfficientNetB3	12 млн	86%	86%	91%	88%

Аналіз таблиці 3.3 доводить, що загальна точність (Accuracy) у всіх трьох моделей знаходиться на схожому рівні (86–87%). Однак у контексті розробки медичного комплексу діагностики критично важливою є повнота (Recall), оскільки медична помилка другого роду – хибнонегативний результат, коли система пропускає хворого пацієнта, несе пряму загрозу його здоров'ю та життю. Орієнтація виключно на Accuracy під час вибору моделі для медичного застосування є методологічно некоректною та потенційно небезпечною [51].

За показником макро-усередненої повноти беззаперечним лідером є EfficientNetB3 (Recall = 91%). Вона продемонструвала виняткову чутливість до клінічно критичних класів – COVID-19 та TUBERCULOSIS, успішно розпізнавши 100% випадків туберкульозу. Водночас ця архітектура є найоптимальнішою з точки зору обчислювальної ефективності – лише 12 мільйонів параметрів проти 138 мільйонів у VGG16, що забезпечує швидшу роботу моделі та менше навантаження на сервер під час інтеграції у веб-застосунок.

На підставі проведеного комплексного аналізу метрик та ресурсних вимог, для фінального етапу донавчання (Fine-tuning) та використання у розробленому програмному комплексі обрано архітектуру EfficientNetB3, як таку, що забезпечує найвищу чутливість до клінічно критичних класів за мінімальних обчислювальних витрат.

3.3 Донавчання (Fine-Tuning) та оцінка точності фінальної моделі

3.3.1 Теоретичне обґрунтування стратегії Fine-Tuning

Незважаючи на те, що стратегія Feature Extraction забезпечила прийнятні результати у діапазоні 86–89%, вона має принципове обмеження – заморожені базові шари моделі зберігають ознаки, оптимізовані для класифікації природних фотографій (ImageNet), а не медичних рентгенограм. Високорівневі шари мережі, що відповідають за розпізнавання складних семантичних концептів, залишаються незмінними і не адаптуються до специфічних патернів медичних зображень: інфільтратів легеневої тканини, вогнищевих змін та характерних ущільнень [58].

Донавчання (Fine-Tuning) є другим, більш глибоким етапом Transfer Learning, під час якого (після початкового навчання класифікаційної голівки) частково розморожуються базові шари моделі та адаптуються до цільового домену з використанням суттєво меншої швидкості навчання (learning rate). Стратегію Fine-Tuning проілюстровано на рисунку 3.17.

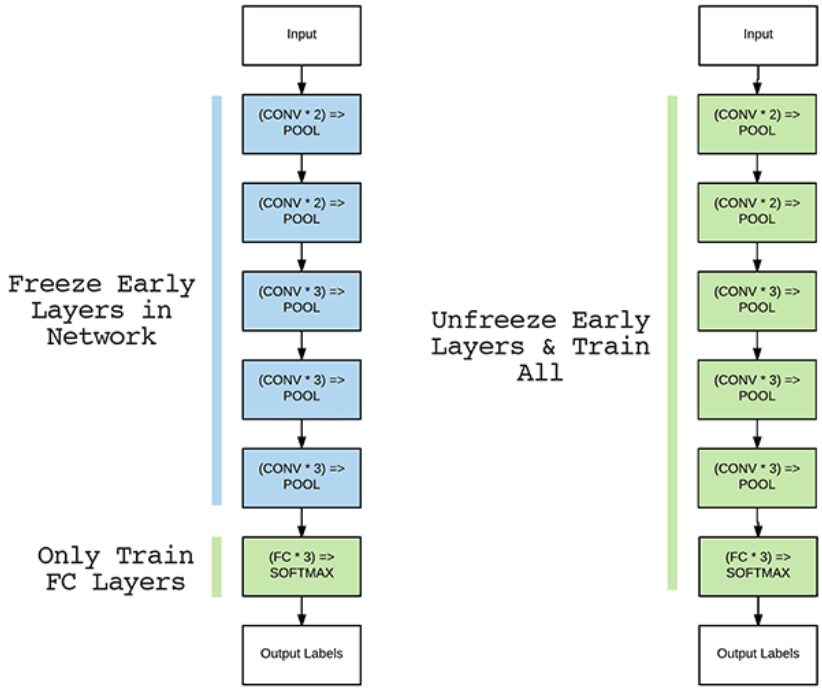

Рисунок 3.17 – Стратегія навчання: Feature Extraction (ліворуч) та Fine-Tuning (праворуч) [59]

Як видно з рисунка 3.17, при Feature Extraction навчаються виключно повнозв'язні шари (FC Layers), тоді як усі згорткові блоки залишаються замороженими. При Fine-Tuning глибокі шари стають тренованими, що дозволяє моделі тонко адаптувати вивчені ознаки до специфіки медичних зображень.

Ключовим архітектурним питанням є вибір кількості шарів, що підлягають розморожуванню. Розморожування всіх шарів одночасно є недоцільним з двох причин. По-перше, низькорівневі ознаки перших блоків (детектори країв, базових текстур) є універсальними та однаково корисними для будь-яких зображень – їх перенавчання лише шкодить стабільності моделі. По-друге, повне розморожування в умовах обмеженого медичного набору даних різко збільшує ризик перенавчання (Overfitting) [60]. Тому в даній роботі розморожено лише останні три блоки архітектури EfficientNetB3, які відповідають за високорівневі ознаки, специфічні для конкретного домену.

Критично важливим параметром є швидкість навчання – під час Fine-Tuning вона повинна бути значно меншою. Використання великого кроку оптимізації зруйнує попередньо навчені ваги, що призведе до катастрофічного забування (catastrophic forgetting) та деградації моделі. У даній роботі застосовано learning_rate = 0.0001 – у 10 разів менший, ніж на попередньому етапі [61].

3.3.2 Конфігурація Fine-Tuning

Порівняльну конфігурацію двох етапів навчання наведено у таблиці 3.4.

Таблиця 3.4 – Порівняння конфігурацій Feature Extraction та Fine-Tuning

Параметр	Feature Extraction	Fine-Tuning
Заморожені шари	Всі базові блоки	Блоки 1–4
Розморожені шари	Тільки класифікаційна голова	Блоки 5–7 + голівка
Learning rate	0.001	0.0001
Кількість епох	10	15
Scheduler (Планувальник)	–	ReduceLROnPlateau
Параметрів, що навчаються	~525 тис.	~3.2 млн

Для автоматичного управління швидкістю навчання застосовано планувальник ReduceLROnPlateau – він динамічно зменшує learning rate вдвічі, якщо протягом двох епох поспіль валідаційна метрика не покращується. Це дозволяє моделі автоматично «дотискати» останні відсотки точності без ризику розходження оптимізаційного процесу.

3.3.3 Результати Fine-Tuning

Криві навчання процесу Fine-Tuning наведено на рисунку 3.18.

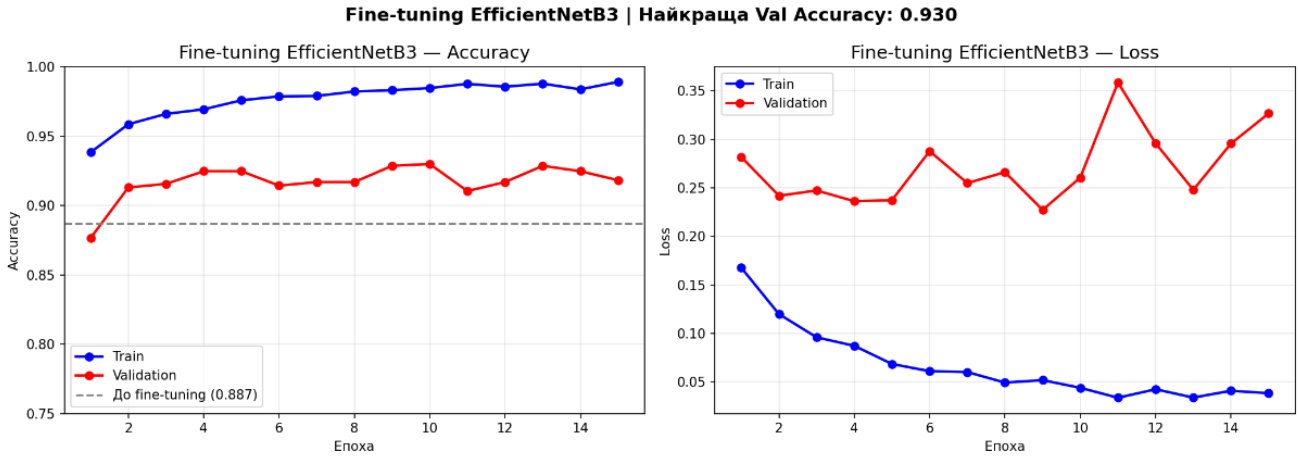

Рисунок 3.18 – Криві навчання моделі EfficientNetB3 на етапі Fine-Tuning

Аналіз кривих навчання виявляє кілька важливих закономірностей. Пунктирна лінія на рисунку 3.18 позначає базовий рівень валідаційної точності до Fine-Tuning (88.7%). Вже з першої епохи донавчання модель перевищує цей показник, досягаючи понад 91%, що підтверджує ефективність обраної стратегії. Тренувальна точність монотонно зростає до 99%, тоді як валідаційна стабілізується на рівні 92–93%. Наявний розрив між тренувальною та валідаційною точністю свідчить про певний ступінь перенавчання, однак завдяки використанню Dropout(0.3) та оптимізації кроку навчання він залишається в контрольованих межах.

Матриця похибок після Fine-Tuning (рисунок 3.19) демонструє разюче покращення відносно базової моделі. Клас COVID-19 класифіковано ідеально – 106 зі 106 знімків (Recall = 100%). Клас TUBERCULOSIS також розпізнано бездоганно – 40 із 40 (Recall = 100%). Основним джерелом помилок залишається пара NORMAL / PNEUMONIA – 54 знімки норми класифіковано як пневмонію. Це є медично пояснюваним явищем, оскільки на ранніх стадіях пневмонія може не мати яскраво виражених рентгенологічних ознак, а артефакти знімання (наприклад, тіні від ребер) можуть інтерпретуватись алгоритмом як інфільтрати. Детальний звіт класифікації наведено у таблиці 3.5.

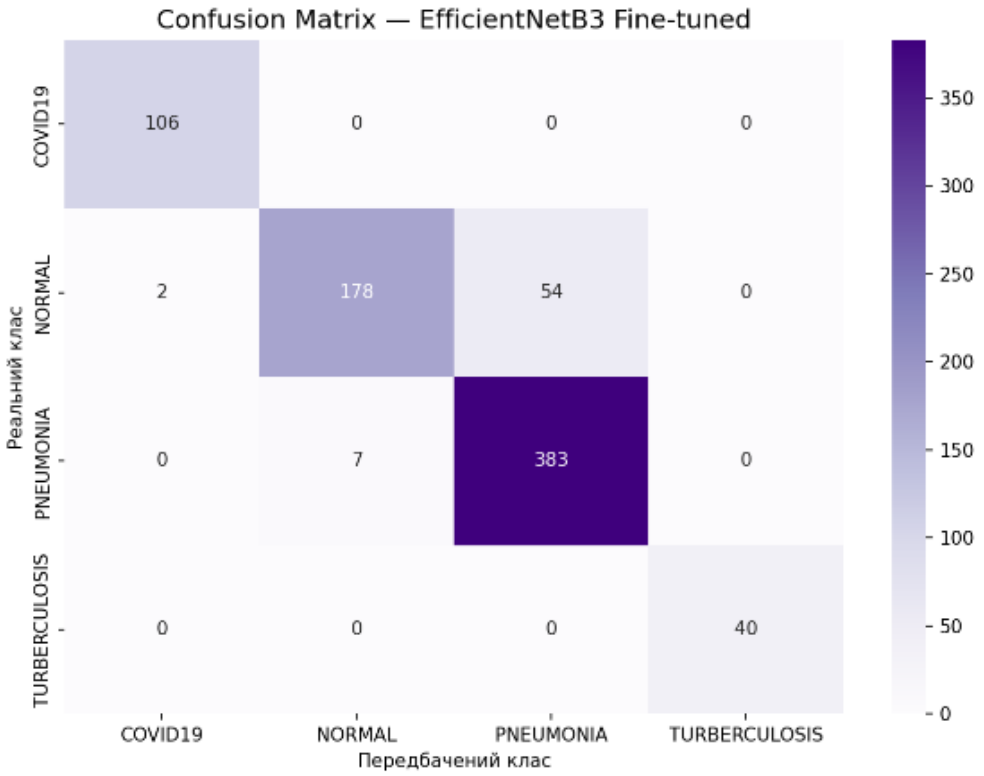

Рисунок 3.19 – Матриця похибок (Confusion Matrix) EfficientNetB3 після Fine-Tuning

Таблиця 3.5 – Звіт класифікації EfficientNetB3 після Fine-Tuning

Клас	Precision	Recall	F1-score	Support
COVID-19	0.98	1.00	0.99	106
NORMAL	0.96	0.76	0.85	234
PNEUMONIA	0.88	0.98	0.93	390
TUBERCULOSIS	1.00	1.00	1.00	40
Accuracy			**0.92**	**770**
Macro avg	0.96	0.94	0.94	770

Аналіз таблиці 3.5 підтверджує видатні результати для клінічно критичних класів. Туберкульоз досяг абсолютних показників (Precision = 1.00, Recall = 1.00) – жоден випадок не був пропущений, і жоден здоровий пацієнт не отримав хибний діагноз туберкульозу. Клас COVID-19 також демонструє бездоганну чутливість (Recall = 1.00) при високій точності (Precision = 0.98). З

медичної точки зору це означає, що система ніколи не пропускає пацієнта з коронавірусною інфекцією, що є критично важливим фактором для запобігання поширенню захворювання.

Єдиним класом із відносно нижчою повнотою розпізнавання є NORMAL (0.76) – 54 здорові знімки класифіковано як пневмонію. З позицій клінічної діагностики така похибка трактується як помилка першого роду (хибнопозитивний результат). Незважаючи на те, що вона призводить до гіпердіагностики, така помилка є значно прийнятнішою порівняно з пропуском реального захворювання (помилкою другого роду), оскільки вона зумовлює лише призначення додаткових уточнювальних обстежень і не становить прямої загрози життю пацієнта. Фінальне порівняння метрик до та після донавчання наведено у таблиці 3.6.

Таблиця 3.6 – Порівняння EfficientNetB3 до та після Fine-Tuning

Метрика	До Fine-Tuning	Після Fine-Tuning	Приріст
Accuracy	86%	92%	+6%
Precision (Macro)	86%	96%	+10%
Recall (Macro)	91%	94%	+3%
F1-score (Macro)	88%	94%	+6%
COVID-19 Recall	97%	100%	+3%
Tuberculosis Recall	100%	100%	=
Pneumonia Recall	83%	98%	+15%

Як видно з таблиці 3.6, стратегія Fine-Tuning забезпечила суттєве покращення за всіма ключовими метриками. Найбільший приріст спостерігається для повноти розпізнавання пневмонії (+15%) та макро-усередненої точності (+10%). Загальна точність (Accuracy) зросла з 86% до 92%, що є надзвичайно конкурентним результатом відносно провідних світових систем медичної візуалізації: зокрема, модель CheXNet (Stanford

University) досягає точності 90.3%, а COVID-Net – 91.0% на порівнянних багатокласових задачах [62, 63].

3.4 Інтеграція та візуальна оцінка алгоритму Grad-CAM

3.4.1 Реалізація Grad-CAM для Fine-tuned EfficientNetB3

Теоретичні основи методу Grad-CAM, його математичне обґрунтування та порівняльний аналіз з іншими методами XAI (Explainable AI) були детально розглянуті у першому розділі. У даному підрозділі описано практичну реалізацію методу для донавченої моделі EfficientNetB3 та проведено візуальну оцінку отриманих карт активації.

Ключовим архітектурним рішенням під час реалізації є вибір цільового шару для вилучення карт ознак. Для EfficientNetB3 обрано шар `features[-1]` – останній згортковий блок мережі. Цей вибір обґрунтований фундаментальною властивістю глибоких згорткових мереж – ієрархічністю представлень: ранні шари виявляють низькорівневі універсальні ознаки (краї, текстури), спільні для всіх класів, тоді як останні шари містять високорівневі семантично значущі ознаки, що безпосередньо пов'язані з класифікаційним рішенням [64]. Саме ці ознаки є найбільш інформативними для лікаря.

Принциповою перевагою обраного підходу є використання механізму функцій зворотного виклику (`hooks`) бібліотеки PyTorch. Ці функції автоматично перехоплюють проміжні активації та градієнти під час прямого та зворотного поширення відповідно. Це дозволяє підключитись до внутрішніх шарів моделі без будь-якої модифікації її архітектури або повторного навчання.

Повну реалізацію класу GradCAM засобами бібліотеки PyTorch наведено у Додатку А (лістинг А.1).

Повний алгоритм роботи методу у вигляді UML-діаграми діяльностей наведено на рисунку 3.21.

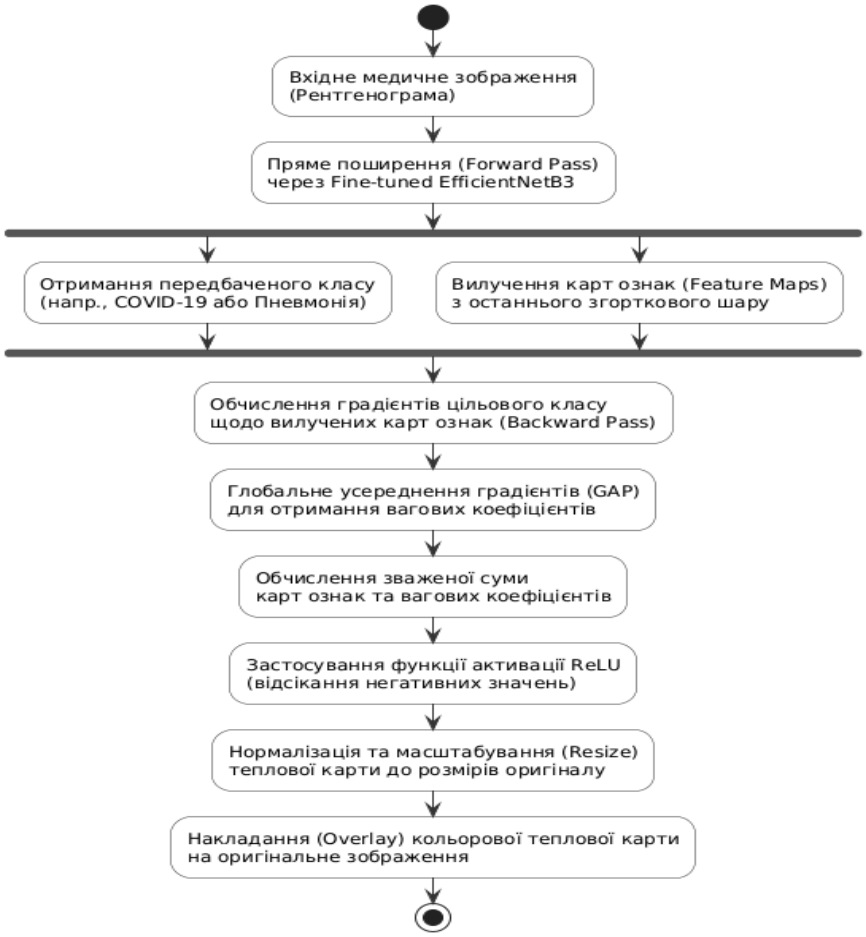

Рисунок 3.21 – UML-діаграма діяльностей алгоритму генерації Grad-CAM теплової карти

Як видно з рисунка 3.21, після прямого поширення (Forward Pass) через Fine-tuned EfficientNetB3 потік виконання розгалужується на дві паралельні гілки – отримання передбаченого класу та вилучення карт ознак з останнього згорткового шару. Після злиття гілок виконується зворотне поширення (Backward Pass) для обчислення градієнтів відносно карт ознак.

Застосування функції активації ReLU є принципово важливим кроком – вона відсікає від'ємні значення, залишаючи лише ті регіони, що позитивно підтверджують належність до цільового класу. Завершальним кроком є накладання нормалізованої теплової карти на оригінальну рентгенограму у вигляді кольорового градієнта.

3.4.2 Візуальна оцінка результатів Grad-CAM

Результати застосування алгоритму Grad-CAM до репрезентативних знімків кожного класу наведено на рисунку 3.22. Для кожного класу показано три зображення: оригінальна рентгенограма, теплова карта Grad-CAM та накладене зображення, де червоний колір (зони найтеплішого спектра) відповідає найбільш дискримінативним регіонам, а синій – найменш важливим для прийнятого рішення.

Аналіз теплових карт є критично важливим кроком валідації моделі – він дозволяє переконатись, що нейронна мережа приймає рішення на основі клінічно значущих ознак, а не артефактів зображення чи технічних міток медичного обладнання [65].

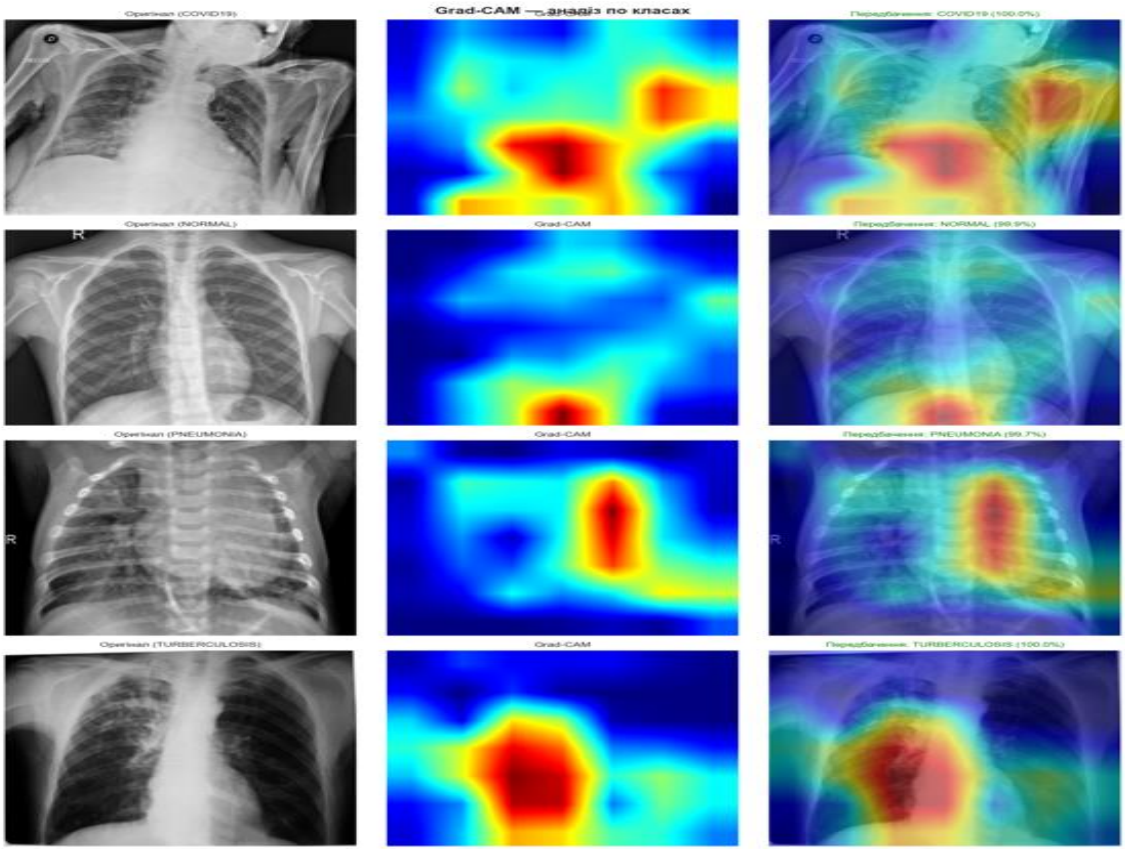

Рисунок 3.22 – Grad-CAM аналіз за класами: оригінал, теплова карта та їх суміщення

- **COVID-19.** Модель формує зони активації у верхній та центральній частинах легеневих полів із характерним білатеральним (двостороннім) розподілом. Така локалізація відповідає типовій рентгенологічній картині пневмонії, викликаної вірусом SARS-CoV-2 – дифузному двосторонньому альвеолярному ураженню [66]. Впевненість моделі 100.0% у поєднанні з клінічно коректною локалізацією підтверджує, що рішення прийнято на основі реальних патологічних ознак.
- **NORMAL.** Для знімків здорових пацієнтів теплова карта розподілена рівномірно по всьому легеневому полю без різко виражених (червоних) вогнищ активації. Це є очікуваним і клінічно коректним результатом – за відсутності патологічних змін модель аналізує загальну структуру легеневого малюнка, не фіксуючись на жодній конкретній ділянці.
- **PNEUMONIA.** Модель концентрує увагу на центральній бронхіальній зоні та прикореневих відділах легень. Характерне вертикальне розташування масивної зони активації вздовж бронхіального дерева відповідає типовій рентгенологічній картині бактеріальної бронхопневмонії та супутньої часткової консолідації.
- **TUBERCULOSIS.** Теплова карта демонструє чіткий локалізований фокус активації у нижній частині правого легеневого поля. Висока впевненість моделі (100.0%) у поєднанні з конкретною просторовою локалізацією підтверджує, що алгоритм виявив специфічну патологічну зону (інфільтрат або каверну), а не прийняв рішення на основі загальних статистичних ознак фону.

Відповідність локалізації теплових карт клінічно очікуваним патологічним зонам для всіх чотирьох класів є ключовим результатом, що підтверджує медичну обґрунтованість алгоритму. Це принципово відрізняє розроблену систему від класичних рішень типу «чорної скриньки» та відповідає найсучаснішим європейським вимогам до медичних систем

штучного інтелекту – кожне діагностичне рішення супроводжується візуальним поясненням, доступним для інтерпретації лікарем [67].

Висновки до розділу 3

У третьому розділі виконано повний цикл розробки, навчання та валідації модуля машинного навчання для диференційної діагностики захворювань легень, що базується на технологіях комп'ютерного зору та глибокого навчання. За результатами проведених досліджень сформульовано такі ключові висновки:

1. **Аналіз та підготовка даних.** На основі розвідувального аналізу (EDA) консолідованого набору медичних знімків (7096 рентгенограм) виявлено суттєву гетерогенність характеристик та класовий дисбаланс. Для нівелювання цих факторів спроєктовано оптимізований конвеєр попередньої обробки, що включає просторове масштабування, нормалізацію, стохастичну аугментацію та зважене балансування класів (WeightedRandomSampler), що дозволило запобігти перенавчанню моделі.
2. **Вибір базової архітектури.** Здійснено порівняльний аналіз трьох згорткових архітектур (VGG16, ResNet50, EfficientNetB3) із застосуванням стратегії Transfer Learning (Feature Extraction). Теоретично та практично обґрунтовано, що метрика загальної точності (Accuracy) є нерепрезентативною для медичних завдань. За критичною метрикою повноти розпізнавання (Recall) найкращою виявилась архітектура EfficientNetB3, маючи при цьому найменшу кількість обчислювальних параметрів (12 млн проти 138 млн у VGG16).
3. **Покращення моделі.** Застосування стратегії глибокого донавчання (Fine-Tuning) із частковим розморожуванням високорівневих згорткових блоків та динамічним зменшенням швидкості покращення дозволило суттєво підвищити метрики якості. Фінальна модель EfficientNetB3 досягла загальної точності 92%, продемонструвавши при

цьому бездоганну чутливість (Recall = 100%) до епідеміологічно найнебезпечніших класів (COVID-19 та туберкульоз).

4. **Інтерпретованість результатів.** Інтеграція методології Explainable AI за допомогою алгоритму Grad-CAM дозволила успішно верифікувати логіку прийняття рішень моделлю. Клінічний аналіз згенерованих теплових карт підтвердив, що навчена нейронна мережа локалізує увагу виключно на реальних рентгенологічних ознаках (апікальні вогнища при туберкульозі, часткові інфільтрати при пневмонії), а не на артефактах зображень.

РОЗДІЛ 4. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ

4.1 Розробка серверної частини та API

4.1.1 Архітектура та структура серверної частини

Серверна частина програмного комплексу реалізована у вигляді самостійного Python-пакета, що розміщується у директорії backend/. Архітектура бекенду побудована за фундаментальним принципом розмежування відповідальності (Separation of Concerns). Це означає, що кожен модуль виконує чітко визначену функцію та не залежить від внутрішньої реалізації інших модулів, що забезпечує високу підтримуваність коду [68]. Дерево файлів серверної частини наведено на рисунку 4.1.

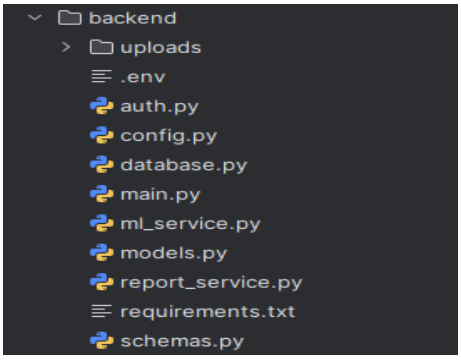

Рисунок 4.1 – Структура директорій та файлів серверної частини

Детальний опис призначення кожного модуля наведено у таблиці 4.1.

Таблиця 4.1 – Структура модулів серверної частини програмного комплексу

Файл/Директорія	Призначення модуля
main.py	Головний модуль FastAPI – визначення маршрутів (endpoints) та middleware
auth.py	Модуль автентифікації – хешування паролів та генерація JWT-токенів
models.py	ORM-моделі SQLAlchemy – декларативне визначення таблиць бази даних
schemas.py	Схеми Pydantic – валідація вхідних (Request) та вихідних (Response) даних
database.py	Управління сесіями та конфігурація підключення до СКБД PostgreSQL
config.py	Централізоване управління конфігурацією програмного комплексу
ml_service.py	ML-сервіс – завантаження ваг моделі та виведення результату
report_service.py	Сервіс генерації медичних звітів у форматах PDF та JSON
requirements.txt	Зафіксований перелік залежностей для відтворення середовища
.env	Змінні середовища (паролі, ключі доступу), що ігноруються системою Git

Конфігурація веб-застосунку винесена в окремий файл .env та завантажується через модуль config.py з використанням бібліотеки pydantic-settings. Це забезпечує повну відповідність методології «12-Factor App» [69] – конфігурація зберігається у середовищі розгортання, а не в коді, що є стандартом безпеки для виробничих комплексів.

4.1.2 Проєктування REST API та механізму авторизації

Програмний інтерфейс серверної частини реалізовано відповідно до архітектурного стилю REST (Representational State Transfer) з використанням сучасного високопродуктивного фреймворку FastAPI. Усі ендпоінти повертають відповіді у стандартизованому форматі JSON та використовують семантично правильні HTTP-коди стану. Специфікацію розробленого REST API наведено у таблиці 4.2.

Таблиця 4.2 – Специфікація REST API розробленого програмного комплексу

Логічна група	Метод	Шлях (Endpoint)	Опис функції	Потребує JWT
Авторизація	POST	/auth/register	Реєстрація нового лікаря в застосунку	Ні
	POST	/auth/login	Вхід до застосунку, генерація JWT-токена	Ні
	GET	/auth/me	Отримання даних поточного користувача	Так
Пацієнти	GET	/patients	Отримання списку пацієнтів лікаря	Так
	POST	/patients	Додавання нового пацієнта до бази	Так

Продовження таблиці 4.2

Діагностика	POST	/analyze	Аналіз рентгенограми нейромережею	Так
	GET	/history	Історія проведених аналізів лікаря	Так
	GET	/analysis/{id}	Детальна інформація про конкретний аналіз	Так
	GET	/analysis/{id}/pdf	Експорт результатів у форматі PDF	Так
	GET	/stats	Отримання статистичних даних лікаря	Так
Модель	GET	/model/info	Метадані ML моделі	Ні
Моніторинг	GET	/health	Перевірка стану (Health Check) сервісу	Ні

Фреймворк FastAPI забезпечує автоматичну генерацію інтерактивної документації у форматі OpenAPI (Swagger UI). Інтерфейс документації, що дозволяє тестувати API безпосередньо з браузера, наведено на рисунках 4.2 та 4.3.

Механізм автентифікації реалізовано на основі відкритого стандарту RFC 7519 – JSON Web Token (JWT) [70]. Обрано підхід без збереження стану (stateless), який не потребує зберігання сесій на сервері та природно масштабується.

Під час реєстрації пароль користувача необоротно хешується алгоритмом `bcrypt`. Під час входу сервер верифікує хеш та повертає криптографічно підписаний JWT-токен. Для доступу до захищених ендпоінтів клієнтський додаток передає токен у HTTP-заголовку Authorization: Bearer

<token>. FastAPI автоматично перевіряє валідність токена через механізм ін'єкції залежностей, зокрема функцію `Depends (get_current_user)`. Це гарантує, що лікар має доступ виключно до власних пацієнтів, унеможливлюючи несанкціонований доступ до конфіденційних медичних даних.

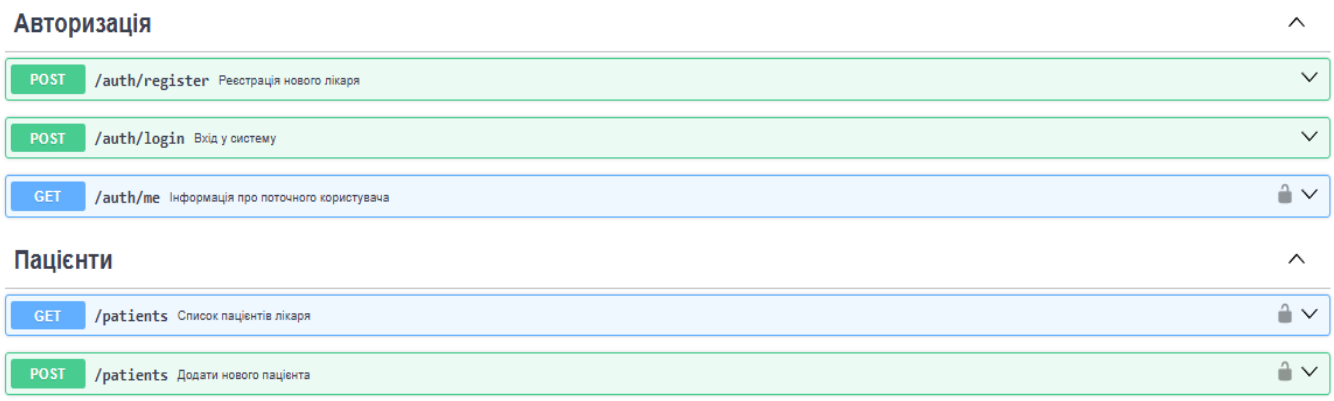

Рисунок 4.2 – Інтерактивна документація Swagger UI (групи «Авторизація» та «Пацієнти»)

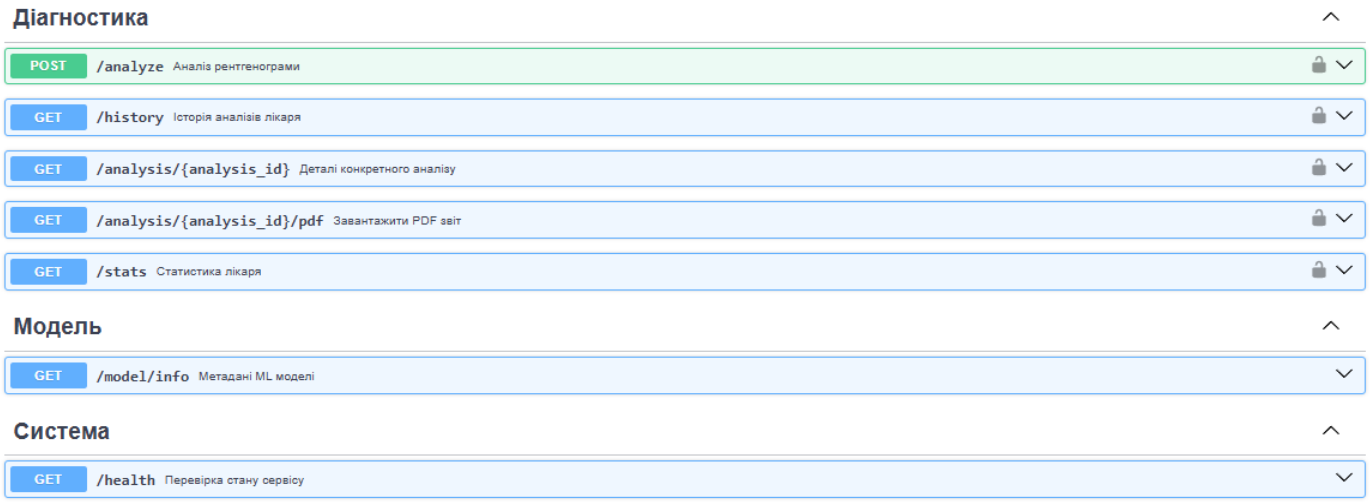

Рисунок 4.3 – Інтерактивна документація Swagger UI (групи «Діагностика», «Модель» та «Моніторинг»)

4.1.3 Інтеграція модуля машинного навчання та конвеєр обробки запитів

Модуль машинного навчання інкапсульовано у файлі ml_service.py у вигляді окремого сервісу. Фундаментальним архітектурним рішенням є використання патерну «Одинак» (Singleton) для управління життєвим циклом моделі. Оскільки завантаження файлу ваг EfficientNetB3 з накопичувача та ініціалізація тензорів у відеопам'яті займає певний час, ця операція виконується виключно один раз – під час запуску вебсервера. Це запобігає витокам пам'яті та критичним затримкам під час обробки клієнтських запитів [71].

Повний життєвий цикл обробки діагностичного запиту наведено у вигляді UML-діаграми послідовності на рисунку 4.4.

Під час отримання POST-запиту на маршрут /analyze, API-маршрутизатор спочатку здійснює сувору валідацію отриманих даних через схеми Pydantic. Перевіряється MIME-тип файлу (дозволено лише image/jpeg та image/png) та відповідні текстові метадані (ідентифікатор пацієнта тощо).

Після успішної валідації запускається комплексний багатоетапний конвеєр виведення результату:

1. **Збереження оригіналу:** Завантажений файл рентгенограми зберігається у локальне файлове сховище (директорія uploads/) з присвоєнням унікального ідентифікатора, згенерованого за алгоритмом uuid.uuid4(). Це унеможливлює колізії імен при паралельному завантаженні файлів різними лікарями.
2. **Аналіз нейромережею:** Байти зображення передаються до сервісу ml_service.py, де виконується конвертація у PyTorch-тензор розмірності [3, 224, 224], нормалізація та пряме поширення (Forward Pass) через архітектуру EfficientNetB3 на базі графічного прискорювача (CUDA).

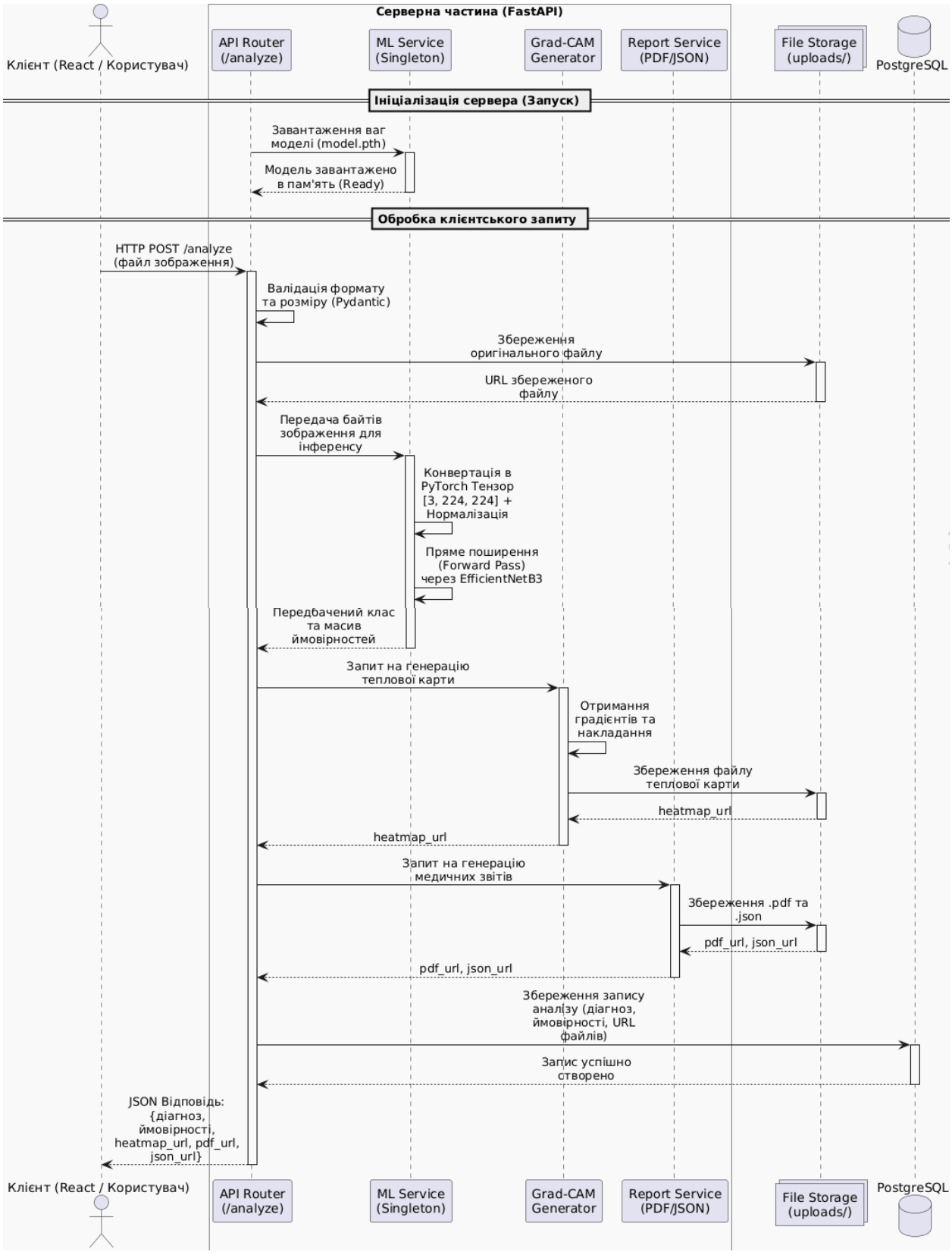

Рисунок 4.4 – UML-діаграма послідовності конвеєра обробки діагностичного запиту

3. **Генерація візуальних пояснень:** Після отримання кінцевого діагнозу та масиву ймовірностей від класифікаційної голови, генератор Grad-CAM обчислює градієнти цільового класу, формує візуальну теплову карту та зберігає її як окремий файл у сховищі, повертаючи серверній частині відповідне посилання (heatmap_url).
4. **Формування медичних звітів:** Сервіс генерації документів (report_service.py) створює формалізований медичний висновок у форматі PDF (для подальшого друку) та машинно-зчитуваний файл JSON (для експорту), які також розміщуються у файловому сховищі з отриманням посилань pdf_url та json_url.
5. **Фіксація результатів у базі даних:** Усі отримані результати (прогнозований діагноз, масив імовірностей по кожному класу, ідентифікатор пацієнта та URL-посилання на всі чотири згенеровані файли) зберігаються як єдина транзакція у реляційній базі даних PostgreSQL.

Завершальним етапом є повернення клієнтському додатку JSON-відповіді, що містить повний набір збережених даних та посилань. Завдяки асинхронній природі фреймворку FastAPI та виконанню ресурсомістких математичних операцій на відеокарті, загальний час виконання цього розгалуженого конвеєра не перевищує 3 секунд на один запит, що гарантує високу інтерактивність діагностичного програмного комплексу.

4.1.4 Розгортання та налаштування бази даних

Фізичне розгортання реляційної бази даних програмного комплексу здійснено на базі СКБД PostgreSQL. Створення схеми бази даних повністю автоматизовано засобами ORM-фреймворку SQLAlchemy – під час першого запуску серверної частини виконується виклик методу `Base.metadata.create_all(bind=engine)`, що автоматично генерує DDL-команди та створює всі необхідні таблиці відповідно до визначених Python-моделей. Такий підхід усуває необхідність ручного написання SQL-

скриптів міграції та гарантує сувору відповідність між ORM-моделями та фізичною схемою бази даних [72].

Результат успішного розгортання схеми наведено на рисунку 4.5 – у дереві об'єктів утиліти адміністрування pgAdmin відображаються три таблиці, що точно відповідають спроєктованій логічній моделі: analyses, patients та users.

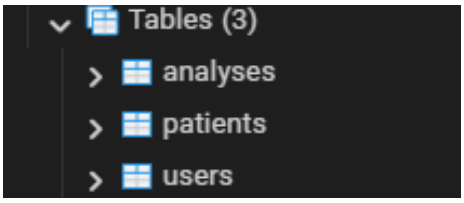

Рисунок 4.5 – Схема бази даних розробленого програмного комплексу у клієнтській утиліті

Для верифікації коректності збереження результатів діагностики на рисунку 4.6 наведено фрагмент реальних даних таблиці analyses, що містить перші вісім тестових записів.

	id [PK] integer	patient_id integer	doctor_id integer	image_path character vary	gradcam_path character varying	pdf_path character varying	json_path character varying	diagnosis character varying (50)	cor dou
1	1	1	1	C:\Users\...	C:\Users\user\Deskt...	C:\Users\user\De...	C:\Users\user\D...	COVID19	
2	2	2	1	C:\Users\...	C:\Users\user\Deskt...	C:\Users\user\De...	C:\Users\user\D...	NORMAL	0.9
3	3	3	1	C:\Users\...	C:\Users\user\Deskt...	C:\Users\user\De...	C:\Users\user\D...	PNEUMONIA	0.9
4	4	4	1	C:\Users\...	C:\Users\user\Deskt...	C:\Users\user\De...	C:\Users\user\D...	TUBERCULOSIS	0.'
5	5	5	1	C:\Users\...	C:\Users\user\Deskt...	C:\Users\user\De...	C:\Users\user\D...	NORMAL	0.9
6	6	6	1	C:\Users\...	C:\Users\user\Deskt...	C:\Users\user\De...	C:\Users\user\D...	NORMAL	0.9

Рисунок 4.6 – Фрагмент даних таблиці analyses після виконання тестових виведень результатів

Аналіз даних, наведених на рисунку 4.6, підтверджує коректну роботу всього діагностичного конвеєра серверної частини. Кожен запис містить зовнішні ключі patient_id та doctor_id, що забезпечують цілісний реляційний зв'язок з відповідними таблицями. Поля image_path, gradcam_path, pdf_path та json_path містять коректні абсолютні шляхи до згенерованих файлів у

локальному сховищі сервера. Поле diagnosis безпомилково фіксує один із чотирьох цільових класів – у наведеному фрагменті присутні прогнози для COVID-19, NORMAL, PNEUMONIA та TUBERCULOSIS, що свідчить про успішну мультикласову класифікацію, виконаною нейромережею в умовах реального тестування.

Підключення серверної частини до PostgreSQL налаштовано через рядок підключення, що зберігається у захищеній змінній середовища `DATABASE_URL` файлу .env:

```
DATABASE_URL=postgresql://username:password@localhost:5432/lungai
```

Управління сесіями бази даних реалізовано через патерн контекстного менеджера з використанням функції-генератора `get_db()`. Завдяки цьому кожен HTTP-запит отримує власну ізольовану сесію, що автоматично закривається після завершення обробки, незалежно від того, чи була операція успішною, чи викликала виняток. Це гарантує відсутність витоків з'єднань (Connection Leaks) при паралельній обробці масиву запитів від кількох лікарів одночасно [72].

4.2 Створення інтерактивного клієнтського інтерфейсу

4.2.1 Архітектура та структура React-застосунку

Клієнтська частина програмного комплексу реалізована у вигляді односторінкового застосунку (Single Page Application, SPA) на базі бібліотеки React 18 з маршрутизацією на стороні клієнта через пакет React Router. Застосунок включає кілька логічних представлень, навігація між якими здійснюється без повного перезавантаження сторінки, що забезпечує високу інтерактивність та швидкість відклику інтерфейсу [73]. Структуру директорій клієнтської частини наведено на рисунку 4.7.

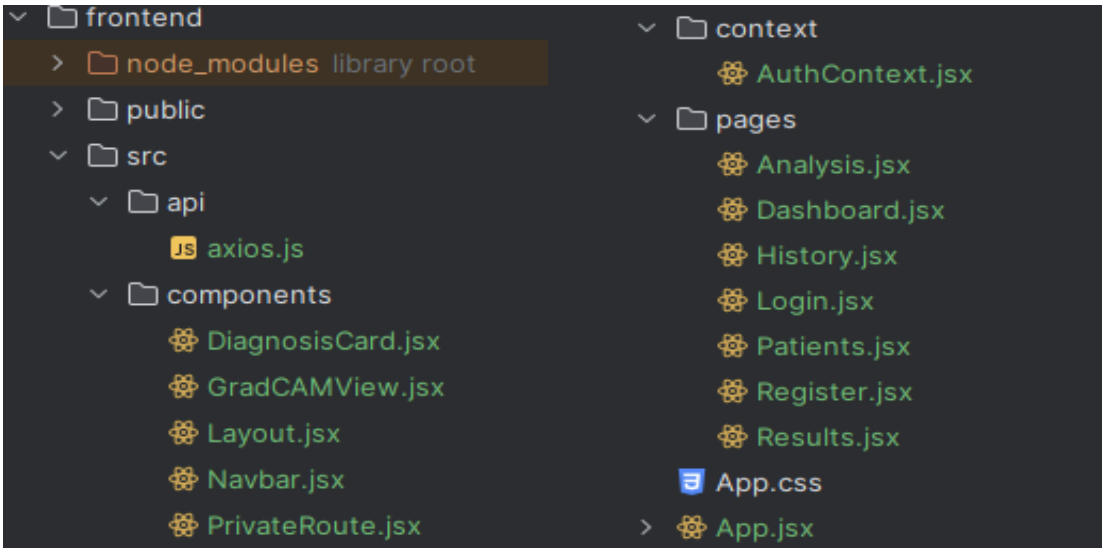

Рисунок 4.7 – Структура директорій та файлів клієнтської частини

Вихідний код організовано за принципом розмежування відповідальності – кожна директорія відповідає за окремий аспект роботи застосунку. Детальний опис структури наведено у таблиці 4.3.

Таблиця 4.3 – Структура модулів клієнтської частини веб-застосунку

Файл / Директорія	Призначення модуля
api/axios.js	Налаштований екземпляр Axios з базовим URL та JWT-перехоплювачем
components/DiagnosisCard.jsx	Компонент відображення діагнозу та горизонтальних смуг імовірностей
components/GradCAMView.jsx	Компонент перемикача режимів відображення «Оригінал / Grad-CAM»
components/Layout.jsx	Базовий макет (Layout) сторінки з навігаційною панеллю
components/Navbar.jsx	Бічна навігаційна панель з меню та даними поточного лікаря

Продовження таблиці 4.3

components/PrivateRoute.jsx	Компонент захисту маршрутів (перенаправлення неавторизованих)
context/AuthContext.jsx	Глобальний стан авторизації через механізм React Context API
pages/Login.jsx	Сторінка входу до застостосунку (авторизація)
pages/Register.jsx	Сторінка реєстрації нового облікового запису лікаря
pages/Dashboard.jsx	Головний робочий простір – форма завантаження рентгенограми
pages/Results.jsx	Сторінка візуалізації результатів після завершення аналізу
pages/Analysis.jsx	Детальний перегляд раніше збереженого аналізу з історії
pages/History.jsx	Хронологічний список усіх проведених аналізів лікаря
pages/Patients.jsx	Модуль управління списком зареєстрованих пацієнтів
App.jsx	Кореневий компонент застосунку із визначенням графа маршрутів
App.css	Глобальні CSS-змінні та базові стилі (темізація)

Усі HTTP-запити до серверної частини (FastAPI) централізовано через єдиний налаштований екземпляр бібліотеки Axios (api/axios.js). Це дозволяє визначити базовий URL сервера в одному місці та автоматично додавати JWT-токен до заголовка кожного запиту за допомогою механізму перехоплювачів

(interceptors), що усуває необхідність ручної передачі токена у кожному компоненті [73].

4.2.2 Механізм автентифікації у клієнтському застосунку

Управління станом авторизації реалізовано через React Context API – вбудований механізм, що дозволяє передавати дані між компонентами без необхідності каскадної передачі властивостей (prop drilling) через проміжні рівні ієрархії [74]. Модуль AuthContext.jsx зберігає дані поточного лікаря у стані React та дублює їх у локальне сховище (localStorage) браузера, що забезпечує збереження активної сесії між перезавантаженнями сторінки.

Захист приватних маршрутів реалізовано за допомогою компонента-обгортки PrivateRoute.jsx, що перевіряє наявність авторизованого користувача у глобальному контексті. За відсутності авторизації відбувається автоматичне перенаправлення на сторінку входу за допомогою компонента Maps бібліотеки React Router. Інтерфейс відповідної сторінки наведено на рисунку 4.8.

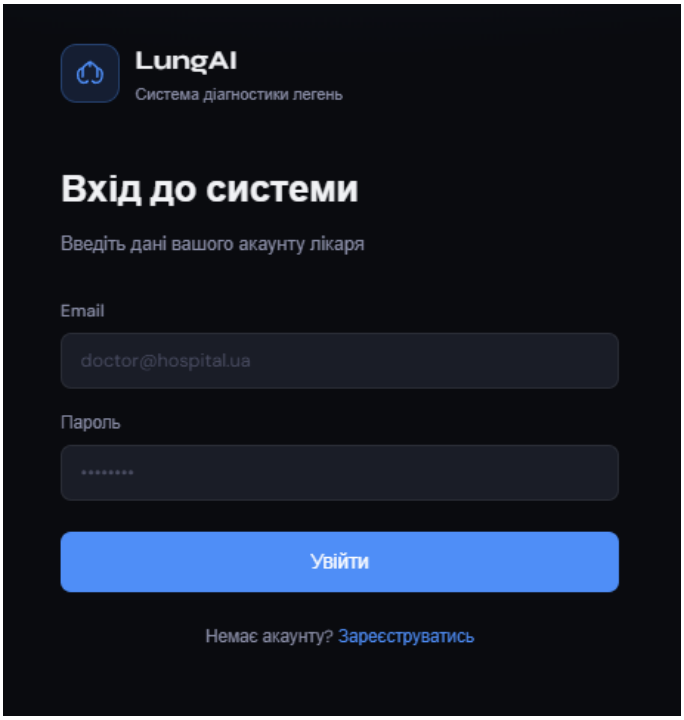

Рисунок 4.8 – Інтерфейс автентифікації лікаря у веб-застосунку

4.2.3 Головна сторінка діагностики (Dashboard)

Головна сторінка є центральним елементом користувацького інтерфейсу та реалізує основний сценарій використання веб-застосунку – завантаження рентгенограми для ШІ-аналізу. Загальний вигляд основного робочого простору наведено на рисунку 4.9. Його інтерфейс логічно розділено на дві функціональні зони.

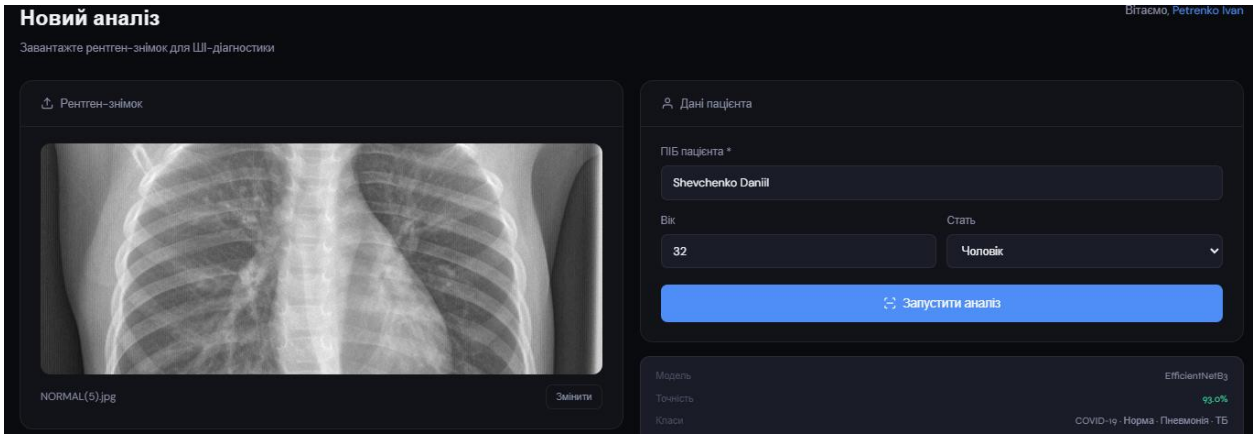

Рисунок 4.9 – Головний робочий простір веб-застосунку (Dashboard)

Ліва зона містить компонент завантаження файлу, що підтримує як стандартний вибір через діалогове вікно, так і механізм перетягування (Drag & Drop). Після вибору файлу відображається попередній перегляд рентгенограми безпосередньо у браузері без попереднього завантаження на сервер, що реалізовано засобами Web API (FileReader та URL.createObjectURL).

Права зона містить форму введення даних пацієнта (ПІБ, вік, стать). Нижче розташований інформаційний блок, що відображає поточні метадані завантаженої ML-моделі (архітектура EfficientNetB3, валідаційна точність 93.0% та перелік підтримуваних класів діагностики). Цей блок асинхронно завантажує дані з ендпоінта /model/info під час ініціалізації сторінки, що реалізує принцип прозорості ШІ (AI Accountability) – лікар завжди бачить, яка саме версія моделі виконує діагностику.

Під час натискання кнопки «Запустити аналіз» формується запит типу multipart/form-data до ендпоінта POST /analyze, де файл рентгенограми передається як бінарні дані, а метадані пацієнта – як query-параметри. Після отримання відповіді відбувається автоматичне перенаправлення на сторінку результатів.

4.2.4 Сторінка результатів аналізу

Сторінка результатів (рис. 4.10) відображається одразу після завершення аналізу та організована у вигляді двоколонкової сітки, що забезпечує оптимальне використання простору екрана.

Ліва колонка містить картку діагнозу з двома блоками. Верхній блок відображає кольоровий бейдж діагнозу (наприклад, червоний для COVID-19, фіолетовий для TUBERCULOSIS) та числовий показник впевненості моделі у відсотках. Нижній блок містить горизонтальні смуги імовірностей (Probability Bars) для всіх чотирьох класів, що дозволяє лікарю візуально оцінити розподіл впевненості нейромережі.

Рисунок 4.10 – Інтерфейс візуалізації результатів ШІ-аналізу

Права колонка містить інтерактивний переглядач знімків з перемикачем режимів. Режим «Оригінал» показує завантажену рентгенограму, а режим «Grad-CAM» відображає згенеровану теплову карту, накладену на оригінальний знімок, що візуалізує ділянки, на які орієнтувалася модель при прийнятті рішення. У верхній частині панелі розташовані елементи керування для експорту формалізованих медичних звітів у форматах PDF та JSON.

4.2.5 Сторінки перегляду історії та управління пацієнтами

Сторінка історії (рис. 4.11) відображає хронологічний список усіх проведених аналізів поточного лікаря, отриманий з ендпоінта GET /history. Кожен запис містить ім'я пацієнта, дату аналізу, діагноз та відсоток впевненості. При натисканні на запис відбувається перехід на сторінку детального перегляду (Analysis.jsx), що завантажує повні дані з бази, включаючи збережену Grad-CAM карту та посилання на звіти.

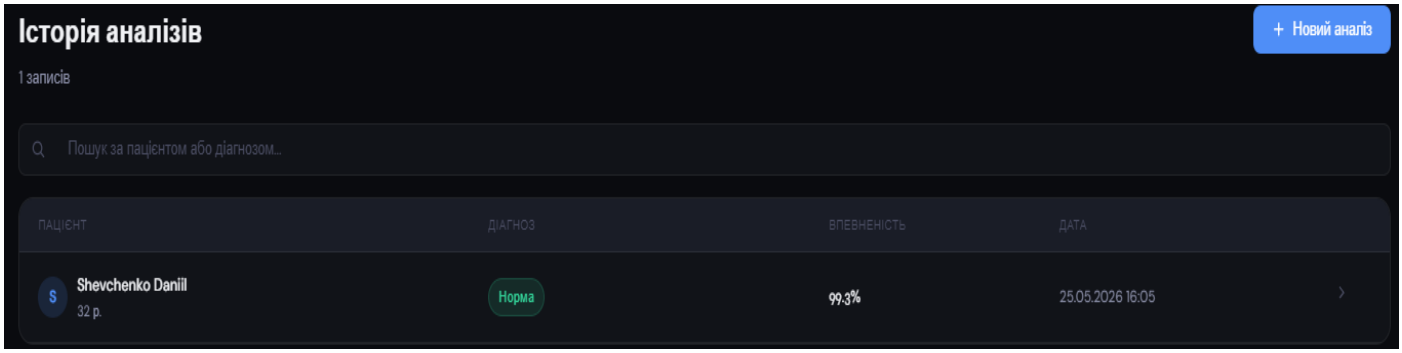

Рисунок 4.11 – Інтерфейс перегляду історії проведених досліджень

Сторінка управління пацієнтами (Patients.jsx) надає лікарю можливість переглядати список зареєстрованих осіб та додавати нових карток через форму, що відправляє запит до ендпоінта POST /patients. Дизайн усього інтерфейсу витримано у єдиному стилі темної теми (Dark Mode) з темно-синім фоном та акцентними кольорами, що відповідає сучасній концепції медичного програмного забезпечення, орієнтованого на мінімізацію зорової втоми лікаря під час тривалої роботи з радіологічними знімками [75].

4.3 Реалізація модуля експорту медичних звітів

Для успішної інтеграції розробленого модуля машинного навчання у реальну клінічну практику критично важливою є можливість експорту результатів у стандартизованих форматах. Сучасні медичні інформаційні системи (МІС) вимагають впровадження двох типів звітності:

1. **Людинозчитуваний формат (PDF):** Необхідний для документування діагнозу, додавання до фізичної медичної картки пацієнта та юридичного засвідчення результатів підписом лікаря-радіолога.
2. **Машиннозчитуваний формат (JSON):** Використовується для автоматизованого обміну даними між різними програмними рішеннями лікувального закладу (наприклад, для інтеграції з електронними медичними картками EHR або передачі в централізовані сховища інформації).

Генерація обох форматів реалізована на стороні серверної частини у модулі report_service.py. Даний сервіс асинхронно отримує вхідні дані від конвеєра інференсу після успішної генерації візуальних пояснень (Grad-CAM) та автоматично формує документи, зберігаючи їх у локальному сховищі сервера.

4.3.1 Структура та генерація PDF-звіту

Для автоматизованої генерації PDF-документів застосовано бібліотеку ReportLab – Python-інструмент, що надає низькорівневий програмний контроль над розташуванням елементів документа та підтримує вставку растрових зображень безпосередньо у тіло PDF. Такий підхід забезпечує високу швидкість формування файлів і точне позиціювання складних медичних графіків, таблиць та діагностичних знімків.

Згенерований медичний звіт (Medical Diagnostic Report) має чітку ієрархічну структуру та складається з кількох логічних блоків, які для підвищення читабельності в основному тексті роботи розділено на окремі фрагменти.

Блок ідентифікації та діагнозу: Містить унікальний ідентифікатор аналізу (Analysis ID), персональні дані пацієнта (ПІБ, вік) та інформацію про лікаря, що проводив діагностику, із зазначенням точної дати та часу (наприклад, 25.05.2026 13:05). У цьому ж блоці виводиться фінальний спрогнозований нейромережею діагноз (наприклад, NORMAL) та рівень загальної впевненості моделі у відсотках (наприклад, 99.3%). Фрагмент шапки звіту наведено на рисунку 4.12.

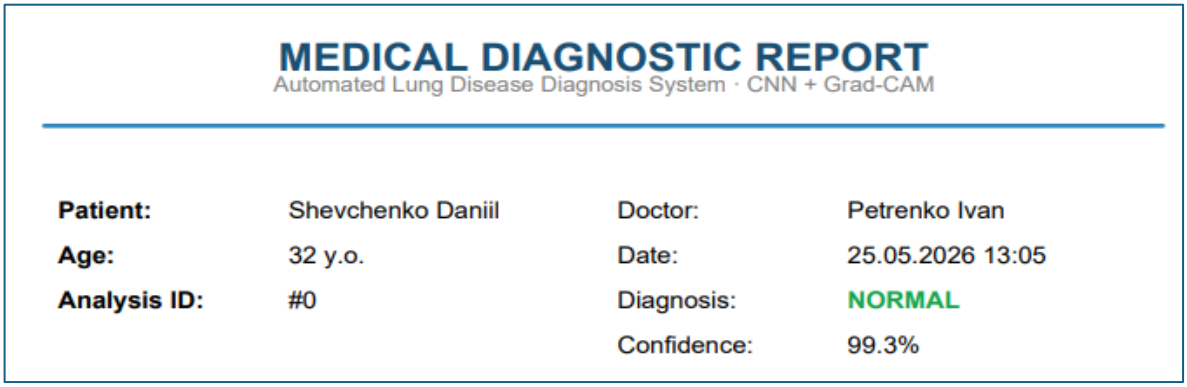

Рисунок 4.12 – Блок ідентифікації пацієнта та висновку у PDF-звіті

Блок візуалізації (X-Ray and Grad-CAM Visualization): Найважливіший компонент звіту для медичного персоналу. Він містить оригінальну рентгенограму пацієнта, яка розміщена пліч-о-пліч зі згенерованою тепловою картою (Grad-CAM Heatmap). Це забезпечує візуальну інтерпретованість результатів, дозволяючи лікарю зрозуміти, на які саме ділянки легень спиралася нейромережа під час аналізу (рисунок 4.13).

Аналітичний блок: Надає детальний розподіл ймовірностей (Probability Distribution by Class) у вигляді горизонтальної гістограми. На графіку наочно відображаються показники для всіх чотирьох цільових класів: передбаченого (наприклад, NORMAL – 99.3%) та інших (PNEUMONIA – 0.7%, COVID19 – 0.0%, TUBERCULOSIS – 0.0%). Цей фрагмент наведено на рисунку 4.14.

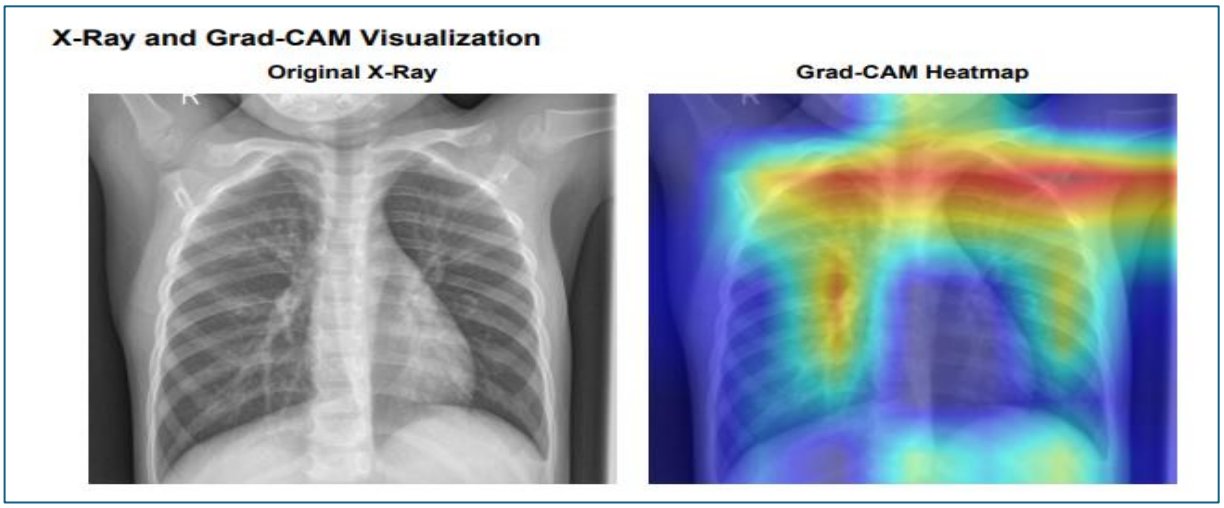

Рисунок 4.13 – Блок візуалізації оригінального знімка та теплової карти

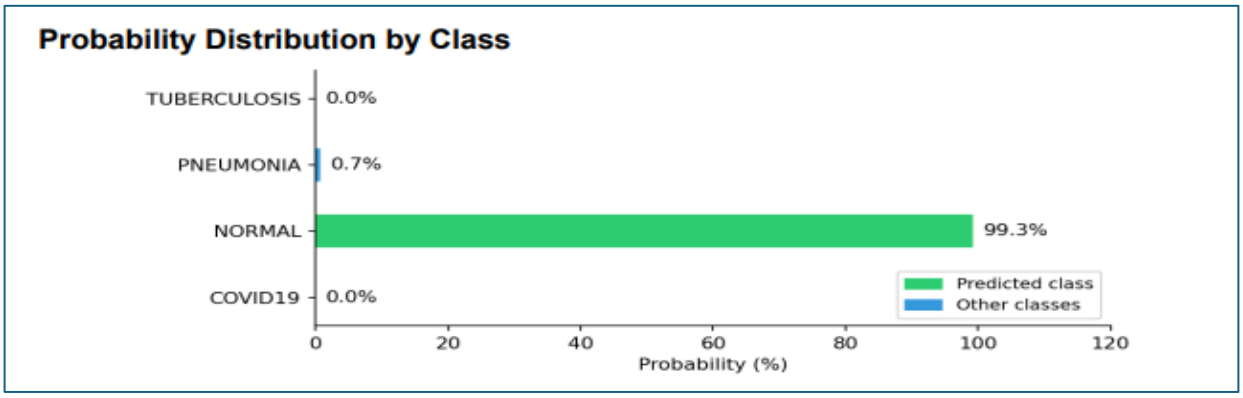

Рисунок 4.14 – Аналітичний блок розподілу ймовірностей за класами

Блок підзвітності ШІ (AI Accountability) та правове застереження: Містить метадані моделі, що виконувала інференс. Сюди входять назва архітектури (EfficientNetB3), версія програмного забезпечення (v1.0.0), використаний фреймворк (PyTorch 2.5.1), а також стратегія навчання (Transfer Learning + Fine-tuning), точність на валідаційній вибірці (93.0%) та відомості про тренувальний набір даних (COVID-19 + Chest X-Ray + TB Dataset).

Для дотримання юридичних норм розробки медичного програмного забезпечення, у нижній частині звіту (рисунок 4.15) розміщено обов'язкове попередження (Disclaimer), яке інформує, що звіт є виключно інструментом підтримки прийняття рішень (Decision Support Tool) і не замінює повноцінну консультацію з кваліфікованим медичним спеціалістом.

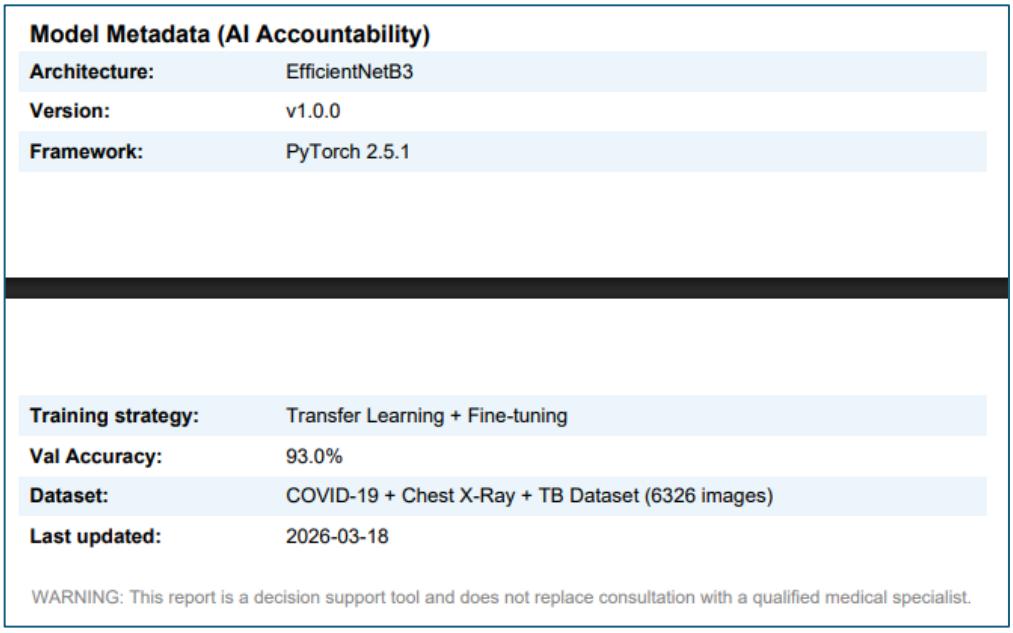

Рисунок 4.15 – Блок метаданих моделі та правове застереження

4.3.2 Формування машиннозчитуваного JSON-експорту

Для забезпечення програмної сумісності розробленого рішення із зовнішніми сервісами, паралельно з друкованою формою звіту сервером генерується машиннозчитуваний файл у форматі JSON. Структура об'єкта спроєктована таким чином, щоб мінімізувати надмірність даних, забезпечити строгу типізацію полів та гарантувати швидкий парсинг сторонніми медичними застосунками.

Повну фізичну структуру згенерованого експортного файлу результатів діагностики наведено у додатку Б.

Згенерований об'єкт містить повний інформаційний зріз на момент проведення дослідження: часову мітку у міжнародному стандарті ISO 8601 (created_at), блоки ідентифікації суб'єктів процесу (пацієнта та лікаря), а також масив точних числових результатів інференсу (result).

Важливою архітектурною особливістю є обов'язкова інкапсуляція метаданих моделі (model) у кожен звіт. Це гарантує, що навіть у разі подальшого донавчання або повної заміни архітектури нейромережі на сервері в майбутньому, лікарняна база даних назавжди збереже точну інформацію про

те, за допомогою якої саме версії алгоритму, на основі якого набору даних та з якими показниками точності було сформовано цей історичний медичний висновок.

4.4 Тестування програмного комплексу

4.4.1 Стратегія та інструментарій тестування

Для підтвердження надійності, безпеки та продуктивності розробленого програмного комплексу було застосовано комплексну стратегію тестування. Враховуючи клієнт-серверну архітектуру медичного комплексу, процес верифікації було розділено на два ключові етапи:

1. **Інтеграційне тестування API (Бекенд):** Здійснювалося за допомогою платформи Postman для перевірки коректності обробки HTTP-запитів, валідації вхідних даних та роботи механізм авторизації.
2. **Автоматизоване наскрізне тестування UI (Фронтенд):** Виконувалося за допомогою фреймворку Selenium WebDriver для симуляції дій реального користувача у браузері та перевірки реакції клієнтського інтерфейсу.

4.4.2 Інтеграційне тестування серверної частини (API) у Postman

Для перевірки серверної частини (FastAPI) у середовищі Postman було створено колекцію автоматизованих тестів, що покривають як позитивні, так і негативні сценарії використання. Особливу увагу було приділено перевірці механізмів безпеки (перевірка JWT-токенів) та стійкості конвеєра машинного навчання до некоректних форматів вхідних даних.

Перелік розроблених тестових сценаріїв (Test Cases) та результати їх виконання наведено у таблиці 4.4.

Таблиця 4.4 – Сценарії інтеграційного тестування API за допомогою Postman

№	Назва та метод запиту	Опис тестового сценарію та вхідні дані	Очікуваний HTTP-код	Фактичний результат
1	POST /auth/register	Реєстрація лікаря з коректними даними (JSON)	201 Created	Успішно. Користувача створено
2	POST /auth/login	Вхід з валідним email та паролем	200 OK	Успішно. Отримано JWT-токен
3	POST /auth/login	**(Негативний)** Вхід з невірним паролем	401 Unauthorized	Запит відхилено сервером
4	GET /history	**(Негативний)** Доступ до історії без токена	403 Forbidden / 401	Доступ заблоковано
5	GET /health	Перевірка доступності сервера	200 OK	Отримано {"status": "ok"}
6	POST /analyze	Завантаження рентгену формату PNG/JPEG + JWT	200 OK	Успішний інференс, повернено JSON
7	POST /analyze	**(Негативний)** Завантаження .txt файлу	422 Unprocessable	Блокування Pydantic-валідатором
8	GET /analysis/99	**(Негативний)** Запит неіснуючого аналізу	404 Not Found	Запис у базі не знайдено

Як видно з таблиці 4.4, API серверної частини демонструє абсолютну стійкість до некоректних запитів. Механізми Dependency Injection у FastAPI надійно захищають приватні маршрути від неавторизованого доступу.

Для візуального підтвердження працездатності основного ШІ-конвеєра, на рисунку 4.16 наведено скріншот виконання найважливішого тесту №6 (POST /analyze) у середовищі Postman.

На рисунку чітко видно, що сервер успішно прийняв файл через форму multipart/form-data, провів його через нейромережу, згенерував необхідні файли звітів та за 1.76 секунди повернув структуровану JSON-відповідь зі статусом 200 OK.

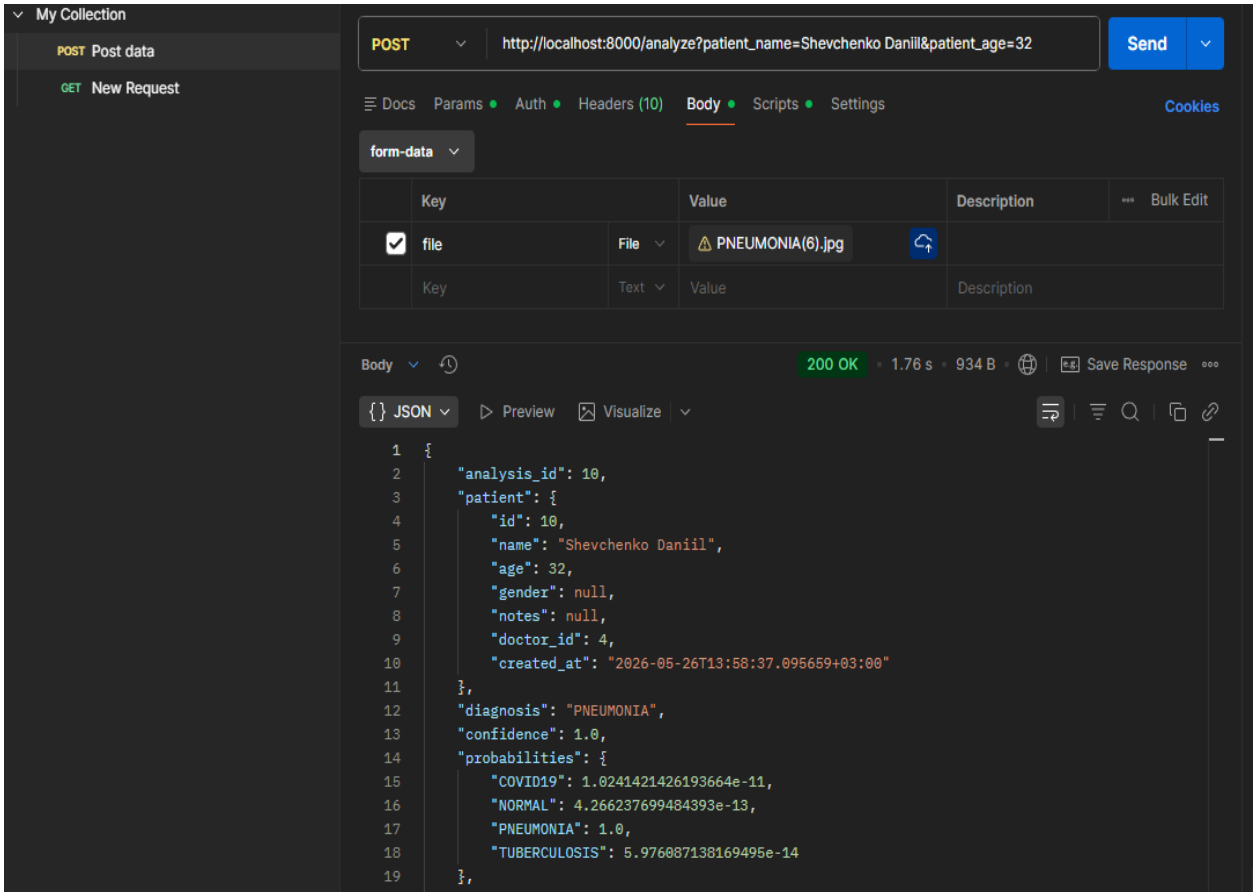

Рисунок 4.16 – Виконання запиту діагностики (/analyze) у середовищі Postman

4.4.3 Автоматизоване наскрізне тестування інтерфейсу (Selenium E2E)

Для забезпечення високої якості користувацького досвіду (UX) та верифікації клієнтської логіки React-застосунку було застосовано методологію автоматизованого наскрізного тестування (End-to-End, E2E). Замість ручної перевірки інтерфейсу було розроблено тестовий набір (Test Suite) мовою Python із використанням інструментарію Selenium WebDriver, який дозволяє програмно керувати екземпляром браузера та симулювати реальні дії лікаря у веб-застосунку.

Розроблений набір охоплює 6 ключових сценаріїв поведінки користувача:

1. Перевірка завантаження компонента авторизації.
2. Валідація відхилення запиту при введенні невірних облікових даних.
3. Успішна авторизація з автоматичним перенаправленням на головний екран.
4. Перевірка наявності елементів керування (форми завантаження рентгенограми) на робочому просторі (Dashboard).
5. Верифікація успішного завантаження компонента історії аналізів.
6. Перевірка механізму захисту маршрутів – блокування доступу до сторінки історії після очищення локального сховища.

Для демонстрації підходу до автоматизації розроблено набір тестових скриптів мовою Python. Повний програмний код автоматизованого E2E-тестування клієнтського інтерфейсу із застосуванням Selenium WebDriver наведено у додатку В.

Скрипт здійснює пошук необхідних DOM-елементів за допомогою CSS-селекторів, вводить облікові дані та ініціює події натискання. Критерієм успішності (Assertion) є перевірка стану поточного URL браузера після виконання дій.

Результат успішного виконання всього набору автоматизованих тестів у консолі середовища розробки наведено на рисунку 4.17.

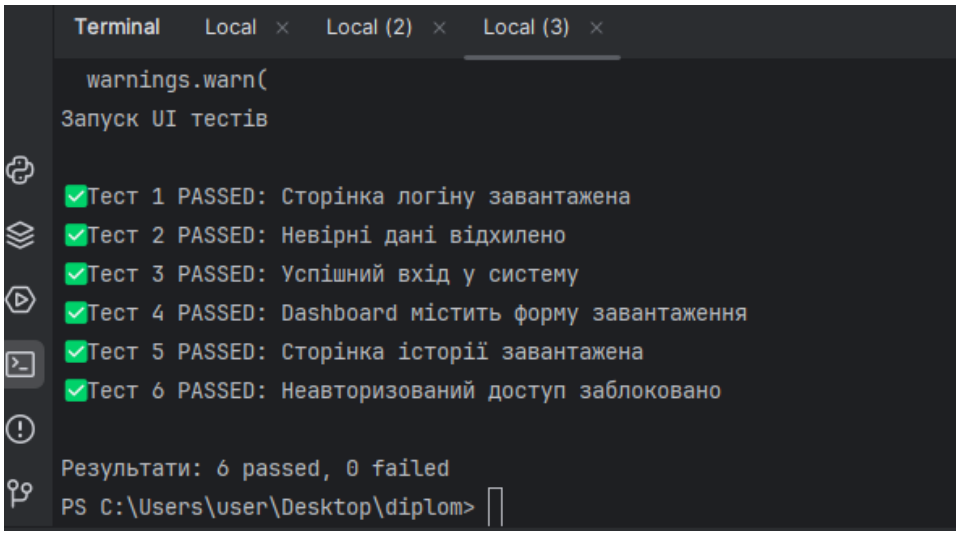

Рисунок 4.17 – Результати виконання E2E-тестів клієнтської частини веб-застосунку

Як видно з консольного виводу (рисунок 4.17), усі 6 розроблених тестів завершилися зі статусом PASSED. Це практично доводить, що клієнтська частина коректно взаємодіє з сервером, правильно обробляє JWT-токени, надійно захищає робочий простір від неавторизованого доступу та стабільно відображає ключові компоненти інтерфейсу.

Висновки до розділу 4

У четвертому розділі було виконано повну практичну реалізацію та комплексне тестування програмного продукту для інтелектуальної діагностики патологій легень. Проведені роботи дозволили перетворити навчену нейромережеву модель на повноцінний, готовий до впровадження медичний комплекс.

1. **Реалізовано серверну частину та базу даних:** Розроблено високопродуктивний REST API на базі асинхронного фреймворку FastAPI. Інтеграцію модуля машинного навчання (EfficientNetB3) виконано за архітектурним патерном «Одинак» (Singleton), що повністю усунуло затримки на ініціалізацію моделі. Для надійного збереження

даних розгорнуто реляційну базу даних PostgreSQL з автоматичним управлінням міграціями через ORM SQLAlchemy.

2. **Створено інтерактивний клієнтський інтерфейс (React):** Розроблено Single Page Application на базі бібліотеки React 18 із використанням захищеної маршрутизації та глобального управління станом. Реалізовано ергономічний робочий простір лікаря-радіолога, що дозволяє завантажувати рентгенограми, переглядати розподіл імовірностей діагнозу та інтерактивно взаємодіяти з тепловими картами Grad-CAM.
3. **Розроблено модуль генерації медичних звітів:** Впроваджено функціонал експорту результатів інференсу. За допомогою бібліотеки ReportLab реалізовано програмне формування юридично значущих PDF-документів із розміщенням оригінальних знімків, теплових карт та метаданих алгоритму. Для забезпечення програмної сумісності з іншими медичними системами (MIC) реалізовано генерацію машиннозчитуваних JSON-файлів.
4. **Проведено комплексне тестування та оцінку швидкодії:** Інтеграційне тестування серверної частини у середовищі Postman підтвердило абсолютну стійкість API до некоректних вхідних даних (зокрема, недопустимих форматів файлів). Наскрізне E2E-тестування клієнтського інтерфейсу за допомогою скриптів Selenium WebDriver довело надійність механізм авторизації та маршрутизації. Оцінка часових характеристик засвідчила, що загальний час обробки одного діагностичного запиту (включно з прямим поширенням через мережу, генерацією Grad-CAM та записом у БД) становить 1.76 секунди, що гарантує високу інтерактивність розробленого програмного комплексу в реальному клінічному процесі.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальне науково-практичне завдання – створено програмний комплекс інтелектуального аналізу рентгенограм для первинного скринінгу патологій легень, що функціонує як надійний інструмент підтримки прийняття медичних рішень.

У ході виконання досліджень та програмної реалізації було отримано такі основні результати:

1. **Проведено комплексний аналіз предметної області.** Доведено доцільність автоматизації процесу розпізнавання легеневих захворювань (пневмонії, туберкульозу та COVID-19) за допомогою методів глибокого навчання. Обґрунтовано використання стратегії Transfer Learning для подолання проблеми обмеженості розмічених медичних даних та забезпечення високої здатності моделі до узагальнення.
2. **Спроєктовано архітектуру програмного комплексу.** Розроблено багаторівневу клієнт-серверну архітектуру, що забезпечує масштабованість та ізоляцію процесів обчислення ШІ від інтерфейсу користувача. За допомогою засобів UML-моделювання (Use-case, Activity діаграми) та ER-діаграм побудовано логічну та фізичну моделі реляційної бази даних, оптимізовані для зберігання результатів інференсу, масивів імовірностей та мультимедійних файлів.
3. **Розроблено та навчено нейромережеву модель.** На базі згорткової архітектури EfficientNetB3 реалізовано алгоритм мультикласової класифікації патологій. Завдяки стратегії поступового розморожування шарів (Fine-Tuning) досягнуто загальної точності на валідаційній вибірці на рівні 93.0%. Ключовим здобутком є інтеграція модуля Explainable AI (алгоритм Grad-CAM), яка генерує теплові карти для візуального пояснення рішень моделі, що є критично важливою вимогою для медичного ПЗ.

4. **Виконано програмну реалізацію серверної та клієнтської частин.** Бекенд веб-застосунку реалізовано мовою Python на базі асинхронного REST-фреймворку FastAPI. Клієнтський інтерфейс розроблено у вигляді Single Page Application за допомогою бібліотеки React 18, що забезпечило високу ергономічність робочого простору лікаря-радіолога. Налаштовано автоматичне розгортання схеми бази даних під управлінням СКБД PostgreSQL із застосуванням ORM SQLAlchemy.
5. **Розроблено модуль генерації звітів.** Для забезпечення юридичної значущості та програмної інтероперабельності результатів діагностики реалізовано модуль експорту. За допомогою інструментарію ReportLab налаштовано автоматичну програмну генерацію PDF-звітів, що містять оригінальні знімки, теплові карти та метадані моделі, а також паралельне формування машинозчитуваних JSON-файлів.
6. **Здійснено комплексне тестування розробленого програмного комплексу.** Інтеграційне тестування API (через Postman) та автоматизоване E2E-тестування клієнтського інтерфейсу (через скрипти Selenium WebDriver) підтвердили високу надійність механізму авторизації, захисту маршрутів та стійкість до некоректних вхідних даних. Хронометраж швидкодії довів, що загальний час обробки діагностичного запиту (включно з інференсом, Grad-CAM та записом у БД) становить лише 1.76 секунди, що гарантує високу інтерактивність розробленого програмного комплексу в реальних клінічних умовах.

Практична цінність одержаних результатів полягає у створенні повністю готового до впровадження програмного продукту. Розроблений медичний комплекс здатний мінімізувати вплив людського фактору, зменшити зорове навантаження на лікарів-радіологів та значно пришвидшити процес первинного скринінгу та пріоритезації обстежень у медичних закладах

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. World Health Organization (WHO). URL: https://www.who.int (дата звернення: 05.05.2026).

2. Doi K. Computer-aided diagnosis in medical imaging: historical review, current status and future potential. *Computerized Medical Imaging and Graphics*. 2007. Vol. 31, no. 4-5. P. 198–211.

3. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385*. 2015. URL: https://arxiv.org/abs/1512.03385 (дата звернення: 05.05.2026).

4. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*. 2014. URL: https://arxiv.org/abs/1409.1556 (дата звернення: 06.05.2026)..

5. Tan M., Le Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv preprint arXiv:1905.11946*. 2019. URL: https://arxiv.org/abs/1905.11946 (дата звернення: 06.05.2026)..

6. Selvaraju R. R. et al. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *arXiv preprint arXiv:1610.02391*. 2016. URL: https://arxiv.org/abs/1610.02391 (дата звернення: 06.05.2026)..

7. Brady A. P. Error and discrepancy in radiology: inevitable or avoidable? *Insights into Imaging*. 2017. Vol. 8, no. 1. P. 171–182. URL: https://doi.org/10.1007/s13244-016-0534-1 (дата звернення: 07.05.2026)..

8. Bruno M. A., Walker E. A., Abujudeh H. H. Understanding and Confronting Our Mistakes: The Epidemiology of Error in Radiology and Strategies for Error Reduction. *RadioGraphics*. 2015. Vol. 35, no. 6. P. 1668–1676. URL: https://doi.org/10.1148/rg.2015150023 (дата звернення: 07.05.2026)..

9. Krupinski E. A., Berbaum K. S., Caldwell R. T. et al. Do long radiology workdays affect nodule detection in dynamic CT interpretation? *Journal of the*

American College of Radiology. 2012. Vol. 9, no. 3. P. 191–198. URL: https://doi.org/10.1016/j.jacr.2011.11.013 (дата звернення: 10.05.2026)..

10. Communicating Radiation Risks in Paediatric Imaging. Geneva : World Health Organization, 2020. 84 p.

11. Fenton J. J., Taplin S. H., Carney P. A. et al. Influence of Computer-Aided Detection on Performance of Screening Mammography. *New England Journal of Medicine*. 2007. Vol. 356, no. 14. P. 1399–1409.

12. Al-Sarayreh M. H. A. et al. An End-to-End Multimodal Automated Diagnosis System for Multiple Sclerosis. *Diagnostics*. 2024. Vol. 14, no. 19. P. 2197. URL: https://doi.org/10.3390/diagnostics14192197 (дата звернення: 11.05.2026)..

13. Liang G. et al. Deep Learning for Chest X-ray Diagnosis: Competition Between Radiologists with or Without Artificial Intelligence Assistance. *Journal of Imaging Informatics in Medicine*. 2024. URL: https://doi.org/10.1007/s10278-024-00990-6 (дата звернення: 11.05.2026)..

14. Rajpurkar P., Irvin J., Zhu K. et al. CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. *arXiv preprint arXiv:1711.05225*. 2017. URL: https://arxiv.org/abs/1711.05225 (дата звернення: 12.05.2026).

15. Wang L., Lin Z. Q., Wong A. COVID-Net: a tailored deep convolutional neural network design for detection of COVID-19 cases from chest X-ray images. *Scientific Reports*. 2020. Vol. 10, no. 1. P. 1–12.

16. Melendez J. et al. Diagnostic accuracy of three CAD systems for detecting pulmonary tuberculosis. *medRxiv*. 2023. URL: https://doi.org/10.1101/2022.03.30.22273191 (дата звернення: 12.05.2026).

17. Chattopadhay A., Sarkar A., Howlader P., Balasubramanian V. N. Grad-CAM++: Generalized gradient-based visual explanations for deep convolutional networks. *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018. P. 839–847. URL: https://arxiv.org/abs/1710.11063 (дата звернення: 12.05.2026).

18. Jiang P. T., Zhang C. B., Hou Q., Cheng M. M., Wei Y. LayerCAM: Exploring Hierarchical Class Activation Maps for Localization. *IEEE Transactions on Image Processing*. 2021. Vol. 30. P. 5875–5888. URL: https://ieeexplore.ieee.org/document/9440698 (дата звернення: 14.05.2026).

19. Lerma M. A., Lucas M. Grad-CAM++ is Equivalent to Grad-CAM with Positive Gradients. *arXiv preprint arXiv:2205.10838*. 2022. URL: https://arxiv.org/abs/2205.10838 (дата звернення: 15.05.2026).

20. Grad-CAM for CNNs. *XAI Tutorials*. URL: https://xai-tutorials.readthedocs.io/en/latest/_model_specific_xai/Grad-CAM.html (дата звернення: 15.05.2026).

21. Nielsen J. Response Times: The 3 Important Limits. *Nielsen Norman Group*. 1993. URL: https://www.nngroup.com/articles/response-times-3-important-limits/ (дата звернення: 15.05.2026).

22. Bass L., Clements P., Kazman R. Software Architecture in Practice. 3rd ed. Addison-Wesley Professional, 2012. 624 p.

23. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 560 p.

24. Kleppmann M. Designing Data-Intensive Applications. O'Reilly Media, 2017. 616 p.

25. Coronel C., Morris S. Database Systems: Design, Implementation, and Management. 13th ed. Cengage Learning, 2018. 864 p.

26. SQLAlchemy Documentation. URL: https://docs.sqlalchemy.org (дата звернення: 15.05.2026).

27. Codd E.F. Further Normalization of the Data Base Relational Model. IBM Research Report RJ909. 1971. P. 33–64.

28. Password Storage Cheat Sheet. OWASP Foundation. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення: 16.05.2026).

29. Developer Survey 2024. Stack Overflow. URL: https://survey.stackoverflow.co/2024 (дата звернення: 16.05.2026).

30. Banks A., Porcello E. Learning React. 2nd ed. O'Reilly Media, 2020. 350 p.

31. Macrae C. Vue.js: Up and Running. O'Reilly Media, 2023. 250 p.

32. FastAPI Documentation. URL: https://fastapi.tiangolo.com (дата звернення: 17.05.2026).

33. Grinberg M. Flask Web Development. 2nd ed. O'Reilly Media, 2018. 316 p.

34. Stevens E., Antiga L., Viehmann T. Deep Learning with PyTorch. Manning Publications, 2020. 504 p.

35. Momjian B. PostgreSQL: Introduction and Concepts. Addison-Wesley, 2001. 512 p.

36. Doshi-Velez F., Kim B. Towards a Rigorous Science of Interpretable Machine Learning. *arXiv preprint arXiv:1702.08608*, 2017. 13 p.

37. Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). *Official Journal of the European Union*, 2024.

38. ISO 32000-1:2008. Document management — Portable document format — Part 1: PDF 1.7. Geneva : International Organization for Standardization, 2008.

39. FHIR R4 — Fast Healthcare Interoperability Resources. HL7 International. URL: https://hl7.org/fhir (дата звернення: 17.05.2026).

40. Larson D.B. et al. Performance of a Deep-Learning Neural Network Model in Assessing Skeletal Maturity on Pediatric Hand Radiographs. *Radiology*, 2021. Vol. 287(1). P. 313–322.

41. Kaggle Dataset: Chest X-Ray: Pneumonia, COVID-19, Tuberculosis. URL: https://www.kaggle.com/datasets/jtiptj/chest-xray-pneumoniacovid19tuberculosis (дата звернення: 19.05.2026).

42. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. 800 p.

43. González G. et al. Disease staging and prognosis in smokers using deep learning in chest computed tomography. *American Journal of Respiratory and Critical Care Medicine*, 2018. Vol. 197(2). P. 193–203.

44. Johnson J.M., Khoshgoftaar T.M. Survey on deep learning with class imbalance. *Journal of Big Data*, 2019. Vol. 6(1). P. 1–54.

45. Tan M., Le Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *ICML. arXiv preprint arXiv:1905.11946*, 2019. 10 p.

46. Shorten C., Khoshgoftaar T.M. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 2019. Vol. 6(1). P. 1–48.

47. Pan S.J., Yang Q. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 2010. Vol. 22(10). P. 1345–1359.

48. Raghu M. et al. Transfusion: Understanding Transfer Learning for Medical Imaging. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. Vol. 32. P. 3347–3357.

49. Yosinski J. et al. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. Vol. 27. P. 3320–3328.

50. Powers D.M.W. Evaluation: From Precision, Recall and F-Factor to ROC. *Journal of Machine Learning Technologies*, 2011. Vol. 2(1). P. 37–63.

51. Obermeyer Z., Emanuel E.J. Predicting the Future — Big Data, Machine Learning, and Clinical Medicine. *The New England Journal of Medicine*, 2016. Vol. 375(13). P. 1216–1219.

52. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014. 14 p.

53. The Annotated VGG-16. *Medium*. URL: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918 (дата звернення: 20.05.2026).

54. He K. et al. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. P. 770–778.

55. The Annotated ResNet-50. *Medium*. URL: https://medium.com/data-science/the-annotated-resnet-50-a6c536034758 (дата звернення: 20.05.2026).

56. Tan M., Le Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *Proceedings of the 36th International Conference on Machine Learning (ICML)*. 2019. P. 6105–6114.

57. Complete Architectural Details of all EfficientNet Models. *Medium*. URL: https://medium.com/data-science/complete-architectural-details-of-all-efficientnet-models-5fd5b736142 (дата звернення: 20.05.2026).

58. Yosinski J. et al. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. Vol. 27. P. 3320–3328.

59. Fine-tuning with Keras and Deep Learning. *PyImageSearch*, 2019. URL: https://pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/ (дата звернення: 21.05.2026).

60. Howard J., Ruder S. Universal Language Model Fine-tuning for Text Classification. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2018. arXiv preprint arXiv:1801.06146.

61. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. 800 p.

62. Rajpurkar P. et al. CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. *arXiv preprint arXiv:1711.05225*, 2017. 7 p.

63. Wang L., Lin Z.Q., Wong A. COVID-Net: a tailored deep convolutional neural network design for detection of COVID-19 cases from chest X-ray images. *Scientific Reports*, 2020. Vol. 10(1). P. 19549.

64. Yosinski J. et al. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. Vol. 27. P. 3320–3328.

65. Zech J.R. et al. Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study. *PLOS Medicine*, 2018. Vol. 15(11). e1002683.

66. Rubin G.D. et al. The Role of Chest Imaging in Patient Management during the COVID-19 Pandemic: A Multinational Consensus Statement from the Fleischner Society. *Chest*, 2020. Vol. 158(1). P. 106–116.

67. Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). *Official Journal of the European Union*, 2024.

68. Martin R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.

69. Wiggins A. The Twelve-Factor App. URL: https://12factor.net/ (дата звернення: 23.05.2026).

70. Jones M. et al. JSON Web Token (JWT). *RFC 7519*. Internet Engineering Task Force (IETF), 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519

71. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. 395 p.

72. Bayer M. SQLAlchemy Documentation: Object Relational Tutorial. 2024. URL: https://docs.sqlalchemy.org/ (дата звернення: 25.05.2026).

73. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. O'Reilly Media, 2020. 338 p.

74. Dodds K.C. React Hooks in Action. Manning Publications, 2021. 360 p.

75. Staggers N., Rodney W.M. Nurse-computer interaction: Staff performance with a hospital information system. *Nursing Research*, 1993. Vol. 42(2). P. 80–85.

ДОДАТОК А. Програмний код модуля машинного навчання

Лістинг А.1 – Реалізація класу GradCAM засобами PyTorch

```
import torch
import cv2
import numpy as np

class GradCAM:
    """
    Реалізація методу Grad-CAM для інтерпретації рішень CNN.
    Використовує механізм hooks PyTorch для вилучення
    карт ознак та градієнтів без модифікації архітектури моделі.
    """
    def __init__(self, model):
        self.model = model
        self.gradients = None
        self.activations = None

        # Підключення hooks до останнього згорткового шару
        target_layer = model.features[-1]

target_layer.register_forward_hook(self._save_activations)

target_layer.register_backward_hook(self._save_gradients)

    def _save_activations(self, module, input, output):
        """Forward hook — зберігає проміжні карти ознак."""
        self.activations = output.detach()

    def _save_gradients(self, module, grad_input, grad_output):
        """Backward hook — зберігає протікаючі градієнти."""
        self.gradients = grad_output[0].detach()

    def generate(self, image_tensor, class_idx=None):
        """
        Генерує Grad-CAM теплову карту для вхідного зображення.
        Args:
            image_tensor: тензор зображення [C, H, W]
            class_idx: індекс цільового класу (None =
передбачений клас)

        Returns:
            cam: нормалізована теплова карта [H, W] в діапазоні
[0, 1]
            class_idx: індекс класу, для якого генерується карта
            probs: вектор імовірностей по всіх класах
        """
        device = next(self.model.parameters()).device
        image_tensor = image_tensor.unsqueeze(0).to(device)
```

```
        # Forward Pass
        output = self.model(image_tensor)

        if class_idx is None:
            class_idx = output.argmax(dim=1).item()

        # Backward Pass для цільового класу
        self.model.zero_grad()
        output[0, class_idx].backward()

        # Глобальне усереднення градієнтів по просторових
вимірах
        weights = self.gradients.mean(dim=[2, 3], keepdim=True)

        # Зважена сума карт ознак із застосуванням ReLU
        cam = (weights * self.activations).sum(dim=1,
keepdim=True)
        cam = torch.relu(cam)

        # Нормалізація та масштабування до розміру вхідного
зображення
        cam = cam.squeeze().cpu().numpy()
        cam = cv2.resize(cam, (224, 224))
        cam = (cam - cam.min()) / (cam.max() - cam.min() + 1e-8)

        return cam, class_idx,
output.softmax(dim=1)[0].detach().cpu().numpy()
```

ДОДАТОК Б. Приклад JSON-відповіді REST API медичного комплексу

Лістинг Б.1 — Приклад JSON-відповіді ендпоінту POST /analyze


```
{
  "analysis_id": 10,
  "created_at": "2026-05-26T13:58:37.095659+03:00",
  "patient": {
    "id": 10,
    "name": "Shevchenko Daniil",
    "age": 32,
    "gender": null,
    "notes": null,
    "doctor_id": 4,
    "created_at": "2026-05-26T13:58:37.095659+03:00"
  },
  "diagnosis": "PNEUMONIA",
  "confidence": 1.0,
  "probabilities": {
    "COVID19": 1.0241421426193664e-11,
    "NORMAL": 4.266237699484393e-13,
    "PNEUMONIA": 1.0,
    "TUBERCULOSIS": 5.976087138169495e-14
  },
  "model": {
    "architecture": "EfficientNetB3",
    "version": "v1.0.0",
    "val_accuracy": 0.93,
    "framework": "PyTorch 2.5.1",
    "training_strategy": "Transfer Learning + Fine-tuning",
    "dataset": "COVID-19 + Chest X-Ray + TB Dataset",
    "training_samples": 6326,
    "last_updated": "2026-03-18"
  }
}
```

ДОДАТОК В. Програмний код автоматизованого E2E тестування клієнтського інтерфейсу

Лістинг В.1 – Скрипт автоматизованого тестування UI засобами Selenium WebDriver

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.service import Service

BASE_URL = "http://localhost:3000"
EMAIL = "doc4@hosp.ua"
PASSWORD = "123456"


def get_driver():
    options = webdriver.ChromeOptions()
    options.add_argument("--window-size=1920,1080")
    service = Service(ChromeDriverManager().install())
    driver = webdriver.Chrome(service=service, options=options)
    return driver


def test_1_login_page_loads():
    """Тест 1: Сторінка логіну завантажується"""
    driver = get_driver()
    try:
        driver.get(f"{BASE_URL}/login")
        time.sleep(2)
        assert "login" in driver.current_url.lower()
        print(" Тест 1 PASSED: Сторінка логіну завантажена")
    finally:
        driver.quit()


def test_2_invalid_login():
    """Тест 2: Невірні дані — помилка авторизації"""
    driver = get_driver()
    try:
        driver.get(f"{BASE_URL}/login")
        time.sleep(2)

        email_field = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.CSS_SELECTOR,
"input[type='email']"))
```

```
        )
        email_field.send_keys("wrong@email.com")

        password_field = driver.find_element(By.CSS_SELECTOR,
"input[type='password']")
        password_field.send_keys("wrongpassword")

        button = driver.find_element(By.CSS_SELECTOR,
"button[type='submit']")
        button.click()
        time.sleep(2)

        assert "/login" in driver.current_url
        print(" Тест 2 PASSED: Невірні дані відхилено")
    finally:
        driver.quit()


def test_3_valid_login():
    """Тест 3: Коректні дані — успішний вхід"""
    driver = get_driver()
    try:
        driver.get(f"{BASE_URL}/login")
        time.sleep(2)

        email_field = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.CSS_SELECTOR,
"input[type='email']"))
        )
        email_field.send_keys(EMAIL)

        password_field = driver.find_element(By.CSS_SELECTOR,
"input[type='password']")
        password_field.send_keys(PASSWORD)

        button = driver.find_element(By.CSS_SELECTOR,
"button[type='submit']")
        button.click()
        time.sleep(3)

        assert driver.current_url == f"{BASE_URL}/"
        print(" Тест 3 PASSED: Успішний вхід у систему")
    finally:
        driver.quit()


def test_4_dashboard_elements():
    """Тест 4: Dashboard містить форму завантаження"""
    driver = get_driver()
    try:
        # Логін
        driver.get(f"{BASE_URL}/login")
```

```
        time.sleep(2)
        driver.find_element(By.CSS_SELECTOR,
"input[type='email']").send_keys(EMAIL)
        driver.find_element(By.CSS_SELECTOR,
"input[type='password']").send_keys(PASSWORD)
        driver.find_element(By.CSS_SELECTOR,
"button[type='submit']").click()
        time.sleep(3)

        # Перевірка Dashboard
        assert driver.current_url == f"{BASE_URL}/"
        upload_area = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.CSS_SELECTOR,
"input[type='file']"))
        )
        assert upload_area is not None
        print(" Тест 4 PASSED: Dashboard містить форму
завантаження")
    finally:
        driver.quit()


def test_5_history_page():
    """Тест 5: Сторінка історії завантажується"""
    driver = get_driver()
    try:
        # Логін
        driver.get(f"{BASE_URL}/login")
        time.sleep(2)
        driver.find_element(By.CSS_SELECTOR,
"input[type='email']").send_keys(EMAIL)
        driver.find_element(By.CSS_SELECTOR,
"input[type='password']").send_keys(PASSWORD)
        driver.find_element(By.CSS_SELECTOR,
"button[type='submit']").click()
        time.sleep(3)

        # Перехід на історію
        driver.get(f"{BASE_URL}/history")
        time.sleep(2)

        assert "/history" in driver.current_url
        print(" Тест 5 PASSED: Сторінка історії завантажена")
    finally:
        driver.quit()


def test_6_protected_route():
    """Тест 6: Захищений маршрут перенаправляє на логін"""
    driver = get_driver()
    try:
        # Очищаємо localStorage
```

```
        driver.get(f"{BASE_URL}/login")
        driver.execute_script("localStorage.clear()")
        time.sleep(1)

        # Спроба зайти на захищену сторінку
        driver.get(f"{BASE_URL}/history")
        time.sleep(2)

        assert "/login" in driver.current_url
        print(" Тест 6 PASSED: Неавторизований доступ
заблоковано")
    finally:
        driver.quit()


if __name__ == "__main__":
    print("Запуск UI тестів\n")

    tests = [
        test_1_login_page_loads,
        test_2_invalid_login,
        test_3_valid_login,
        test_4_dashboard_elements,
        test_5_history_page,
        test_6_protected_route,
    ]

    passed = 0
    failed = 0

    for test in tests:
        try:
            test()
            passed += 1
        except Exception as e:
            print(f" {test.__name__} FAILED: {e}")
            failed += 1

    print(f"\nРезультати: {passed} passed, {failed} failed")
```