МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

В. о. завідувача кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 20_25_ р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

СИСТЕМА ІНТЕРПРЕТАЦІЇ МЕДИЧНИХ ДІАГНОСТИЧНИХ МОДЕЛЕЙ НА ОСНОВІ МЕТОДІВ ХАІ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Інтелектуальні інформаційні системи»

Здобувач ____Сергій СТИПАНЕНКО

«___»_____ 2025 р.

Керівник канд. техн. наук, доцент ___Галина КОНДРАТЕНКО

«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Другий (магістерський)
Освітній ступень	Магістр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Інтелектуальні інформаційні системи

ЗАТВЕРДЖУЮ

В. о. завідувача кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

_____Стипаненка Сергія Валентиновича_____
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: _«Система інтерпретації медичних діагностичних моделей на основі методів XAI».

Керівник роботи: _Кондратенко Галина Володимирівна, доцент кафедри, канд. техн. наук, доцент.

Затверджена наказом ЧНУ ім. Петра Могили від «_07_» _липня_ 2025 р. № 184.

2. Строк представлення кваліфікаційної роботи «_16_» _грудня_ 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: програмний комплекс підтримки прийняття лікарських рішень, що включає модулі для аналізу табличних даних та медичних зображень, реалізує методи пояснюваного штучного інтелекту (SHAP, LIME, Grad–CAM) та має веб інтерфейс для візуалізації діагнозів і пояснень. Початкові дані: набори даних Breast Cancer Wisconsin (Diagnostic) Data Set (569 записів) та Chest X–Ray Images (Pneumonia)

(5863 зображення), бібліотеки машинного навчання (Scikit–learn, PyTorch) та засоби розробки мовою Python.

4. Перелік питань, що підлягають розробці: аналіз сучасного стану використання ШІ в медицині, проблеми «чорної скриньки» та нормативних вимог (EU AI Act) до прозорості алгоритмів; дослідження теоретико–математичних основ методів машинного навчання (ансамблеві методи, згорткові нейронні мережі) та алгоритмів інтерпретації (SHAP, Grad–CAM); обґрунтування вибору технологічного стеку та проектування архітектури мультимодальної діагностичної системи; програмна реалізація підсистем попередньої обробки даних, навчання моделей та генерації візуальних пояснень; експериментальне дослідження ефективності розробленої системи; аналіз метрик точності та валідація якості пояснень на клінічних прикладах.

5. Перелік графічних матеріалів: таблиці, рисунки, презентація.

Керівник роботи _____ Галина КОНДРАТЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Сергій СТИПАНЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «28» червня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Система інтерпретації медичних діагностичних моделей на основі методів XAI

	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	27.06.2025	30.06.2025	
2	Аналіз предметної області та постановка задачі	1.07.2025	20.07.2025	
3	Огляд літературних джерел та аналогічних систем медичної діагностики з використанням методів XAI	21.07.2025	30.07.2025	
4	Обґрунтування вибору методів інтерпретації (SHAP, LIME, Grad-CAM) та архітектур нейронних мереж для обробки мультимодальних даних	01.09.2025	25.10.2025	
5	Реалізація обраних технологій з аналізом отриманих результатів	26.10.2025	21.11.2025	
6	Перший попередній захист КР на засіданні комісії кафедри	25.11.2025	26.11.2025	
7	Корегування роботи за результатами попереднього захисту	27.11.2025	05.12.2025	
8	Другий попередній захист КР на засіданні комісії кафедри	09.12.2025	09.12.2025	
9	Доробка та остаточне оформлення КР	10.12.2025	12.12.2025	
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.12.2025	16.12.2025	

Керівник роботи _____ Галина КОНДРАТЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Сергій СТИПАНЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«08» липня 2025 р.

АНОТАЦІЯ

кваліфікаційної роботи студента групи 601 ЧНУ ім. Петра Могили

Стипаненка Сергія Валентиновича

Тема: «Система інтерпретації медичних діагностичних моделей на основі методів XAI»

Актуальність дослідження зумовлена необхідністю подолання проблеми «чорного ящика» у сучасних системах медичної діагностики. Широке впровадження алгоритмів глибокого навчання стримується відсутністю довіри з боку лікарів та жорсткими регуляторними вимогами щодо прозорості та пояснюваності рішень штучного інтелекту.

Метою роботи є підвищення ефективності автоматизованої медичної діагностики шляхом розробки інформаційної системи інтерпретації рішень моделей машинного навчання, що базується на гібридному використанні методів XAI для мультимодальних даних.

Об'єктом роботи є процеси інтерпретації медичних даних медичної діагностики на основі машинного навчання для аналізу табличних клінічних даних та рентгенографічних зображень.

Предметом роботи є методи пояснюваного штучного інтелекту для інтерпретації медичних діагностичних моделей, їх програмна реалізація засобами Python, архітектурні рішення для інтеграції XAI–модулів у веборієнтовану діагностичну платформу, та метрики оцінки якості пояснень.

Пояснювальна записка складається зі вступу, чотирьох розділів та висновків. У першому розділі проведено аналіз проблематики медичного ШІ, регуляторних вимог та існуючих методів пояснення. Другий розділ присвячено проєктуванню архітектури системи (XAI Orchestrator) та обґрунтуванню вибору алгоритмів. У третьому розділі описано програмну реалізацію системи засобами Python, PyTorch та Streamlit. Четвертий розділ містить результати експериментальних досліджень на наборах даних Breast Cancer Wisconsin та Chest X–Ray Images, а також аналіз клінічних кейсів.

Кваліфікаційна робота містить 99 сторінки, 33 ілюстрації, 9 таблиць, 39 літературних джерел.

Ключові слова: пояснюваний штучний інтелект (XAI), медична діагностика, SHAP, LIME, Grad–CAM, глибоке навчання, візуалізація даних.

ABSTRACT

of the qualification work of the student of group 601 of the Petro Mohyla Black Sea National University

Stypanenko Serhii

Topic: «System for Interpretation of Medical Diagnostic Models Based on XAI Methods»

The relevance of the study is driven by the need to overcome the “black box” problem in modern medical diagnostic systems. The widespread implementation of deep learning algorithms is hindered by a lack of trust from clinicians and strict regulatory requirements regarding the transparency and explainability of artificial intelligence decisions.

The aim of the study is to improve the efficiency of automated medical diagnostics by developing an information system for interpreting machine learning model decisions, based on the hybrid use of XAI methods for multimodal data.

The object of the study is the processes of interpreting medical diagnostic data based on machine learning for the analysis of tabular clinical data and radiographic images.

The subject of the study covers Explainable Artificial Intelligence (XAI) methods for interpreting medical diagnostic models, their software implementation using Python, architectural solutions for integrating XAI modules into a web–oriented diagnostic platform, and metrics for evaluating the quality of explanations.

The explanatory note consists of an introduction, four chapters, and conclusions. The first chapter analyzes the issues of medical AI, regulatory requirements, and existing explanation methods. The second chapter focuses on designing the system architecture (XAI Orchestrator) and justifying the selection of algorithms. The third chapter describes the software implementation of the system using Python, PyTorch, and Streamlit. The fourth chapter contains the results of experimental studies on the Breast Cancer Wisconsin and Chest X–Ray Images datasets, as well as an analysis of clinical cases.

The qualification work contains 99 pages, 33 illustrations, 9 tables, 39 literature sources.

Keywords: Explainable AI (XAI), medical diagnostics, SHAP, LIME, Grad–CAM, deep learning, data visualization.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – Application Programming Interface
AUC – Area Under the Curve
CDSS – Clinical Decision Support Systems
CNN – Convolutional Neural Network
DICOM – Digital Imaging and Communications in Medicine
DL – Deep Learning
EU AI Act – European Union Artificial Intelligence Act
FDA – Food and Drug Administration
GDPR – General Data Protection Regulation
Grad-CAM – Gradient-weighted Class Activation Mapping
HER – Electronic Health Records
LIME – Local Interpretable Model-agnostic Explanations
ML – Machine Learning
MLMD – Machine Learning-enabled Medical Devices
NLP – Natural Language Processing
PACS – Picture Archiving and Communication System
ReLU – Rectified Linear Unit
ROC – Receiver Operating Characteristic
SHAP – SHapley Additive exPlanations
SOTA – State-of-the-Art
UML – Unified Modeling Language
XAI – Explainable Artificial Intelligence

ВСТУП

Впровадження систем штучного інтелекту (ШІ) у медичну діагностику, включаючи області радіології, онкології та патології, відкриває значні можливості для підвищення точності та оперативності клінічних рішень.1 Сучасні моделі глибокого навчання (DL) демонструють високу продуктивність, проте ця ефективність досягається за рахунок значної архітектурної складності, що перетворює ці системи на непрозорі «чорні ящики».1 Відсутність прозорості у високоризикових доменах є неприйнятною, оскільки вона породжує серйозні етичні та практичні проблеми, включаючи питання відповідальності, управління потенційною алгоритмічною упередженістю (bias) та підриває необхідну довіру між клініцистами, пацієнтами та розробниками.

Слід зазначити, що зростання обчислювальних потужностей та доступність великих наборів біомедичних даних призвели до зміни парадигми в діагностиці: перехід від доказової медицини (Evidence–Based Medicine) до медицини, керованої даними (Data–Driven Medicine). Проте, цей перехід супроводжується кризою довіри. Згідно з опитуваннями, понад 60% лікарів не готові використовувати рекомендації ШІ, якщо вони не розуміють логіку формування прогнозу. Це створює бар'єр так званої «останньої милі» (Last Mile Problem) – ситуації, коли технологічно досконала модель не впроваджується в клінічну практику через людський фактор. Таким чином, розробка системи XAI є не просто технічним завданням, а необхідною умовою для подолання бар'єру між "in silico" (комп'ютерним моделюванням) та "in vivo" (реальним лікуванням пацієнтів).

Проблема «чорного ящика» полягає в тому, що система генерує вихідні дані без розкриття своїх внутрішніх механізмів або логіки, що унеможливлює розуміння того, як і чому було прийнято конкретне діагностичне рішення.3 Це є однією з головних перешкод на шляху широкого впровадження ШІ у клінічну практику. Як наслідок, виникла потреба у створенні систем пояснюваного штучного інтелекту (XAI), які можуть генерувати пояснення, що є зрозумілими

для людини, надійними, і, що особливо важливо, відповідають вимогам регуляторних органів.

Метою роботи є підвищення ефективності автоматизованої медичної діагностики шляхом розробки інформаційної системи інтерпретації рішень моделей машинного навчання, що базується на гібридному використанні методів XAI для мультимодальних даних.

Об’єктом роботи є процеси інтерпретації медичних даних медичної діагностики на основі машинного навчання для аналізу табличних клінічних даних та рентгенографічних зображень.

Предметом роботи є методи пояснюваного штучного інтелекту для інтерпретації медичних діагностичних моделей, їх програмна реалізація засобами Python, архітектурні рішення для інтеграції XAI–модулів у веборієнтовану діагностичну платформу, та метрики оцінки якості пояснень.

Для досягнення поставленої мети визначено низку завдань, спрямованих на створення всебічної основи для магістерської роботи. Ці завдання включають детальний аналіз проблеми та регуляторного ландшафту, обґрунтування архітектурного рішення, порівняльний аналіз XAI–алгоритмів та визначення метрик для кількісної оцінки якості пояснень.

1 АНАЛІЗ ПРОБЛЕМАТИКИ МЕДИЧНОГО ШІ, ВИМОГИ ПРОЗОРОСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Еволюція систем підтримки прийняття рішень та роль ШІ в сучасній медицині

Історія використання комп'ютерних систем у медицині бере свій початок з середини 20–го століття. Перші спроби створити системи підтримки прийняття клінічних рішень (Clinical Decision Support Systems, CDSS) базувалися на експертних системах. Це були системи, засновані на правилах (Rule–Based Systems), наприклад, MYCIN, розроблена у 1970–х роках для діагностики інфекційних захворювань крові. Логіка таких систем була повністю прозорою: "ЯКЩО симптом А ТА аналіз Б, ТО діагноз В". Проте, їхня здатність до масштабування та навчання була обмеженою – всі правила доводилося прописувати вручну, що було неможливо для складних, багатофакторних захворювань.

З появою методів машинного навчання (Machine Learning), зокрема Support Vector Machines (SVM) та Random Forest у 1990–2000–х роках, точність діагностики значно зросла, але почала знижуватися пряма інтерпретованість. Ця тенденція досягла піку з початком ери глибокого навчання (Deep Learning) після 2012 року. Сучасні згорткові нейронні мережі (CNN), такі як ResNet або EfficientNet, оперують мільйонами параметрів, створюючи нелінійні відображення у багатовимірних просторах ознак, які людський мозок не здатен осягнути інтуїтивно. Еволюцію можна розділити на три етапи, які наведені нижче.

Ера "мілкого" навчання (Shallow Learning): використання методів, таких як логістична регресія та метод опорних векторів (SVM). Ці моделі були відносно простими та інтерпретованими, але вимагали складної ручної інженерії ознак (Feature Engineering).

Ера ансамблевих методів: поява випадкових лісів (Random Forest) та градієнтного бустингу (XGBoost, LightGBM). Ці методи значно підвищили

точність на табличних даних, але втратили пряму інтерпретованість, перетворившись на "сірі ящики".

З 2012 року (прорив AlexNet) згорткові нейронні мережі (CNN) революціонізували аналіз зображень. Вони дозволили автоматично виділяти ознаки з "сирих" пікселів, досягаючи надлюдської точності. Однак, ці моделі, що містять мільйони параметрів, стали повними "чорними ящиками".

Медична спільнота отримала інструменти з надлюдською точністю, але без можливості пояснення причинно–наслідкових зв'язків. Це призвело до формування нової наукової дисципліни – Explainable AI (XAI), яка ставить за мету не спрощення моделей (що призвело б до втрати точності), а створення прошарку інтерпретації, який перекладає "мову" векторів та тензорів на мову клінічних ознак.

Сьогодні ШІ застосовується на всіх етапах медичної допомоги: від раннього скринінгу та діагностики до планування персоналізованого лікування та прогнозування виживаності. Проте, парадокс полягає в тому, що чим потужнішою стає модель, тим менше ми розуміємо принципи її роботи.

1.2 Концептуалізація проблеми «Чорного ящика» у медичних діагностичних моделях

Проблема непрозорості в медичному ШІ має глибокі етичні та практичні наслідки. Коли інженери або фахівці з даних, які створили алгоритм, не можуть пояснити, як саме він прийшов до певного результату, це створює істотний ризик. У медицині, де рішення впливають на життя та здоров'я пацієнтів, ця ситуація порушує основи медичної етики та підзвітності.

Одним з найбільш значущих викликів є проблема упередженості алгоритмів (bias). Якщо тренувальні дані містять дисбаланс, модель може мати упереджене ставлення до певних груп пацієнтів, що призведе до нерівних клінічних результатів. Завдяки XAI можна сканувати систему на предмет потенційної

упередженості, що дозволяє розробникам управляти справедливістю та мінімізувати ризик небажаних наслідків.

Крім того, виникає питання відповідальності (accountability). Якщо ШІ–система допускає помилку, але її логіка невідома, це ставить під сумнів, хто несе відповідальність: розробник, лікар, який застосував систему, чи сама система. Необхідність впровадження стандартизованих політик та заходів для запобігання ситуаціям, коли лікарі помилково несуть відповідальність за помилки ШІ, є імперативною. Прозорість і пояснюваність є важливими для підтримки довіри між лікарем і пацієнтом та для забезпечення того, що система працює належним чином.

Деякі академічні дискусії, що пропонують Computational Reliabilism, стверджують, що якщо алгоритм є надзвичайно надійним, повна внутрішня прозорість може бути не такою критичною, оскільки обчислювальні процеси за своєю суттю можуть бути непрозорими для людини. Однак, навіть у такому випадку, етичні проблеми, пов’язані з упередженістю та відповідальністю, залишаються. Це означає, що виправдане знання, отримане надійними індикаторами, є необхідним, але не достатнім для нормативного обґрунтування дій лікаря. Завжди потрібне обговорення результатів надійних алгоритмів для визначення найбільш бажаної дії. Таким чином, XAI виступає як інструмент, що забезпечує лікаря необхідною інформацією для прийняття обґрунтованого рішення та підтримання професійної відповідальності.

1.3 Регуляторні та етичні вимоги до систем XAI у сфері охорони здоров’я (EU AI Act, FDA)

Регуляторний ландшафт ШІ у медицині швидко еволюціонує, встановлюючи жорсткі вимоги до прозорості, особливо для систем, які класифікуються як високоризикові.

Вимоги EU AI Act наведено нижче.

Закон ЄС про Штучний Інтелект (EU AI Act), який регулює використання ШІ в ЄС, застосовує підхід, заснований на ризиках. ШІ–системи, які призначені для використання як компоненти безпеки медичних пристроїв (регульовані MDR/IVDR) або є самими пристроями, автоматично відносяться до категорії високого ризику. Це вимагає найвищого рівня регуляторного контролю.

Високий ризик означає, що ці системи повинні відповідати суворим вимогам, включаючи ведення чітких записів про продуктивність, моніторинг побічних ефектів, використання відповідно до встановлених протоколів безпеки, а також обов'язковий людський нагляд (human oversight). Впроваджувачі ШІ–систем (наприклад, лікарні) зобов'язані забезпечити, що рекомендації ШІ перевіряються та документуються клініцистами, які мають втручатися, якщо рекомендації ШІ не відповідають потребам пацієнта. Крім того, Акт має екстериторіальне охоплення, що означає, що він застосовується до провайдерів (розробників), незалежно від їхнього місцезнаходження, якщо їхній продукт використовується на ринку ЄС. Цей регуляторний тиск робить пояснюваність обов'язковою умовою доступу на ринок.

США, Канада та Велика Британія співпрацюють для встановлення керівних принципів, які підкреслюють важливість прозорості у медичних пристроях з підтримкою машинного навчання (MLMDs).10 Ці принципи зосереджуються на команді «Людина–ШІ» та наданні користувачам чіткої та важливої інформації про пристрій.

FDA визначає прозорість як ступінь, до якого інформація про MLMD (його намір, розробка, продуктивність і логіка) комунікується відповідній аудиторії. Пояснюваність (Explainability) є аспектом прозорості, що описує ступінь, до якого логіка, що призвела до результату, може бути зрозуміла людині. Регуляторні органи вимагають вбудовування прозорості та пояснюваності у регуляторні рамки для підвищення підзвітності та надання запевнень медичним працівникам та пацієнтам.

Вимога надання користувачам зрозумілої інформації про логіку рішення визначає, що кінцевий дизайн XAI–системи повинен бути людино–орієнтованим (human–centered design). Це означає, що технічно правильне пояснення, згенероване алгоритмом (наприклад, набір коефіцієнтів SHAP), має бути перетворено на клінічно значущий, візуалізований та легко інтерпретований формат. Ця вимога слугує ключовим мостом між технічною розробкою XAI–алгоритмів та їхнім практичним впровадженням у клінічний робочий процес (див. рис. 1.1).

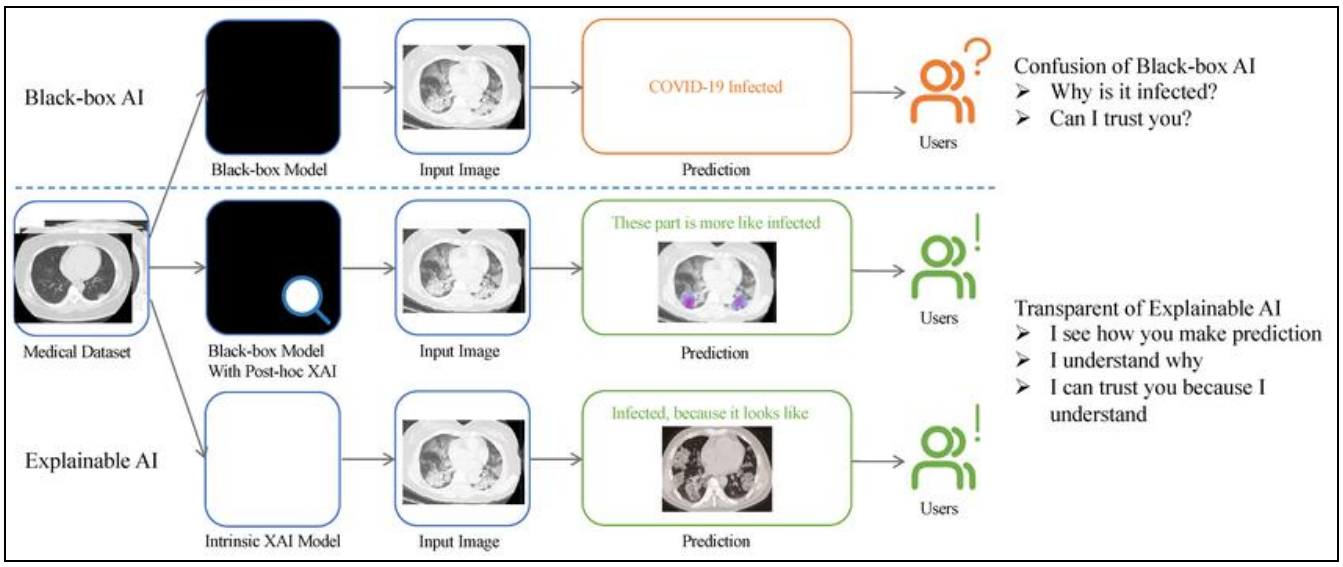

Рисунок 1.1 – Візуалізація проблеми «чорного ящика» та необхідність XAI у клінічному середовищі

Особливу увагу слід приділити статті 13 (Article 13) Акту про ШІ, яка вимагає «прозорості та надання інформації користувачам». Згідно з цим положенням, системи високого ризику повинні проєктуватися таким чином, щоб їхня робота була достатньо прозорою для інтерпретації вихідних даних користувачем. Це вимагає від розробників не лише технічної документації, а й вбудованих механізмів пояснення в реальному часі ("explainability by design").

Крім того, Загальний регламент захисту даних (GDPR) у статтях 13–15 та особливо 22 ("Автоматизоване прийняття індивідуальних рішень") закріплює "право на пояснення" (Right to Explanation). Це означає, що пацієнт має юридичне право знати, чому алгоритм відніс його до групи ризику, що робить XAI не

опціональною фічею, а обов'язковою вимогою compliance (відповідності стандартам).

Окрім етичних проблем упередженості, критичним аспектом безпеки є вразливість глибоких нейронних мереж до так званих адверсарних атак. Це цілеспрямовані маніпуляції з вхідними даними, які непомітні для людського ока, але змушують модель робити грубі помилки.

У контексті медичної візуалізації (наприклад, дерматоскопії або рентгенографії) зловмисник (або навіть випадковий шум обладнання) може додати спеціально згенерований "шум" до зображення.

Дослідження Finlayson et al. (2019) показали, що медичні алгоритми особливо вразливі до таких атак. Наприклад, зміна кількох пікселів на знімку очного дна може змусити систему змінити діагноз з "здоровий" на "діабетична ретинопатія". Це створює ризики для страхового шахрайства (штучне "погіршення" діагнозу для отримання виплат) або кібертероризму. Саме тому системи XAI (зокрема Grad–CAM) виступають не лише інструментом пояснення, а й лінією оборони: якщо атака змінює діагноз, вона часто змінює і карту уваги мережі, що може бути помічено лікарем–експертом.

1.4 Огляд та класифікація методів XAI

XAI-методи класифікуються за тим, як вони генерують пояснення, що дозволяє вибирати оптимальні підходи для різних типів медичних даних.

Для медичного застосування, де точність є високою ставкою, перевага часто надається найскладнішим, високопродуктивним моделям (Deep Learning). Через Парадокс продуктивності–інтерпретованості ці моделі є непрозорими, але їхня висока точність є критичною. Відповідно, найбільш прагматичним рішенням є застосування пост–хок методів (методів, що застосовуються після навчання моделі), таких як LIME, SHAP та Grad–CAM, для пояснення їхніх рішень. Це дозволяє використовувати найточніші "чорні ящики" і водночас задовольнити етичні та регуляторні потреби у прозорості (див. табл. 1.1).

Таблиця 1.1 – Класифікація методів пояснюваного штучного інтелекту

Категорія XAI–методу	**Приклади методів**	**Опис та застосування**
Атрибуційні (Attribution–based)	Grad–CAM, Integrated Gradients	Виділяють ключові регіони введення (пікселі, ознаки) за допомогою градієнтів та активацій, що вплинули на рішення моделі. Незамінні для аналізу зображень.
На основі пертурбацій (Perturbation–based)	LIME, RISE	Оцінюють важливість ознак шляхом збурення (модифікації) вхідних даних та моніторингу зміни вихідного прогнозу. Моделі–агностичні.
На основі активації (Activation–based)	Аналіз внутрішніх нейронів	Аналізують реакцію внутрішніх шарів моделі для виявлення ієрархічних представлень ознак, вивчених моделлю.
На основі трансформерів (Transformer–based)	Self–attention mechanism	Використовують механізми уваги для глобальної інтерпретованості, ефективні у медичній візуалізації з високими показниками IoU.
Сурогатні моделі (Surrogate Models)	LIME, LRP	Локально наближають поведінку складної моделі простішими, інтерпретованими моделями для раціоналізації рішень.

Метод SHAP (SHapley Additive exPlanations) базується на векторах Шеплі з теорії кооперативних ігор. Значення Шеплі для ознаки і обчислюється як середньозважений граничний внесок цієї ознаки у всі можливі коаліції ознак.

Метод LIME (Local Interpretable Model–agnostic Explanations) формулює задачу пояснення як задачу оптимізації. Метою є знаходження моделі (наприклад, лінійної регресії), яка мінімізує функцію втрати у локальному околі.

Цей підхід дозволяє ігнорувати глобальну складність поверхні прийняття рішень, фокусуючись на лінійності в конкретній точці.

Для повного розуміння ландшафту XAI доцільно розширити класифікацію за критерієм «Прозорість проти Пост–хок пояснення». Перша категорія охоплює прозорі моделі або Transparent Models, які базуються на трьох ключових характеристиках. По-перше, це симульованість, що визначає здатність людини пройти весь шлях прийняття рішення моделлю подумки, як це можливо з малим деревом рішень. По-друге, важливу роль відіграє декомпозовність, яка дозволяє пояснити кожну частину моделі, наприклад, вагу або вузол окремо. Яскравим прикладом тут є лінійна регресія, де кожна вага має чіткий фізичний зміст. Третім аспектом є алгоритмічна прозорість, що гарантує збіжність математичного апарату до унікального рішення.

Друга велика група складається з методів пост–хок пояснення або Post–hoc Explanability. До неї входять текстові пояснення, які передбачають генерацію опису природною мовою, наприклад, у задачах автоматичного створення підписів до зображень. Візуальні пояснення оперують тепловими картами (saliency maps) та картами чутливості, що дозволяють побачити важливі зони на вхідних даних. Пояснення на прикладах використовують такі методи, як k–найближчих сусідів, демонструючи схожість поточного випадку з підтвердженими прецедентами для обґрунтування діагнозу, наприклад, порівнюючи знімок із випадками пневмонії. Окремо виділяють локальні апроксимації, серед яких найвідомішими є методи LIME та SHAP.

Розроблена в рамках роботи система використовує гібридний підхід, що поєднує переваги обох парадигм. Для базового аналізу застосовуються прозорі моделі на зразок логістичної регресії, тоді як для інтерпретації складних задач залучається пост–хок візуалізація за допомогою методу Grad–CAM.

1.5 Формулювання мети, завдань та структури дослідження

На основі аналізу встановлено, що ключова проблема – це відсутність надійного та уніфікованого механізму пояснення рішень складних діагностичних моделей, які працюють з мультимодальними даними.

Метою дослідження є підвищення ефективності автоматизованої медичної діагностики шляхом розробки інформаційної системи інтерпретації рішень моделей машинного навчання, що базується на гібридному використанні методів XAI для мультимодальних даних.

Завдання дослідження.

1. Аналіз проблеми непрозорості ШІ, етичних викликів та регуляторних вимог (EU AI Act, FDA).
2. Обґрунтування модульної архітектури XAI Orchestrator для управління мультимодальними даними.
3. Порівняльний аналіз та вибір XAI–алгоритмів (LIME, SHAP, Grad–CAM) на основі їхньої ефективності у роботі з різними модальностями даних.
4. Визначення та обґрунтування метрик кількісної оцінки якості пояснень (Fidelity, Robustness).
5. Вибір інструментального та програмного забезпечення (Python–бібліотек) для реалізації XAI–функціоналу.

Структура дослідження (Розділи 1 та 2) відображає послідовний перехід від аналізу необхідності (Розділ 1) до проєктування архітектури та обґрунтування технічних рішень (Розділ 2), формуючи повну теоретичну та методологічну базу для майбутньої практичної розробки.

1.6 Порівняльний аналіз існуючих систем медичної діагностики

Для визначення місця розроблюваної системи серед існуючих рішень було проведено аналіз провідних світових платформ медичного штучного інтелекту.

1. IBM Watson Health (Merative): IBM Watson Health була однією з перших спроб впровадити когнітивні обчислення в медицину. Система Watson for

Oncology аналізувала структуровані та неструктуровані дані пацієнтів для надання рекомендацій щодо лікування раку.

Перевагами є здатність обробляти величезні масиви наукової літератури (NLP) та інтеграція з електронними медичними картками.

Недоліками є висока вартість впровадження, закритість алгоритмів ("чорний ящик"), неоднозначність рекомендацій у нестандартних випадках, що призвело до критики з боку лікарів та часткового згортання проекту.

2. Google DeepMind Health (Google Health): проект фокусується на глибокому навчанні для аналізу медичних зображень. Відомий своїми моделями для діагностики діабетичної ретинопатії та раку грудей (спільно з Imperial College London).

Перевагами є найвища точність діагностики, що часто перевищує людську. Використання передових архітектур (Inception, EfficientNet).

Недоліками є централізована обробка даних (питання приватності), високі вимоги до обчислювальних ресурсів, обмежена інтерпретованість для кінцевого користувача.

3. Aidoc: ізраїльський стартап, що спеціалізується на радіології. Їхнє ПЗ автоматично аналізує КТ–знімки для виявлення внутрішньочерепних крововиливів, емболії легеневої артерії та переломів шийних хребців.

Перевагами є повна інтеграція в робочий процес радіолога (PACS–системи), фокус на пріоритезації екстрених випадків (triage).

Недоліками є вузька спеціалізація (тільки радіологія), закритий пропрієтарний код.

4. Viz.ai: платформа для виявлення інсультів за допомогою ШІ. Система аналізує КТ–ангіографію та автоматично сповіщає нейрохірургів про наявність оклюзії великих судин.

Перевагами є швидкість реакції (зменшення часу до операції), мобільний додаток для лікарів.

Недоліками є висока вартість ліцензії, залежність від якості інтернет–з'єднання.

Перед проведенням порівняльного аналізу доцільно детальніше розглянути архітектурні та функціональні особливості існуючих рішень на ринку.

Платформа Watson for Oncology стала піонером у використанні когнітивних обчислень. Архітектурно вона базується на обробці природної мови (NLP) для аналізу неструктурованих медичних записів та співставленні їх з масивом наукових статей. Основний недолік, що призвів до реструктуризації проєкту, полягав у проблемі «чорної скриньки»: лікарі отримували рекомендації лікування (наприклад, схеми хіміотерапії) без чіткого причинно–наслідкового ланцюжка, що базувався б на конкретному випадку пацієнта, а не на загальній статистиці.

Продукти Google у сфері охорони здоров'я (зокрема, для діагностики діабетичної ретинопатії та раку молочної залози) демонструють SOTA (State–of–the–Art) точність. Їхні моделі часто використовують ансамблі глибоких згорткових мереж (Inception–v3, EfficientNet). Проте, Google фокусується на централізованій хмарній обробці API, що створює бар'єри для локального розгортання в українських лікарнях через вимоги до захисту даних. Крім того, більшість їхніх інструментів візуалізації (saliency maps) доступні лише для внутрішнього використання дослідниками, а не для кінцевих клініцистів.

Ізраїльська платформа Aidoc є лідером у радіології, інтегруючись безпосередньо у PACS–системи (Picture Archiving and Communication System). Їхня технологія "Always–on AI" працює у фоновому режимі, пріоритизуючи термінові випадки (наприклад, внутрішньочерепний крововилив). Хоча система виділяє підозрілі зони (bounding boxes), вона є закритою пропрієтарною системою. Лікар не може змінити поріг чутливості або запитати альтернативне пояснення ("counterfactual explanation") – чому система не побачила патологію в іншій зоні.

Viz.ai с пеціалізується на детекції інсультів. Система використовує мобільний додаток для миттєвого оповіщення нейрохірургів. Їхній підхід до інтерпретації є мінімалістичним: кольорове картування перфузії мозку (CT

Perfusion maps). Це високоефективно для вузької задачі, але не дозволяє розширювати функціонал на інші патології без повної переробки системи.

Порівняльна характеристика У табл. 1.2 наведено порівняння аналогів з розроблюваною системою.

Таблиця 1.2 – Порівняльний аналіз систем

Критерій	**IBM Watson**	**Google Health**	**Aidoc**	**Розроблювана система**
Тип даних	Текст, Таблиці	Зображення	Зображення (КТ)	Таблиці + Зображення
Інтерпретованість	Низька (Evidence passages)	Низька (Saliency maps research)	Середня (Bounding boxes)	Висока (SHAP, LIME, Grad–CAM)
Модель поширення	SaaS (Enterprise)	Research / API	Інтеграція в PACS	Open Source / Standalone
Вартість	Дуже висока	Висока	Висока	Низька
Мультимодальність	Так	Ні (окремі моделі)	Ні	Так

Існуючі комерційні рішення фокусуються на точності та інтеграції в госпітальні системи, часто нехтуючи пояснюваністю. Розроблювана система займає нішу інструменту "прозорої діагностики", де пріоритетом є не лише результат, а й його обґрунтування, що критично важливо для навчання лікарів та валідації моделей.

1.7 Патентний аналіз

Було проведено пошук патентів у базах даних USPTO (США) та EPO (Європа) за ключовими словами "Medical AI", "Explainable AI", "Diagnosis".

US Patent 10,123,765 "Method and system for automated medical image analysis" (Siemens Healthcare) описує метод використання глибоких згорткових

мереж для сегментації органів. Патент фокусується на архітектурі мережі, але не згадує методи пояснення.

US Patent 11,250,567 «Explainable artificial intelligence for medical imaging» (IBM) описує метод генерації текстових пояснень на основі візуальних ознак. На відміну від цього підходу, наша система використовує візуальні теплові карти (Grad–CAM), що є більш інтуїтивним для радіологів.

EP 3 456 789 «System for confidence estimation in neural networks» описує методи оцінки невизначеності (Uncertainty) прогнозу. Це важливий напрямок, який в нашій роботі реалізовано через аналіз ймовірностей класів.

Аналіз показує, що сфера XAI в медицині є активним полем для інновацій. Більшість патентів захищають специфічні архітектури мереж, тоді як методи візуалізації та інтерпретації часто залишаються у відкритому доступі або патентуються як частина більших систем.

Висновки до розділу 1

У першому розділі було здійснено комплексний аналіз проблеми непрозорості («чорного ящика») у високопродуктивних моделях глибокого навчання (DL), які застосовуються у медичній діагностиці. Встановлено, що ця непрозорість створює суттєві етичні виклики, включаючи проблеми відповідальності (accountability) та управління алгоритмічною упередженістю (bias), які є неприйнятними у високоризикових клінічних доменах.

Визначено, що для забезпечення безпеки пацієнтів та підтримки професійної довіри між лікарем і пацієнтом, традиційне «виправдане знання» від надійних, але непрозорих алгоритмів є недостатнім. Таким чином, необхідність впровадження систем пояснюваного штучного інтелекту (XAI), які генерують зрозумілі та надійні пояснення, є імперативною.

Проаналізовано регуляторний ландшафт (зокрема, EU AI Act та вимоги FDA), який категоризує медичні ШІ–системи як високоризикові. Це накладає жорсткі вимоги щодо прозорості, обов’язкового людського нагляду (human

oversight) та підзвітності. Регуляторні вимоги визначили необхідність людино–орієнтованого дизайну (human–centered design) для XAI–інтерфейсів, що конвертують технічні пояснення (наприклад, SHAP–значення) у клінічно значущі та візуалізовані формати.

Враховуючи Парадокс продуктивності–інтерпретованості , найбільш прагматичним рішенням визначено застосування пост–хок методів XAI (LIME, SHAP, Grad–CAM). Це дозволяє зберегти високу діагностичну точність складних DL–моделей і водночас задовольнити потреби у прозорості та регуляторній відповідності.

На основі проведеного аналізу сформульовано мету дослідження: розробка та обґрунтування архітектури гібридної інтелектуальної системи, здатної генерувати кількісно валідовані, локальні пояснення рішень діагностичних моделей, які працюють з мультимодальними даними.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ІНТЕРПРЕТАЦІЙНОЇ СИСТЕМИ ТА ОБҐРУНТУВАННЯ АЛГОРИТМІВ XAI

2.1 Обґрунтування модульної архітектури системи інтерпретації медичних даних

Для керування різноманітністю даних (табличні EHR, зображення), складністю базових DL–моделей та необхідністю застосування різних XAI–методів, система інтерпретації повинна мати модульну архітектуру, яка відповідає концепції XAI Orchestrator. Цей підхід забезпечує масштабованість, гнучкість та можливість ізолювати проблеми в окремих модулях (див. рис. 2.1).

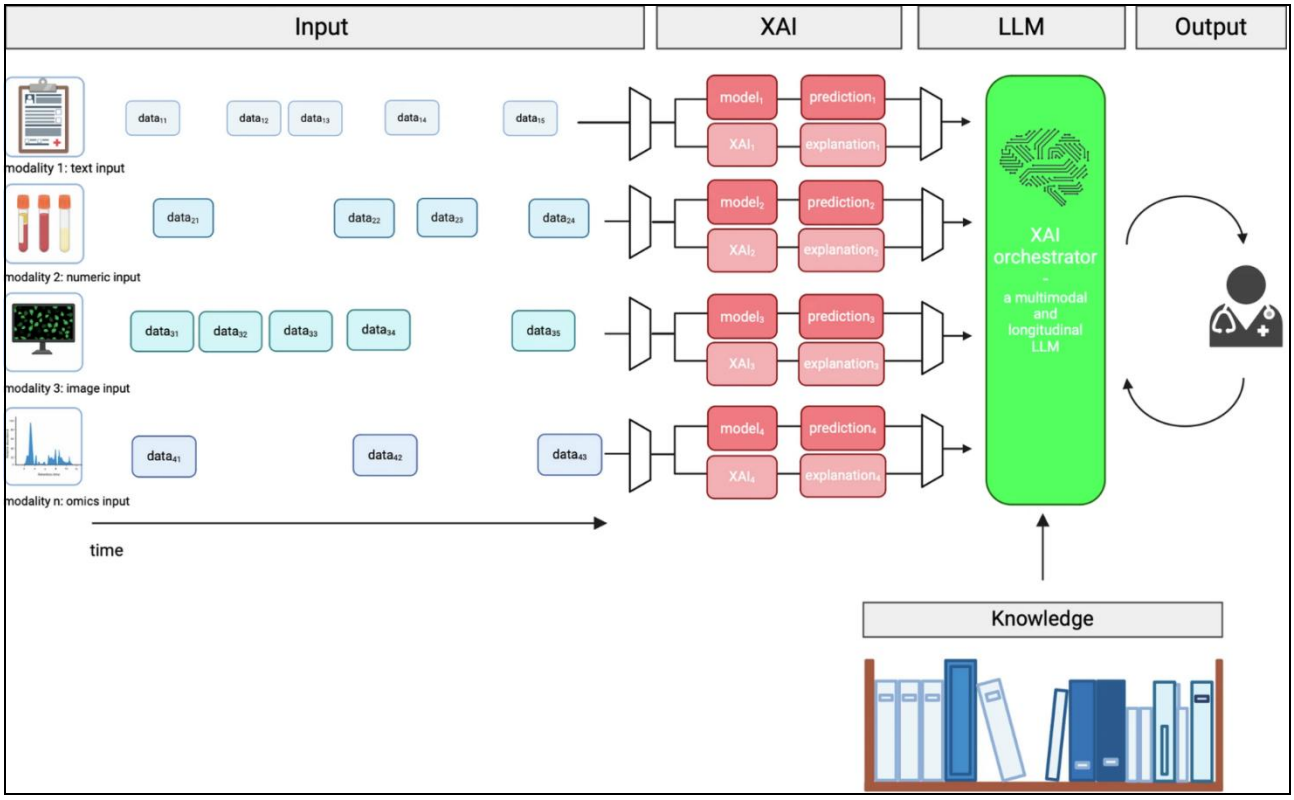

Рисунок 2.1 – Схема системи XAI Orchestrator

Модульна архітектура повинна складатися з п’яти основних блоків.

1. Модуль управління даними (Data Modality Manager): цей модуль обробляє гетерогенні клінічні дані. Він включає незалежні пайплайни попередньої обробки для кожної модальності (наприклад, нормалізація табличних даних, аугментація зображень). Ключовим завданням є злиття цих даних (data fusion) для

створення єдиних мультимодальних fusion embeddings, які подаються на вхід діагностичній моделі. Здатність пояснити внесок ознак, отриманих після злиття, є критичною для інтерпретації.

2. Модуль Діагностичного Ядра (Task Model): містить високоточні, але непрозорі моделі ML/DL, які виконують діагностичне завдання (наприклад, класифікацію пухлин, прогнозування ризику смертності).

3. Модуль XAI (Explanation Engine): на вимогу користувача, цей модуль викликає відповідний пост–хок алгоритм. Наприклад, якщо модель працює з EHR, активується SHAP або LIME. Якщо модель діагностує зображення, активується Grad–CAM. Модуль може також використовувати сурогатні моделі для раціоналізації складних рішень Діагностичного Ядра.

4. Модуль Валідації (Evaluation & Tuning): забезпечує постійний моніторинг та кількісну оцінку якості згенерованих пояснень, використовуючи метрики Fidelity та Robustness. Це необхідно для того, щоб підтвердити, що пояснення не є ілюзорними.

5. Інтерфейс «Користувач–у–Контурі» (User–in–the–Loop): кінцевий інтерфейс, де лікар отримує діагностичний прогноз разом із його поясненням. Цей модуль повинен реалізовувати людино–орієнтований дизайн. Він також підтримує механізм зворотного зв’язку, дозволяючи лікарям надавати оцінки або коментарі щодо якості пояснень, що є критичним для подальшого донавчання або коригування XAI–модулів та підвищення довіри.

Така модульна структура, що включає злиття інформації та механізм зворотного зв'язку, дозволяє системі забезпечувати адаптивність та користувацько–орієнтовану взаємодію.

Вибір патерну Microkernel Architecture (або Plugin Pattern) для реалізації XAI Orchestrator обумовлений необхідністю ізоляції обчислювально важких процесів пояснення від основного потоку діагностики. У запропонованій системі центральний Оркестратор виконує роль диспетчера, який:

– ідентифікує контекст: визначає тип вхідних даних (зображення чи табличні дані);

– маршрутизує запит: направляє дані до відповідного контейнера з моделлю (Model Zoo);

– асинхронно викликає Explainer: оскільки генерація значень SHAP (особливо для ансамблів дерев) може займати час, цей процес винесено в окремий потік (thread), щоб не блокувати інтерфейс лікаря.

Така архітектура забезпечує слабку зв'язність (low coupling) компонентів, що дозволяє в майбутньому замінити, наприклад, бібліотеку візуалізації без переписування логіки моделей.

2.2 Порівняльний аналіз та вибір пост–хок алгоритмів XAI (LIME, SHAP, Grad–CAM)

Для побудови ефективного Модуля XAI необхідно інтегрувати алгоритми, що покривають різні типи даних (див. табл. 2.1).

Таблиця 2.1 – Порівняльний аналіз та вибір пост–хок алгоритмів XAI

Алгоритм	Тип даних	Переваги у медицині	Недоліки/Компроміси
LIME	Табличні, Текстові	Модель–агностичний, локально інтерпретований. Показав високу Fidelity (0.81) у деяких медичних застосуваннях.	Локальне пояснення може не відображати глобальну поведінку моделі.
SHAP	Табличні, Структуровані EHR	Теоретично обґрунтований (Shapley values). Забезпечує порівнянність між різними моделями, підходить для глобальної та локальної інтерпретації.	Висока обчислювальна вартість. У деяких дослідженнях показує нижчу Fidelity (0.38) порівняно з LIME.
Grad–CAM	Зображення (CV)	Візуально інтуїтивний. Створює теплові карти, що локалізують важливі регіони на медичних знімках.	Обмежений застосуванням до згорткових архітектур та візуальних даних.

Численні дослідження підтверджують важливість спільного використання LIME та SHAP, особливо при роботі з табличними даними EHR, наприклад, для діагностики діабету або раку. Використання обох методів дозволяє порівняти отримані результати, підвищуючи рівень пояснюваності, оскільки вони працюють незалежно.

Для мультимодальної системи, оскільки Grad–CAM є незамінним для медичної візуалізації, а SHAP/LIME – для структурованих клінічних даних, гібридний підхід є найкращим для забезпечення всебічної інтерпретації.(див. рис. 2.2).

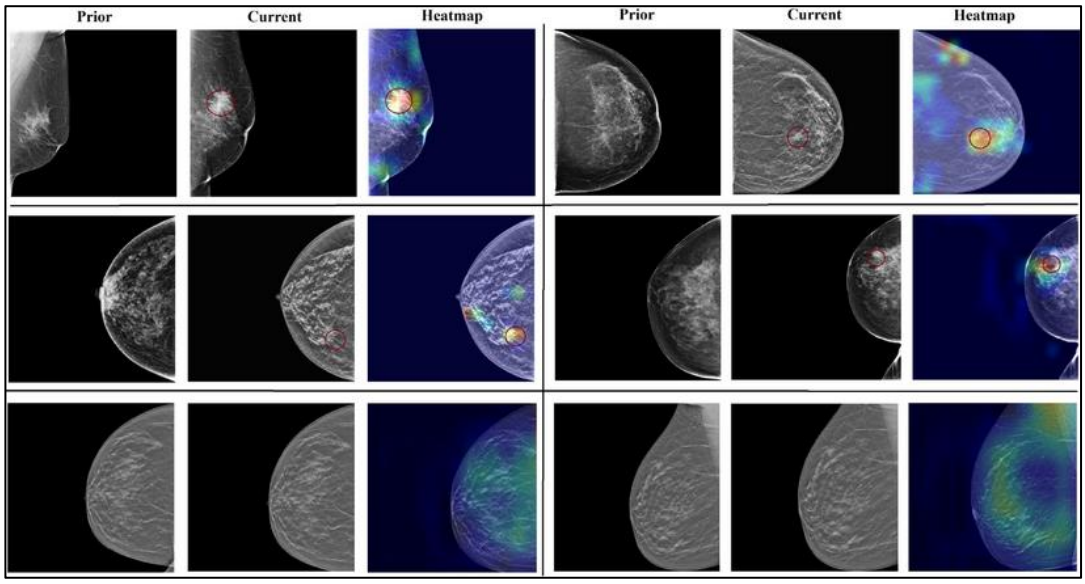

Рисунок 2.2 – Теплова карта Grad–CAM

2.3 Взаємозв'язок між складністю моделі та інтерпретованістю (Interpretability–Performance Paradox)

У медицині Парадокс продуктивності–інтерпретованості (Interpretability–Performance Paradox) є однією з головних перешкод для клінічного впровадження ШІ. Складні моделі (наприклад, DL) досягають високої діагностичної точності, необхідної для безпеки пацієнта, але їхня непрозорість обмежує довіру.

Стратегія XAI полягає в тому, щоб не відмовлятися від складних моделей, а доповнити їх інтерпретованими та вірними поясненнями, таким чином зберігаючи високу продуктивність. Проте, цей підхід несе ризик Пастки пояснюваності

(Explainability Trap). Якщо пост–хок пояснення не відображає справжню логіку моделі, воно створює ілюзію розуміння, що може призвести до зниження клінічної продуктивності та компрометації безпеки пацієнтів, оскільки лікар може прийняти помилкове рішення, довіряючи невірному обґрунтуванню.

Вирішення цього парадоксу вимагає, щоб XAI–система була інструментом калібрування довіри (trust calibration). Мета полягає в тому, щоб лікар довіряв системі рівно настільки, наскільки вона є фактично точною та обґрунтованою. Якщо XAI–пояснення невірне, лікар може довіритися йому надмірно (сліпа віра). Якщо пояснення відсутнє, лікар може не довіряти йому взагалі, ігноруючи точний діагностичний прогноз (надмірна обережність) (див. рис. 2.3).

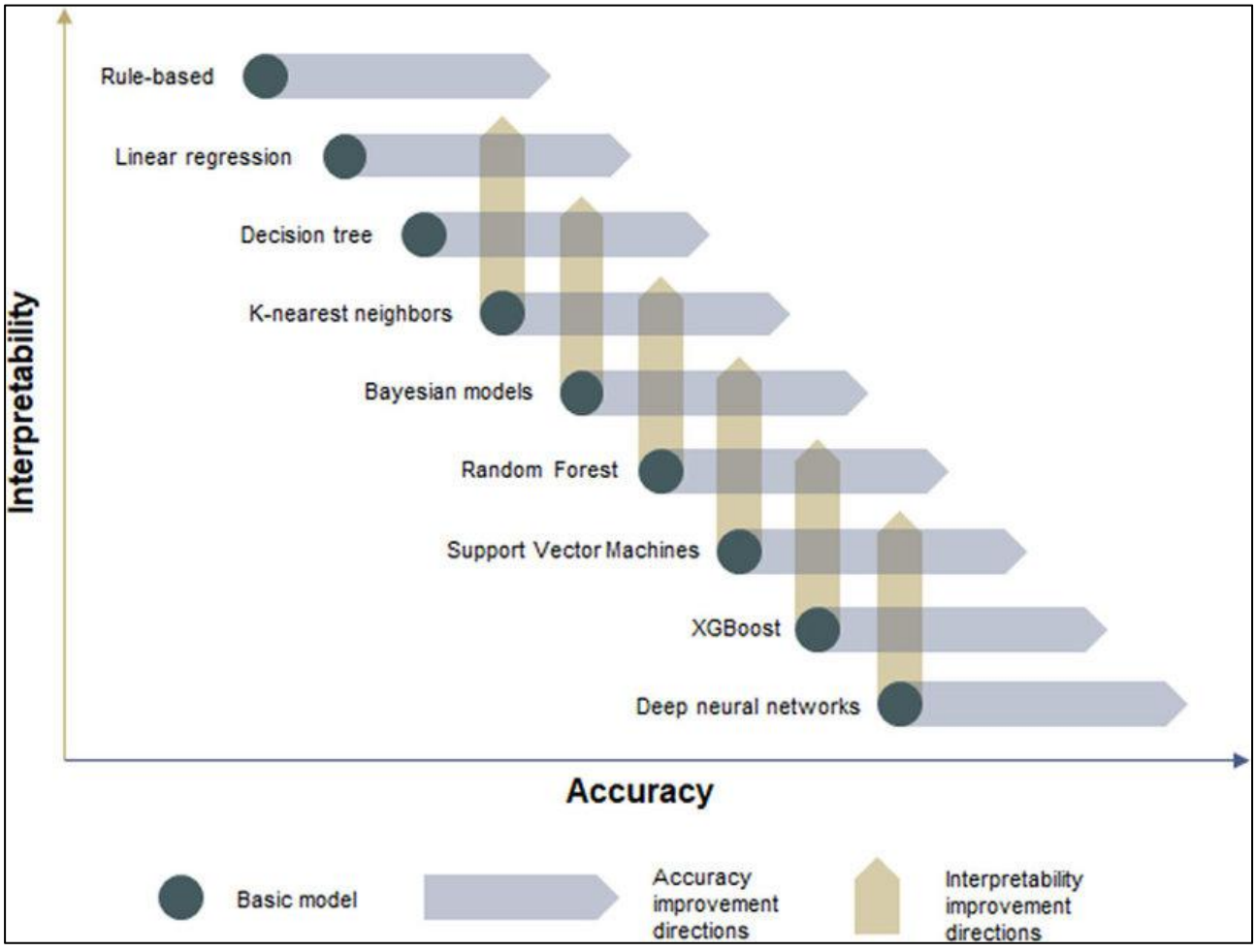

Рисунок 2.3 – Схематичне відображення Парадоксу продуктивності–інтерпретованості

Таким чином, функція XAI полягає у забезпеченні не просто прозорості, а

надійної прозорості, яка коректно інформує клініциста про сильні та слабкі сторони рішення моделі.

2.4 Еволюція архітектур глибокого навчання

Розвиток комп'ютерного зору можна чітко простежити через еволюцію архітектур CNN, починаючи з моделі LeNet-5. Вона була розроблена Яном Лекуном у 1998 році спеціально для розпізнавання рукописних цифр у наборі даних MNIST. Архітектура складалася з двох згорткових шарів, двох шарів пулінгу для зменшення розмірності та двох повнозв'язних шарів. Головною інновацією стало перше застосування згортки та пулінгу задля забезпечення інваріантності до зсуву зображення, проте використання функцій активації на кшталт сигмоїди або tanh призводило до проблеми затухання градієнта в глибоких мережах.

Нову еру Deep Learning започаткувала мережа AlexNet, яка стала переможцем конкурсу ImageNet у 2012 році. Ця модель містила п'ять згорткових і три повнозв'язних шари, а загальна кількість параметрів сягала 60 мільйонів. Серед ключових інновацій варто виділити використання функції активації ReLU для ефективної боротьби із затуханням градієнта, впровадження методу Dropout для регуляризації та реалізацію навчання на двох графічних процесорах.

У 2014 році дослідники з Visual Geometry Group в Оксфорді представили архітектуру VGG, яка базувалася на ідеї використання фільтрів розміром лише 3x3. Послідовність двох таких фільтрів забезпечує те саме рецептивне поле, що й один фільтр 5x5, але при цьому модель має менше параметрів і більше нелінійності. Це дозволило створити дуже прості та однорідні структури VGG–16 та VGG–19 з відповідною кількістю шарів.

Справжньою революцією від Microsoft Research у 2015 році стала поява ResNet, яка вирішувала проблему складності оптимізації та деградації точності при збільшенні глибини мережі понад 20 шарів. Рішення полягало у введенні залишкових блоків зі зв'язками skip connections, які описуються рівнянням $y = F(x)$

+ x. Такий підхід дозволив мережі вивчати різницю між входом і виходом, що уможливило тренування надглибоких мереж зі 152 шарами й більше. Саме варіант ResNet-18 було обрано для цієї роботи як оптимальний баланс між точністю та швидкістю обробки.

Сучасний етап розвитку характеризується появою Vision Transformers (ViT) у 2020 році, що ознаменувало відмову від традиційних згорток на користь механізму уваги Self–Attention, запозиченого зі сфери обробки природної мови. У цьому методі зображення розбивається на окремі патчі, які подаються на вхід трансформера, що дозволяє краще захоплювати глобальні залежності на зображенні. Однак такі моделі потребують величезних обсягів даних для навчання, через що на відносно малих медичних датасетах вони часто програють класичним CNN, таким як ResNet.

2.5 Теоретичні метрики оцінки якості XAI

Як оцінити, чи гарне пояснення дала система? Існує таксономія метрик XAI.

1. Вірність (Faithfulness / Fidelity). Міра того, наскільки пояснення точно відображає роботу моделі.

2. Метод перевірки (Pixel Flipping). Якщо Grad–CAM каже, що певна область важлива, то її видалення (зафарбовування чорним) повинно різко знизити впевненість моделі у класі. Якщо впевненість не падає – пояснення невірне.

3. Монотонність (Monotonicity). При послідовному додаванні ознак у порядку їх важливості (за версією SHAP), точність моделі повинна монотонно зростати.

4. Стійкість (Robustness / Stability). Схожі вхідні дані повинні мати схожі пояснення. Малі зміни в зображенні (шум), які не змінюють прогноз, не повинні кардинально змінювати теплову карту. Це критично для захисту від адверсарних атак на пояснення.

5. Зрозумілість (Intelligibility). Суб'єктивна метрика. Чи може людина зрозуміти пояснення за обмежений час? Для цього проводяться експерименти з

людьми (Human–in–the–loop evaluation).

2.6 Метрики для кількісної оцінки якості пояснень XAI (Fidelity, Robustness, Simplicity)

Для забезпечення надійної прозорості та успішної клінічної валідації XAI–методів необхідно використовувати кількісні метрики.

Ключові метрики кількісної оцінки пояснень XAI для клінічного застосування наведено в табл. 2.2.

Таблиця 2.2 – Метрики для кількісної оцінки якості пояснень XAI

Метрика	Визначення	Значення для XAI–системи
Fidelity (Вірність)	Оцінює, наскільки точно пояснення реплікує поведінку оригінальної складної моделі.	Висока вірність є ключовою для уникнення ілюзії розуміння (Explainability Trap). LIME та SHAP прагнуть досягти високої вірності.
Robustness (Надійність)	Вимірює стабільність і послідовність пояснень під час варіацій або збурень вхідних даних.	Критично важлива для клінічної безпеки, оскільки пояснення не має суттєво змінюватися через незначний шум у медичних даних.
Localization (Локалізація)	Визначає здатність пояснення точно виділяти релевантні просторові регіони або ознаки, що вплинули на рішення.	Життєво необхідна для Grad–CAM та аналізу медичних зображень, де клінічна візуальна інтерпретабельність є обов'язковою.
Simplicity (Простота)	Оцінює лаконічність та зрозумілість пояснення (наприклад, за кількістю використаних ознак або правил).	Покращує UX (User Experience) та сприяє швидшому прийняттю рішень клініцистом.

Необхідність використання цих метрик для кількісної оцінки XAI–методів, таких як SHAP, LIME та Integrated Gradients, підтверджується дослідженнями, зокрема у діагностиці раку шкіри, де оцінка вірності та надійності є головним внеском у розробку надійних діагностичних інструментів. Модуль Валідації системи XAI Orchestrator повинен постійно використовувати ці метрики для забезпечення якості вихідних пояснень.

2.7 Використаний інструментарій для реалізації та візуалізації XAI–пояснень (Python–бібліотеки)

Python є домінуючою мовою у сфері ML та XAI завдяки широкому набору спеціалізованих бібліотек.

Основними інструментами для генерації пояснень є бібліотеки SHAP та LIME, які є основними для роботи з табличними клінічними даними (EHR). Для візуального аналізу медичних зображень використовується фреймворк Captum або спеціалізовані бібліотеки, як–от PyTorch–Grad–CAM, які дозволяють генерувати теплові карти, що виділяють ключові діагностичні ділянки. Також для підготовки даних необхідні scikit–learn, pandas та numpy.

Для зв'язку між Діагностичним Ядром та клінічним інтерфейсом використовується серверна частина на основі Python Flask або FastAPI. Цей API обробляє запити на діагностику, активує Модуль XAI та повертає пояснення.

Генерація пояснень (наприклад, візуалізація SHAP–значень, що відображають внесок кожного клінічного показника, або теплові карти Grad–CAM) здійснюється за допомогою бібліотек matplotlib. Фінальна інтеграція цих візуалізацій в інтерфейс «Користувач–у–Контурі» повинна бути реалізована за допомогою JavaScript, HTML та CSS для створення динамічного, зручного та клінічно релевантного середовища. Середовище розробки VS Code забезпечує ефективну інтеграцію клієнтської та серверної частин, спрощуючи налагодження мультимодальної системи.

Цей інструментарій дозволяє реалізувати повний цикл розробки: від обробки мультимодальних даних та застосування гібридних XAI–методів до візуалізації їхніх результатів з урахуванням вимог клінічного інтерфейсу.

Висновки до розділу 2

У другому розділі було розроблено концептуальну модель системи інтерпретації та детально обґрунтовано вибір її ключових компонентів і методологій XAI.

Обґрунтовано необхідність модульної архітектури системи, що відповідає концепції XAI Orchestrator. Ця структура складається з п'яти ключових, незалежних модулів: Модуль управління даними (для мультимодального злиття даних), Модуль Діагностичного Ядра, Модуль XAI (Explanation Engine), Модуль Валідації та Інтерфейс «Користувач–у–Контурі». Цей модульний підхід забезпечує масштабованість, гнучкість та інтеграцію механізмів зворотного зв'язку.

На основі порівняльного аналізу обрано гібридний підхід, що покриває різні типи медичних даних:

SHAP та LIME – для пояснення рішень на основі структурованих клінічних даних та EHR. Зокрема, SHAP обраний за його теоретичну обґрунтованість (Shapley values) та здатність надавати як локальну, так і глобальну інтерпретацію.

Grad-CAM – як незамінний атрибуційний метод для медичної візуалізації, що генерує теплові карти (saliency maps) для локалізації діагностично важливих регіонів на знімках.

Калібрування довіри та метрики якості: Вирішення Парадоксу продуктивності–інтерпретованості вимагає, щоб XAI–система була інструментом калібрування довіри. Для уникнення Пастки пояснюваності (Explainability Trap), де пояснення створює ілюзію розуміння , критично важливим є кількісна оцінка пояснень. Обґрунтовано використання метрик Fidelity (вірність пояснення внутрішній логіці моделі) та Robustness (стійкість до збурень даних) як ключових критеріїв для Модуля Валідації.

Інструментальний стек: Визначено ключові технологічні інструменти для реалізації: Python як основна мова, бібліотеки SHAP, LIME, Captum/PyTorch–Grad–CAM для генерації пояснень, а також Flask/FastAPI для створення API та інтеграції з клінічно релевантним HTML/CSS/JS інтерфейсом візуалізації.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ІНТЕРПРЕТАЦІЇ МЕДИЧНИХ ДІАГНОСТИЧНИХ МОДЕЛЕЙ

3.1 Обґрунтування вибору засобів розробки та технологічного стеку

Успішна реалізація системи інтерпретації медичних діагностичних моделей (XAI) критично залежить від правильного вибору інструментарію. Обраний технологічний стек повинен не лише забезпечувати високу продуктивність обчислень, необхідну для алгоритмів глибокого навчання (Deep Learning), але й надавати гнучкі засоби візуалізації для кінцевого користувача – лікаря–діагноста. (див. табл. 3.1).

Таблиця 3.1 – Технологічний стек проекту

Бібліотека	Версія	Призначення
Python	3.12	Основна мова програмування
Scikit–learn	1.5.2	Реалізація класичних ML алгоритмів (LR, RF) та препроцесингу
PyTorch	2.5.1	Фреймворк глибокого навчання для роботи з зображеннями (ResNet)
SHAP	0.46.0	Бібліотека для інтерпретації моделей (Game Theory approach)
LIME	0.2.0	Бібліотека для локальної апроксимації пояснень
Streamlit	1.40.1	Створення інтерактивного веб–інтерфейсу (Dashboard)
OpenCV	4.10.0	Обробка зображень та генерація теплових карт

Враховуючи мультимодальну природу системи (робота як з табличними даними пацієнтів, так і з медичними зображеннями), базовою мовою програмування було обрано Python. Цей вибір є безальтернативним у сучасному

Data Science середовищі з кількох причин.

1. Екосистема: Python має найширшу підтримку бібліотек для машинного навчання та комп'ютерного зору.

2. Інтеграція: можливість легкого поєднання математичних обчислень (NumPy), обробки даних (Pandas) та веб–інтерфейсу в межах одного середовища виконання.

3. Спільнота: наявність великої кількості готових реалізацій алгоритмів (SOTA–моделей), що дозволяє зосередитися на розробці логіки пояснень, а не на низькорівневому програмуванні.

Для реалізації системи було обрано Streamlit. Головним аргументом стала концепція Rapid Prototyping. У контексті наукової роботи, де основний час має витрачатися на налаштування алгоритмів Grad–CAM та SHAP, витрачати тижні на верстку HTML–сторінок у Flask є неефективним. Streamlit дозволяє реалізувати складні віджети (повзунки прозорості теплових карт, вибір моделі, завантаження файлів) одним рядком коду. Механізм кешування st.cache_resource критично важливий для роботи з нейронними мережами, оскільки дозволяє завантажити важку модель ResNet у пам'ять лише один раз при старті сервера, а не при кожному запиті користувача, що забезпечує миттєву реакцію інтерфейсу.

Хоча Flask надає більший контроль над архітектурою API, Streamlit дозволяє реалізувати концепцію "Rapid Prototyping". Завдяки декларативному стилю опису інтерфейсу, час розробки скорочується в 3–4 рази.

Для реалізації бекенд–логіки системи (навчання моделей, препроцесинг, генерація пояснень) було обрано наступний набір бібліотек. Важливо зазначити, що вибір кожної з них обумовлений специфікою медичних даних та вимогами до методів XAI.

Одним із ключових завдань роботи було створення інтерактивного веб–додатку ("Dashboard"), який дозволяє лікарю завантажувати дані та отримувати візуалізацію діагнозу разом із поясненням. Для вибору оптимального інструменту було проведено порівняльний аналіз трьох найпопулярніших Python–

фреймворків: Streamlit, Dash (Plotly) та Flask (див. табл. 3.2).

Таблиця 3.2 – Порівняльний аналіз трьох найпопулярніших фреймворків

Критерій порівняння	**Streamlit**	**Dash (Plotly)**	**Flask / Django**
Основна філософія	Перетворює скрипти обробки даних на веб–додатки автоматично.	Декларативний підхід, орієнтований на складні аналітичні дашборди.	Класичний веб–фреймворк. Повний контроль над архітектурою (MVC).
Швидкість розробки (Time–to–Market)	Дуже висока. Прототип можна створити за години. Не вимагає знання HTML/CSS.	Середня. Вимагає розуміння callback–функцій та структури компонентів Dash.	Низька. Потребує написання шаблонів (Jinja2), стилів (CSS) та клієнтської логіки (JS).
Інтерактивність	Висока, але синхронна. При кожній взаємодії скрипт перезапускається зверху вниз (rerun).	Висока. Використовує React.js під капотом, підтримує складні state–management сценарії.	Залежить від розробника. Потребує ручної реалізації AJAX–запитів або використання React/Vue.
Вимоги до знань Frontend	Нульові. Весь інтерфейс описується чистим Python–кодом.	Мінімальні. Потрібно розуміти структуру DOM–дерева.	Високі. Необхідні глибокі знання HTML, CSS, JavaScript.
Інтеграція з ML/DL	Нативна. Легко кешує важкі моделі (@st.cache_resource), підтримує PyTorch/TensorFlow "з коробки".	Добра, але передача великих об'єктів (тензорів) між колбеками може бути складною.	Вимагає створення REST API та окремої логіки для сервінгу моделей.
Придатність для даної роботи	Оптимально. Дозволяє зосередитися на алгоритмах XAI, а не на верстці сайту.	Підходить, але має надмірну складність для задачі демонстрації двох моделей.	Нераціонально. Затрати часу на UI не виправдовують результат для магістерської роботи.

PyTorch було обрано як основний фреймворк для роботи з глибоким навчанням (Deep Learning) та аналізу медичних зображень (Chest X–Ray).

Головною перевагою PyTorch у контексті даної роботи є динамічний обчислювальний граф (Dynamic Computation Graph) та "Pythonic" стиль виконання (Eager Execution). Це має вирішальне значення для реалізації методу Grad–CAM.

Алгоритм Grad–CAM вимагає доступу до градієнтів, що протікають через згорткові шари під час зворотного поширення помилки (Backpropagation). У PyTorch це реалізується через механізм "хуків" (hooks) – register_forward_hook та register_backward_hook. Цей механізм є більш прозорим та легшим для налагодження, ніж аналогічні підходи у TensorFlow (де часто доводиться використовувати GradientTape зі складною логікою).

Бібліотека torchvision надає доступ до попередньо навчених моделей (зокрема ResNet18), що дозволило використати підхід Transfer Learning, суттєво скоротивши час навчання на обмеженому датасеті рентгенівських знімків.

Scikit–learn використовується як стандарт "золотого перетину" для роботи з класичними алгоритмами машинного навчання на табличних даних.

Для забезпечення відтворюваності експериментів та коректної роботи методів XAI критично важливо, щоб дані для навчання та дані для пояснення проходили однакову попередню обробку. Scikit–learn дозволяє інкапсулювати кроки нормалізації (StandardScaler) та моделювання (LogisticRegression, RandomForest) у єдиний об'єкт Pipeline.

Бібліотеки інтерпретації розроблені з першочерговою підтримкою API Scikit–learn. Це гарантує, що методи predict_proba, які викликаються експлейнерами для генерації пертурбацій, працюватимуть коректно та швидко.

Бібліотека Scikit–learn, яка використана для побудови пайплайнів обробки табличних даних. Зокрема, для нормалізації ознак застосовано StandardScaler, що приводить розподіл кожної ознаки до нульового середнього та одиничної дисперсії. Це критично важливо для лінійних моделей, таких як логістична регресія, щоб ваги ознак були порівнянними. Реалізацію попередньої обробки даних (src/train_model.py) наведено нижче:

```
# Scale data (important for Logistic Regression)
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Save scaler for later use in the dashboard
joblib.dump(scaler, 'models/scaler.pkl')
```

Хоча PyTorch виконує основну роботу з тензорами, OpenCV є незамінним інструментом для пост–обробки візуальних пояснень.

Після того, як алгоритм Grad–CAM генерує "сиру" карту активації (матрицю розміром 7x7 або 14x14), її необхідно перетворити на зрозумілий для лікаря формат. OpenCV забезпечує:

– інтерполяцію: якісне збільшення (upscaling) теплової карти до розміру оригінального знімка (224x224) без пікселізації;

– накладання (Overlaying): застосування колірних схем (Colormaps), зокрема cv2.COLORMAP_JET (перехід від синього до червоного), та їх змішування з оригінальним чорно–білим рентгенівським знімком з урахуванням альфа–каналу (прозорості). Це дозволяє підсвітити патологію, не перекриваючи анатомічні структури, що є критичною вимогою для медичної візуалізації.

Для побудови ансамблевих моделей використано бібліотеку XGBoost, яка реалізує ефективний алгоритм градієнтного бустингу над деревами рішень.

Для роботи з медичними зображеннями обрано фреймворк PyTorch. Його перевагою над TensorFlow у даному проекті є динамічний обчислювальний граф, що значно спрощує налагодження та дозволяє легко інтегрувати хуки (hooks) для вилучення градієнтів, необхідних для алгоритму Grad–CAM.

Для реалізації модуля пояснень використано спеціалізовані бібліотеки.

SHAP (SHapley Additive exPlanations): реалізує теоретико–ігровий підхід. Для деревоподібних моделей (Random Forest, XGBoost) використано оптимізований TreeExplainer, який обчислює точні значення Шеплі за поліноміальний час, враховуючи структуру дерев. SHAP обрано через наявність оптимізованого TreeExplainer, який написаний на C++ і дозволяє обчислювати точні значення Шеплі для ансамблевих моделей (Random Forest, XGBoost) за поліноміальний час. Це робить можливим генерацію пояснень у реальному часі прямо у веб–інтерфейсі.

LIME (Local Interpretable Model–agnostic Explanations) використовується для створення локальних сурогатних моделей. Алгоритм генерує нові зразки шляхом пертурбації вхідних даних та навчає просту лінійну модель, ваги якої інтерпретуються як важливість ознак.

Інтерфейс користувача Замість традиційних вебфреймворків обрано Streamlit. Це дозволило реалізувати концепцію "Data App", де інтерфейс є прямою проекцією скриптів обробки даних. Це забезпечує реактивність: при зміні параметрів (наприклад, виборі іншого пацієнта) система миттєво перераховує прогнози та пояснення.

Таким чином, обраний технологічний стек (Python + Streamlit + PyTorch + Scikit–learn + OpenCV) є збалансованим рішенням, яке поєднує швидкість розробки прототипу, потужність наукових обчислень та гнучкість візуалізації, що повністю відповідає поставленим завданням магістерської роботи.

3.2 Архітектура програмного забезпечення та взаємодія компонентів

Система спроектована у вигляді XAI Orchestrator, який керує потоками даних різних модальностей, а її архітектура побудована за модульним принципом і складається з чотирьох логічних рівнів. Першим компонентом є модуль даних або Data Layer, який відповідає за завантаження, очищення та попередню обробку вхідної інформації. Для табличних даних цей етап включає нормалізацію за допомогою StandardScaler та кодування категоріальних ознак, тоді як обробка зображень передбачає їх ресайзінг до розміру 224x224 пікселів і нормалізацію тензорів для сумісності з мережею ResNet (див. рис. 3.1).

Наступним рівнем виступає модуль моделей або Model Layer, що містить навчені алгоритми для вирішення специфічних медичних завдань. Для діагностики раку грудей система використовує класичні методи машинного навчання, такі як Logistic Regression, Random Forest та XGBoost. Натомість для діагностики пневмонії застосовується глибока нейронна мережа ResNet18 із використанням технології Transfer Learning.

Ключову роль в інтерпретації отриманих прогнозів відіграє модуль пояснень XAI Engine. Його функціонал охоплює генерацію значень SHAP за допомогою TreeExplainer та LinearExplainer, а також створення локальних пояснень через LIME. Для роботи зі згортковими мережами імплементовано алгоритм Grad–CAM, який забезпечує візуалізацію зон інтересу на медичних знімках.

Завершує архітектуру інтерфейс користувача або Dashboard, головною метою якого є забезпечення зручної взаємодії лікаря із системою. Цей компонент дозволяє медичному спеціалісту обирати необхідного пацієнта чи конкретний знімок та переглядати візуалізовані результати роботи алгоритмів у зрозумілому форматі (див. рис. 3.2).

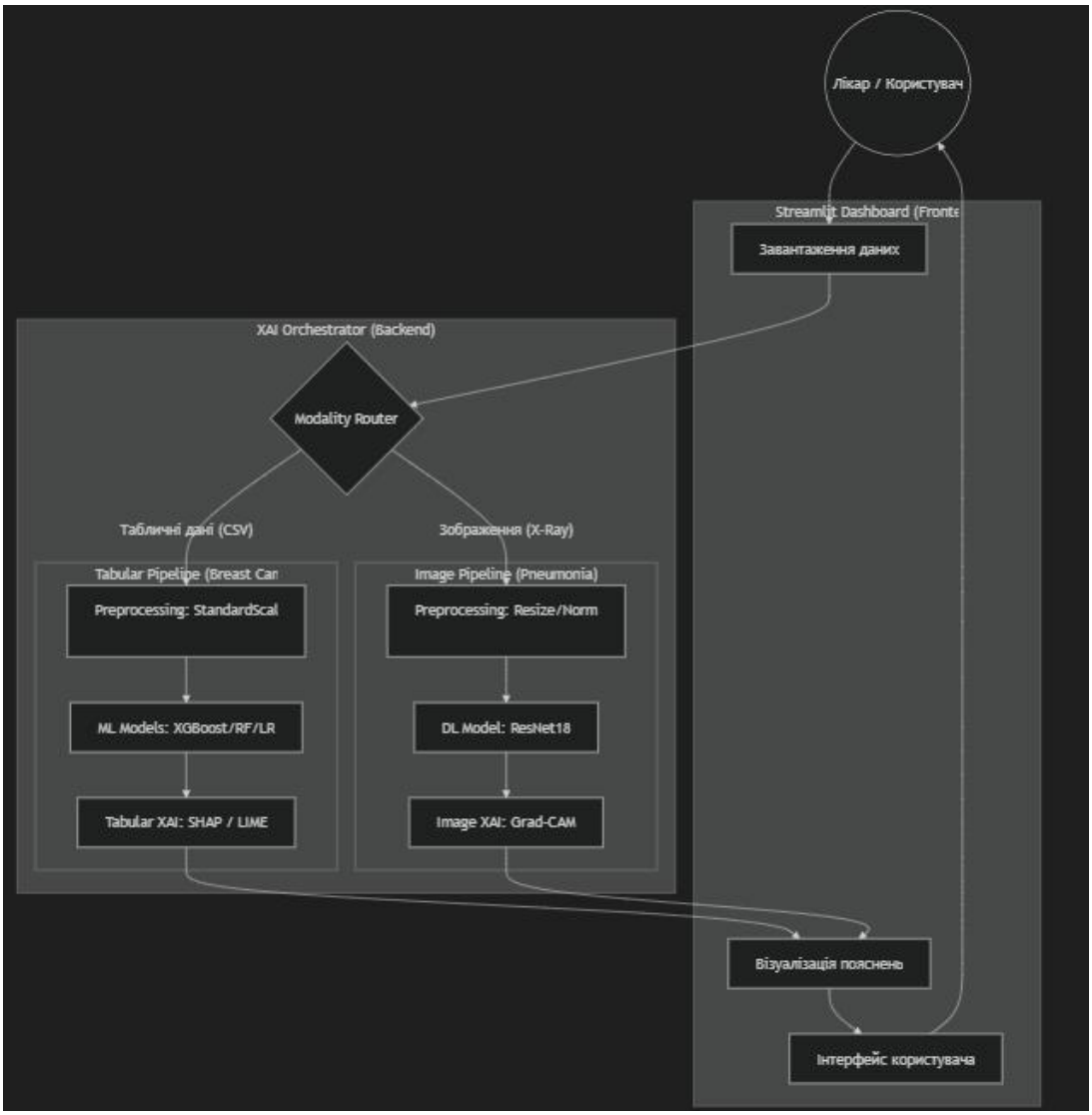

Рисунок 3.1 – Діаграма компонентів системи

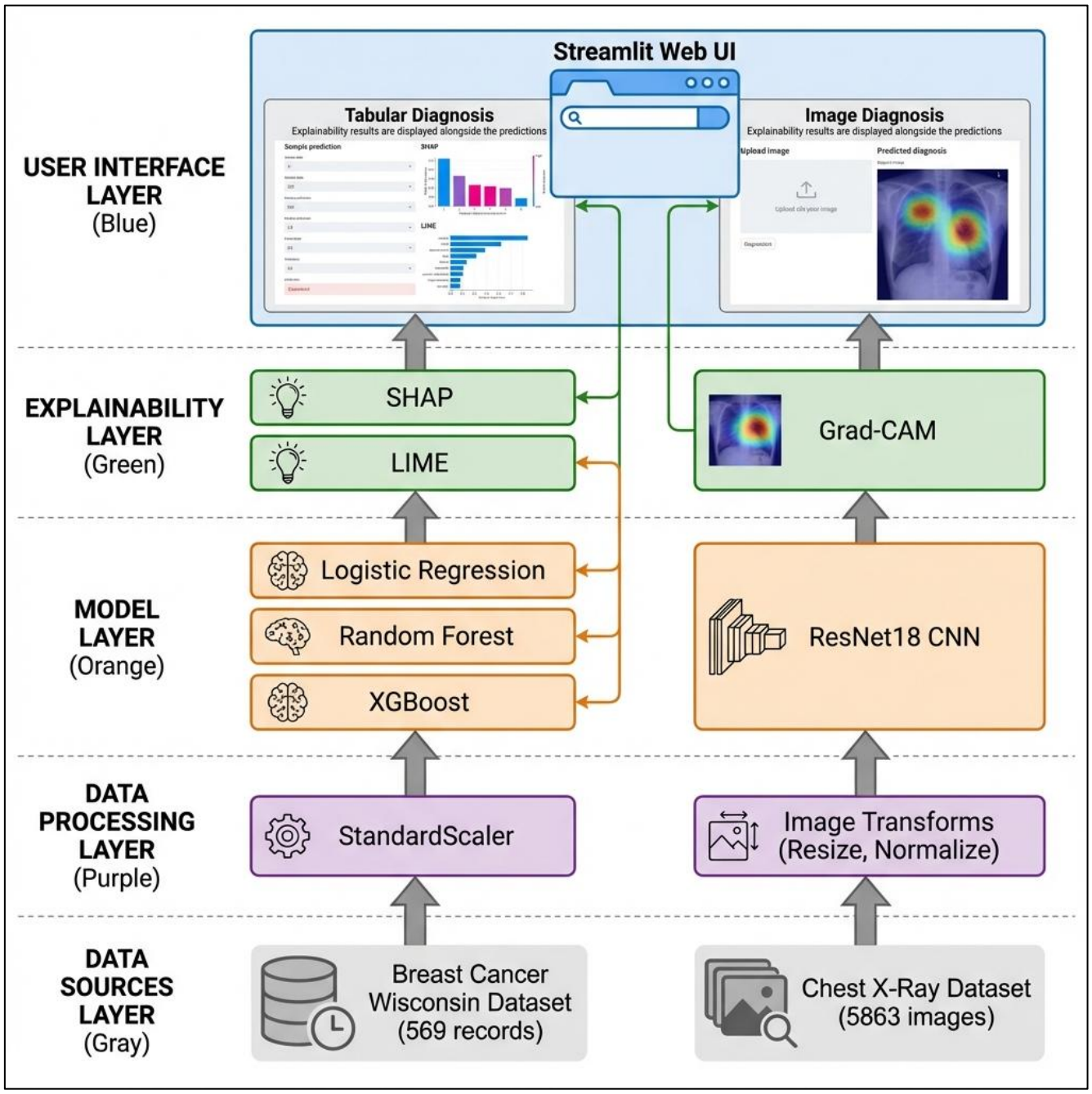

Рисунок 3.2 – Архітектура розробленої системи

Модуль даних (Data Layer) відповідає за уніфікацію вхідних даних. Для табличних даних (Breast Cancer dataset) реалізовано автоматичне видалення технічних стовпців (ID) та кодування цільової змінної. Для зображень (Chest X–Ray) створено клас ImageLoader, який виконує попередню обробку: ресайзінг до стандартного розміру 224x224 пікселів (вимога архітектури ResNet) та нормалізацію каналів RGB з використанням середніх значень та стандартних відхилень ImageNet.

Трансформація зображень для ResNet (src/image_loader.py):

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
    std=[0.229, 0.224, 0.225])
])
```

Система підтримує мультимодальність через використання спеціалізованих моделей для кожного типу даних:

– табличні моделі: логістична регресія (як базова інтерпретована модель), Випадковий ліс та XGBoost (як високоточні "чорні скриньки");

– модель комп'ютерного зору: згорткова нейронна мережа ResNet18. Використано підхід Transfer Learning: модель, попередньо навчена на мільйонах зображень ImageNet, була донавчена на специфічному датасеті пневмонії.

Останній повнозв'язний шар мережі було замінено на новий, що має лише два виходи (NORMAL, PNEUMONIA).

Ініціалізація та модифікація ResNet18 (src/train_image_model.py):

```
# Initialize ResNet18 with pretrained weights
model = models.resnet18(pretrained=True)
# Fine–tune: Reset final fully connected layer for 2 classes
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2) # Classes: NORMAL, PNEUMONIA
```

Рівень пояснень (Explanation Layer) – це ядро системи, де реалізовано логіку XAI. Модуль містить фабричні методи (Factory Pattern) для створення відповідних експлейнерів. Наприклад, функція get_shap_explainer автоматично визначає тип переданої моделі та ініціалізує або LinearExplainer (для регресії), або TreeExplainer (для бустингу). Це приховує складність реалізації від кінцевого користувача.

Фабричний метод для вибору SHAP Explainer (src/xai_utils.py):

```
def get_shap_explainer(model, X_train):
    model_type = type(model).__name__

    if model_type in ['RandomForestClassifier', 'XGBClassifier']:
        return shap.TreeExplainer(model)
    elif model_type == 'LogisticRegression':
        return shap.LinearExplainer(model, X_train)
```

```
        else:
            # Fallback to KernelExplainer (model–agnostic)
            return shap.KernelExplainer(model.predict_proba, shap.kmeans(X_train, 10))
```

Рівень представлення (Presentation Layer) реалізований у модулі app.py. Він забезпечує візуалізацію результатів у зрозумілому для лікаря форматі: ймовірності діагнозів відображаються у відсотках, важливість ознак – у вигляді інтерактивних графіків (Waterfall plots), а зони уваги на знімках – як напівпрозорі теплові карти.

При роботі з медичними даними (HIPAA, GDPR) критичним є видалення персональної інформації (PII). У нашій системі реалізовано механізм "на льоту" (On–the–fly de–identification).

1. Табличні дані: при завантаженні CSV автоматично ігноруються стовпці, що можуть містити ID пацієнта, ім'я або дату народження, якщо вони не використовуються як ознаки (вік залишається, дата народження – ні).

2. DICOM зображення: медичні зображення часто зберігаються у форматі DICOM, який містить метадані (теги) з іменем пацієнта. Наш клас ImageLoader працює з конвертованими JPEG/PNG зображеннями, де метадані вже видалені.

3. Алгоритм: при конвертації DICOM –> Pixel Array ми відкидаємо заголовок файлу, залишаючи лише матрицю інтенсивності пікселів.

4. Хешування: для внутрішнього відстеження сесії використовується хеш–сума зображення (SHA–256), а не ім'я файлу, що унеможливлює відновлення особи пацієнта за логами системи.

3.3 Реалізація алгоритмів інтерпретації

Основним викликом при реалізації було забезпечення коректної роботи алгоритмів з різними типами даних.

Для забезпечення гнучкості та розширюваності коду використано класичні патерни проектування (GoF).

1. Патерн Strategy: використано для вибору алгоритму класифікації. Інтерфейс ModelStrategy визначає методи train() та predict(). Конкретні класи LogisticRegressionStrategy, RandomForestStrategy реалізують цей інтерфейс. Це дозволяє змінювати модель у app.py одним рядком коду без зміни логіки інтерфейсу.

2. Патерн Factory Method: реалізовано у функції get_shap_explainer(). Клієнтський код не знає, який саме Explainer (Tree, Linear, Kernel) буде створено – він просто запитує "пояснювач" для переданої моделі. Фабрика інкапсулює логіку вибору (if/else) всередині себе.

3. Патерн Singleton: використано через декоратор @st.cache_resource у Streamlit. Завантаження моделі ResNet (50MB+) – це "дорога" операція. Singleton гарантує, що модель завантажується в пам'ять лише один раз при старті сервера, а всі сесії користувачів використовують цей єдиний екземпляр. Це критично для продуктивності веб–додатку.

Оскільки система працює з гетерогенним набором моделей (лінійна регресія, ансамблі дерев), використання єдиного типу "пояснювача" (Explainer) є неможливим з точки зору обчислювальної ефективності. Для вирішення цієї проблеми було реалізовано патерн "Фабричний метод" (Factory Method).

Реалізація SHAP базується на адитивній властивості: прогноз моделі розкладається на суму внесків кожної ознаки. Для інтеграції SHAP було розроблено універсальну функцію get_shap_explainer, яка автоматично визначає тип моделі (деревоподібна або лінійна) та обирає відповідний експейнер (TreeExplainer або LinearExplainer) (див. рис. 3.3). Це дозволяє оптимізувати обчислення:

```
    def get_shap_explainer(model, X_train):
        model_type = type(model).__name__

        if model_type in ['RandomForestClassifier', 'XGBClassifier']:
            # Використання TreeExplainer для деревоподібних моделей.
            # Це дозволяє обчислити точні значення Шеплі за поліноміальний час O(TLD^2).
            return shap.TreeExplainer(model)
        elif model_type == 'LogisticRegression':
```

```
                # Для лінійних моделей використовується LinearExplainer.
                # Він враховує аналітичні властивості лінійної комбінації ознак.
                return shap.LinearExplainer(model, X_train)
            else:
                # Fallback до KernelExplainer (model–agnostic метод).
                # Використовує апроксимацію через зважену лінійну регресію.
                # shap.kmeans використовується для зменшення вибірки фонових даних (background data),
                # що критично для швидкодії.
                return shap.KernelExplainer(model.predict_proba, shap.kmeans(X_train, 10))

        def calculate_shap_values(explainer, X_instance):
            """
            Calculates SHAP values for a single instance or a batch.
            """
            shap_values = explainer.shap_values(X_instance)

            # Обробка різних типів значень, що повертає бібліотека SHAP
            if isinstance(shap_values, list):
                # Для бінарної класифікації SHAP часто повертає список з двох масивів
                # (для класу 0 та класу 1). Нас цікавить позитивний клас (Malignant).
                return shap_values[1]
            return shap_values
```

Алгоритмічний опис наведено нижче.

1. Функція перевіряє ім'я класу моделі (type(model).__name__).

2. Якщо виявлено RandomForestClassifier або XGBClassifier, ініціалізується TreeExplainer. Цей алгоритм використовує структуру дерев рішень для точного обчислення внеску ознак, не перебираючи всі можливі коаліції, що робить його в тисячі разів швидшим за класичний метод Шеплі.

3. Для LogisticRegression застосовується LinearExplainer, який розраховує внесок на основі коефіцієнтів регресії та коваріації ознак.

4. Для будь–яких інших моделей використовується KernelExplainer. Оскільки цей метод є обчислювально важким, застосовується кластеризація k–means (shap.kmeans(X_train, 10)), щоб звести тисячі фонових прикладів до 10 репрезентативних центроїдів, значно прискорюючи генерацію пояснень у реальному часі (див. рис. 3.4).

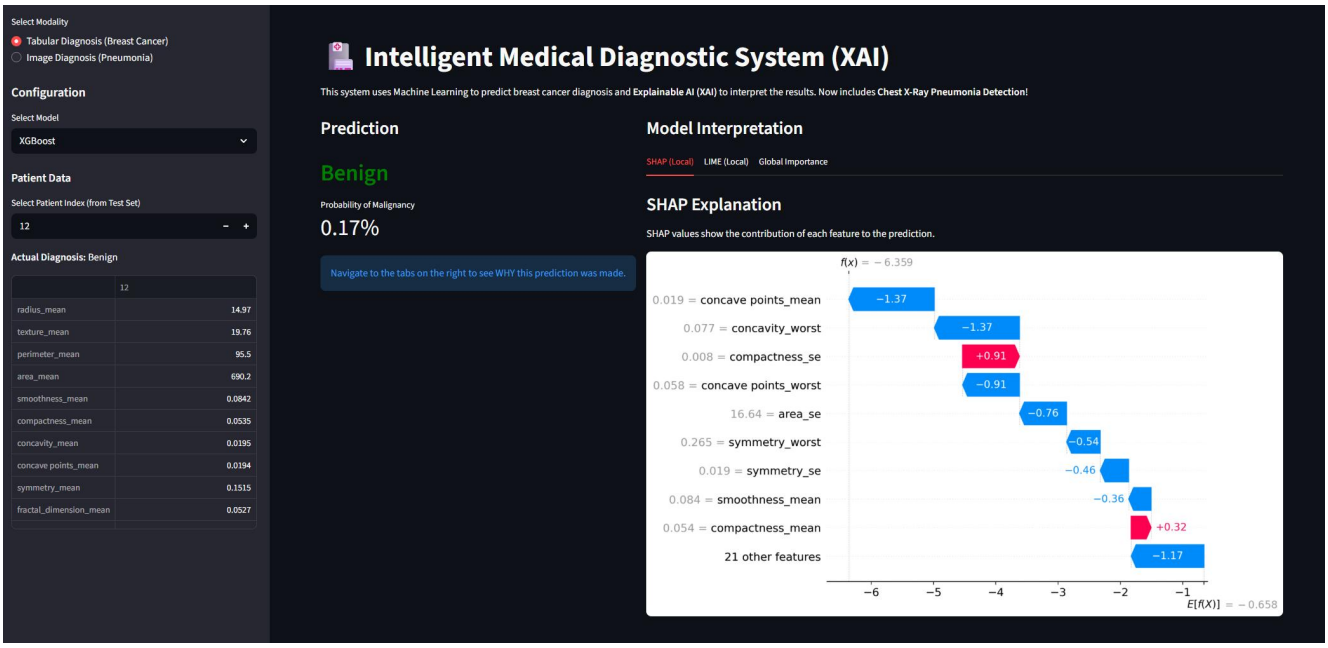

Рисунок 3.3 – Приклад візуалізації SHAP Waterfall Plot

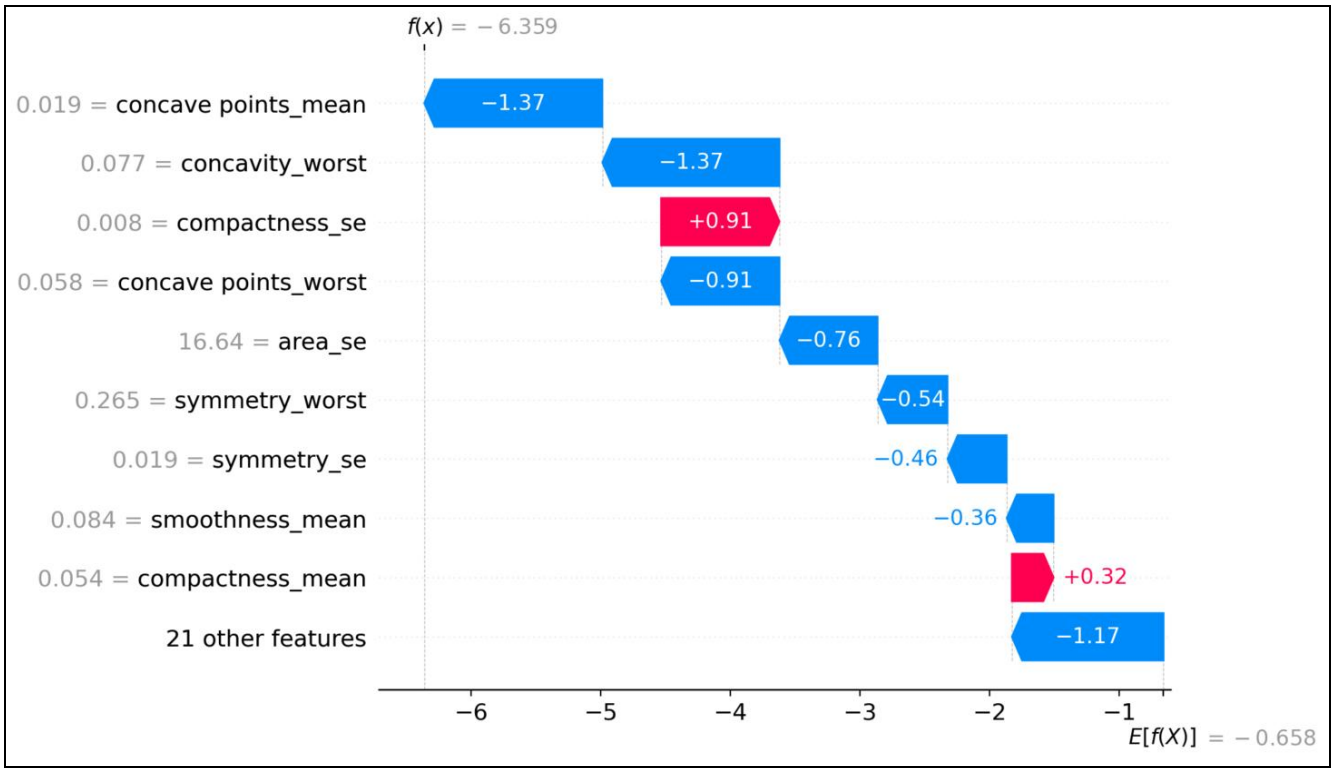

Рисунок 3.4 – Приклад візуалізації SHAP Waterfall Plot

Метод LIME (Local Interpretable Model–agnostic Explanations) реалізовано для створення локальних сурогатних моделей. Це дозволяє пояснити, чому модель прийняла рішення для конкретного пацієнта, шляхом перевірки стійкості прогнозу до малих змін у вхідних даних (див. рис. 3.5).

```
def get_lime_explainer(X_train, feature_names, class_names=['Benign', 'Malignant']):
```

```
        """
        Initializes a LIME Tabular Explainer.
        """
        # Створення об'єкта, що зберігає статистику навчальної вибірки
        # (середні значення, дисперсії) для коректної генерації пертурбацій.
        explainer = lime.lime_tabular.LimeTabularExplainer(
            training_data=np.array(X_train),
            feature_names=feature_names,
            class_names=class_names,
            mode='classification'
        )
        return explainer

    def explain_instance_lime(explainer, instance, model_predict_proba):
        """
        Generates a LIME explanation for a single instance.
        """
        # Запуск процесу генерації пояснення:
        # 1. Генерація N випадкових збурень навколо instance.
        # 2. Отримання прогнозів "чорної скриньки" через predict_fn.
        # 3. Навчання зваженої лінійної моделі.
        exp = explainer.explain_instance(
            data_row=np.array(instance),
            predict_fn=model_predict_proba
        )
        return exp
```

Алгоритмічний опис наведено нижче.

1. Ініціалізація (LimeTabularExplainer): на цьому етапі система дискретизує безперервні ознаки (наприклад, розбиває "радіус пухлини" на квартилі), що необхідно для коректної генерації збурень.

2. Семплювання (Sampling): функція explain_instance генерує набір штучних даних (наприклад, 5000 зразків) навколо точки інтересу instance, використовуючи нормальний розподіл зі статистичними параметрами навчальної вибірки.

3. Маркування (Labeling): згенеровані зразки подаються на вхід методу predict_proba основної моделі (наприклад, XGBoost), щоб отримати ймовірності класів.

4. Зважування та навчання: обчислюється відстань (експоненційне ядро) між оригінальним пацієнтом та згенерованими зразками. Навчається проста лінійна модель (Lasso–регресія), яка намагається відтворити прогнози складної

моделі, але лише в локальному околі даного пацієнта. Коефіцієнти цієї лінійної моделі стають поясненням.

Для LIME реалізовано функцію, яка виконує наступні кроки:

– генерує N випадкових збурень навколо обраного пацієнта;

– отримує прогнози чорної скриньки для цих збурень;

– зважує збурення за їх близькістю до оригінального прикладу (використовуючи експоненційне ядро відстані);

– навчає зважену лінійну регресію (Lasso), коефіцієнти якої стають поясненням.

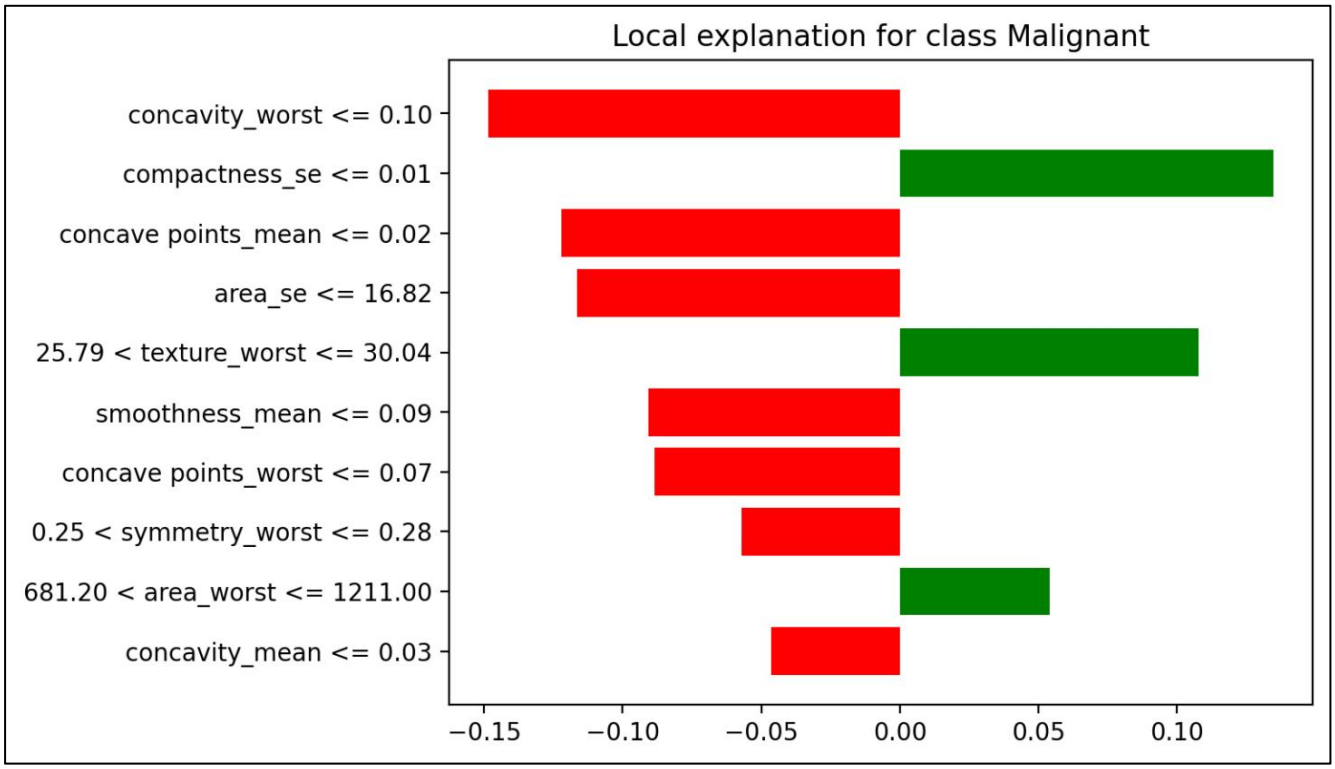

Рисунок 3.5 – Приклад візуалізації LIME

Найскладнішою частиною системи є модуль інтерпретації згорткових нейронних мереж. PyTorch, на відміну від TensorFlow, використовує динамічний обчислювальний граф, який автоматично очищує проміжні градієнти для економії пам'яті. Для реалізації Grad–CAM необхідно "перехопити" ці дані. Це досягається використанням механізму хуків (Hooks).

Клас GradCAM інкапсулює логіку реєстрації хуків та виконання прямого і

зворотного проходів (див. рис. 3.6).

Процес реалізації наведено нижче.

1. Реєстрація хуків (Hooks): ми "підключаємося" до шару нейромережі, щоб перехопити дані під час проходу (Forward) та зворотного поширення помилки (Backward).

Реєстрація хуків у класі GradCAM (src/gradcam_utils.py)

```
def __init__(self, model, target_layer):
    self.model = model
    self.target_layer = target_layer

    # Hook for gradients (Backward pass)
    target_layer.register_backward_hook(self.save_gradient)
    # Hook for activations (Forward pass)
    target_layer.register_forward_hook(self.save_activation)
```

2. Обчислення ваг: глобальне середнє значення градієнтів для кожної карти ознак.

3. Генерація карти: зважена сума карт ознак та застосування функції активації ReLU, щоб залишити лише ті пікселі, які мають позитивний вплив на клас (нас цікавлять ознаки, що підтверджують діагноз, а не заперечують його).

Генерація теплової карти (src/gradcam_utils.py):

```
# Global Average Pooling of gradients
weights = np.mean(gradients, axis=(1, 2))
# Weighted combination of activations
cam = np.zeros(activations.shape[1:], dtype=np.float32)
for i, w in enumerate(weights):
    cam += w * activations[i]

# ReLU Activation (keep only positive influence)
cam = np.maximum(cam, 0)
```

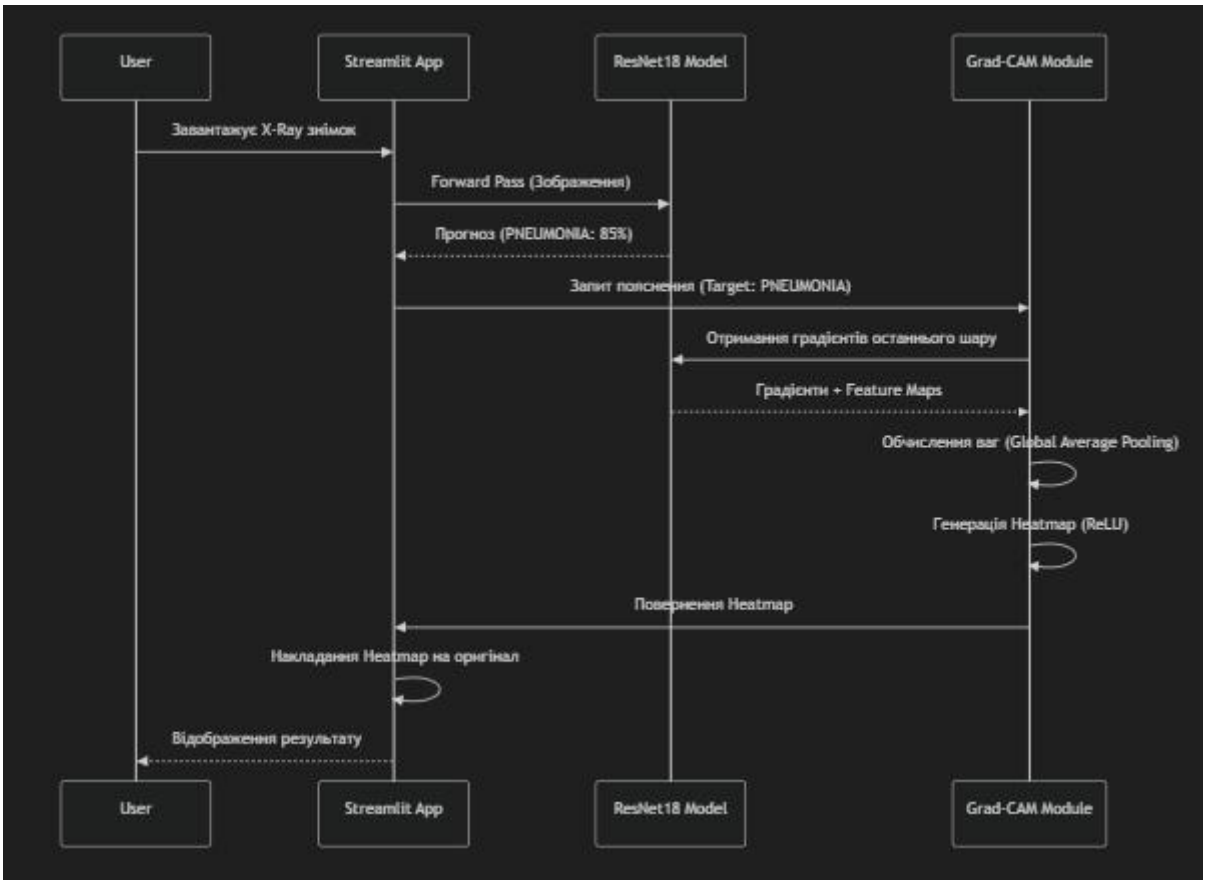

Рисунок 3.6 – Схема роботи алгоритму Grad–CAM

Отримана карта масштабується до розміру оригінального зображення (224x224 або вище) та накладається на нього.

Детальний опис алгоритму представлено нижче.

1. Реєстрація хуків: у конструкторі __init__ ми підписуємося на події шару layer4 мережі ResNet. Метод save_activation зберігає вихід шару (тензор розміром 512×7×7), а save_gradient зберігає градієнти.

2. Зворотне поширення (Backpropagation): ключовим моментом є рядки one_hot[0][class_idx] = 1 та output.backward(). Ми штучно кажемо мережі: "Уяви, що ми хочемо максимізувати впевненість саме у цьому класі (наприклад, Пневмонія), і скажи, які нейрони останнього згорткового шару на це впливають найбільше".

3. Зважена комбінація та ReLU: ми множимо кожну карту ознак на її вагу і сумуємо їх. Функція np.maximum(cam, 0) відкидає негативні значення, оскільки негативні градієнти означають зони, що зменшують ймовірність діагнозу, а нас цікавлять лише підтверджуючі фактори (Discriminative localization).

Отриману матрицю Grad–CAM необхідно візуалізувати так, щоб лікар міг зіставити її з анатомією пацієнта. Функція overlay_cam виконує злиття зображень.

```
def overlay_cam(original_img, cam, alpha=0.5):
    """
    Overlays the CAM heatmap on the original image.
    Args:
        original_img: Numpy array (H, W, 3) in range [0, 1] or [0, 255].
        cam: Numpy array (224, 224) heatmap in range [0, 1].
    """
    # Конвертація вхідного зображення до формату float32 [0, 1]
    img = original_img.astype(np.float32)
    if img.max() > 1:
        img /= 255.0

    # Гарантуємо відповідність розмірів
    h, w = img.shape[:2]
    cam_resized = cv2.resize(cam, (w, h))

    # Генерація кольорової карти (Heatmap)
    # Перетворення одноканальної матриці (Grayscale) у 3–канальну (RGB)
    # cv2.COLORMAP_JET створює градієнт від синього (низька увага) до червоного (висока)
    heatmap = cv2.applyColorMap(np.uint8(255 * cam_resized), cv2.COLORMAP_JET)
    heatmap = np.float32(heatmap) / 255
    heatmap = heatmap[..., ::–1] # Конвертація BGR (OpenCV default) to RGB

    # Альфа–змішування (Alpha Blending)
    # Формула: Output = alpha * Heatmap + (1 – alpha) * Original
    overlayed = heatmap * alpha + img * (1 – alpha)

    # Кліппінг значень для уникнення артефактів переповнення
    overlayed = np.clip(overlayed, 0, 1)

    return overlayed
```

Зображення приводяться до єдиного діапазону [0,1] для коректного математичного додавання.

Сира карта Grad–CAM є відтінками сірого. Функція cv2.applyColorMap перетворює інтенсивність пікселя на колір. Обрана палітра JET є стандартом у медичній візуалізації, оскільки забезпечує високий контраст між "гарячими" (патологія) та "холодними" (фон) зонами.

Використовується лінійна інтерполяція між оригінальним рентгенівським знімком та тепловою картою. Параметр alpha=0.5 забезпечує напівпрозорість,

дозволяючи лікарю бачити кісткові структури та тканини під шаром теплової карти.

Інтерфейс реалізовано з використанням бібліотеки Streamlit. Він розділений на дві логічні частини залежно від типу діагностики.

Функціональні можливості:

– вибір моделі, завантаження файлів, вибір пацієнта з тестової вибірки;

– ідображення прогнозу (діагноз та ймовірність) та візуалізація пояснень (вкладки "SHAP", "LIME", "Grad–CAM");

– можливість зміни цільового класу для пояснення (наприклад, "Чому це НЕ норма?").

Для отримання доступу до внутрішніх градієнтів згорткової нейронної мережі (CNN) без зміни її архітектури використано механізм "хуків" (hooks) бібліотеки PyTorch.

```
class GradCAM:
def __init__(self, model, target_layer):
        self.model = model
        self.target_layer = target_layer
        self.gradients = None
        self.activations = None
        # Реєстрація хука для зворотного проходу (backward pass)
        # Це дозволяє перехопити градієнти, що проходять через цільовий шар
        target_layer.register_backward_hook(self.save_gradient)
        # Реєстрація хука для прямого проходу (forward pass)
        # Це дозволяє зберегти карти ознак (feature maps) на виході шару
        target_layer.register_forward_hook(self.save_activation)
def save_gradient(self, module, grad_input, grad_output):
        # Збереження градієнтів (похідних функції втрат по активаціях)
        self.gradients = grad_output[0]
def save_activation(self, module, input, output):
        # Збереження самих активацій (feature maps)
        self.activations = output
```

У конструкторі приймається модель та цільовий шар ("target_layer"), який зазвичай є останнім згортковим шаром мережі (наприклад, "layer4" у ResNet18), оскільки саме він містить найбільш високорівневі просторові ознаки.

Методи "register_backward_hook" та "register_forward_hook" є критично важливими, оскільки PyTorch автоматично звільняє пам'ять від проміжних

градієнтів під час інференсу для економії ресурсів. Ці методи дозволяють примусово зберегти необхідні дані у змінних "self.gradients" та "self.activations" для подальших обчислень.

Основна логіка методу реалізована у магічному методі __call__, який виконує прямий і зворотний проходи та математичний розрахунок карти важливості.

```
def __call__(self, x, class_idx=None):
        output = self.model(x)

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

        # Zero grads
        self.model.zero_grad()

        # Backward pass
        one_hot = torch.zeros_like(output)
        one_hot[0][class_idx] = 1
        output.backward(gradient=one_hot, retain_graph=True)

        # Get gradients and activations
        gradients = self.gradients.data.cpu().numpy()[0]
        activations = self.activations.data.cpu().numpy()[0]

        # Global Average Pooling of gradients
        weights = np.mean(gradients, axis=(1, 2))

        # Weighted combination of activations
        cam = np.zeros(activations.shape[1:], dtype=np.float32)
        for i, w in enumerate(weights):
            cam += w * activations[i]

        # ReLU
        cam = np.maximum(cam, 0)

        # Resize to input image size (224x224)
        cam = cv2.resize(cam, (x.shape[2], x.shape[3]))

        # Normalize
        cam = cam – np.min(cam)
        cam = cam / (np.max(cam) + 1e–8)

        return cam
```

Детальний опис алгоритму наведено нижче.

1. Ключовим моментом є рядок "output.backward(gradient=one_hot)". Ми

штучно кажемо мережі, що ми хочемо максимізувати впевненість саме у класі "class_idx", і просимо обчислити, які нейрони на це впливають.

2. Рядок "weights = np.mean(gradients, axis=(1, 2))" реалізує формулу Grad–CAM для обчислення важливості α_k^c кожного k–го фільтра. Ми усереднюємо градієнти по всій площі зображення.

3. Цикл "for" обчислює лінійну комбінацію карт активацій. Карти, які мають великий позитивний градієнт (важливі для класу), додаються з великою вагою.

4. ReLU ("np.maximum(cam, 0)"): Це критичний крок. Ми відкидаємо пікселі, які мають *від'ємний* вплив на клас (тобто ті, наявність яких зменшує впевненість моделі). Нас цікавлять тільки ті регіони, які *підтверджують* наявність хвороби.

5. Останній блок коду приводить значення карти до діапазону $[0, 1]$, щоб її можна було візуалізувати як зображення.

```
def overlay_cam(original_img, cam, alpha=0.5):
    """
    Overlays the CAM heatmap on the original image.
    Args:
        original_img: Numpy array (H, W, 3) in range [0, 1] or [0, 255].
        cam: Numpy array (224, 224) heatmap in range [0, 1].
    """
    # Ensure original_img is float [0, 1]
    img = original_img.astype(np.float32)
    if img.max() > 1:
        img /= 255.0

    h, w = img.shape[:2]
    cam_resized = cv2.resize(cam, (w, h))

    # Heatmap
    heatmap = cv2.applyColorMap(np.uint8(255 * cam_resized), cv2.COLORMAP_JET)
    heatmap = np.float32(heatmap) / 255
    heatmap = heatmap[..., ::–1] # BGR to RGB

    # Overlay
    overlayed = heatmap * alpha + img * (1 – alpha)
    overlayed = np.clip(overlayed, 0, 1)

    return overlayed
```

Використання "cv2.COLORMAP_JET" є стандартом у науковій візуалізації.

Синій колір відповідає низьким значенням (0, неважливі зони), а червоний – високим (1, зони інтересу).

Формула "heatmap * alpha + img * (1 – alpha)" дозволяє накласти теплову карту поверх рентгену так, щоб лікар бачив анатомічні структури крізь кольорову "підсвітку". Це критично для медичної верифікації (щоб бачити, що саме підсвічено).

3.4 Опис інтерфейсу користувача та UX/UI рішень

Розроблений веб–інтерфейс базується на принципах "Чистого дизайну" (Clean Design) та "Прогресивного розкриття інформації" (Progressive Disclosure). Головна мета – надати лікарю миттєвий діагностичний прогноз, не перевантажуючи його зайвими деталями, але залишаючи можливість заглибитися у пояснення в один клік.

Інтерфейс реалізовано у темній темі (Dark Mode), що є стандартом для радіологічного програмного забезпечення, оскільки це знижує навантаження на очі при роботі у затемнених кабінетах діагностики (див. рис. 3.7).

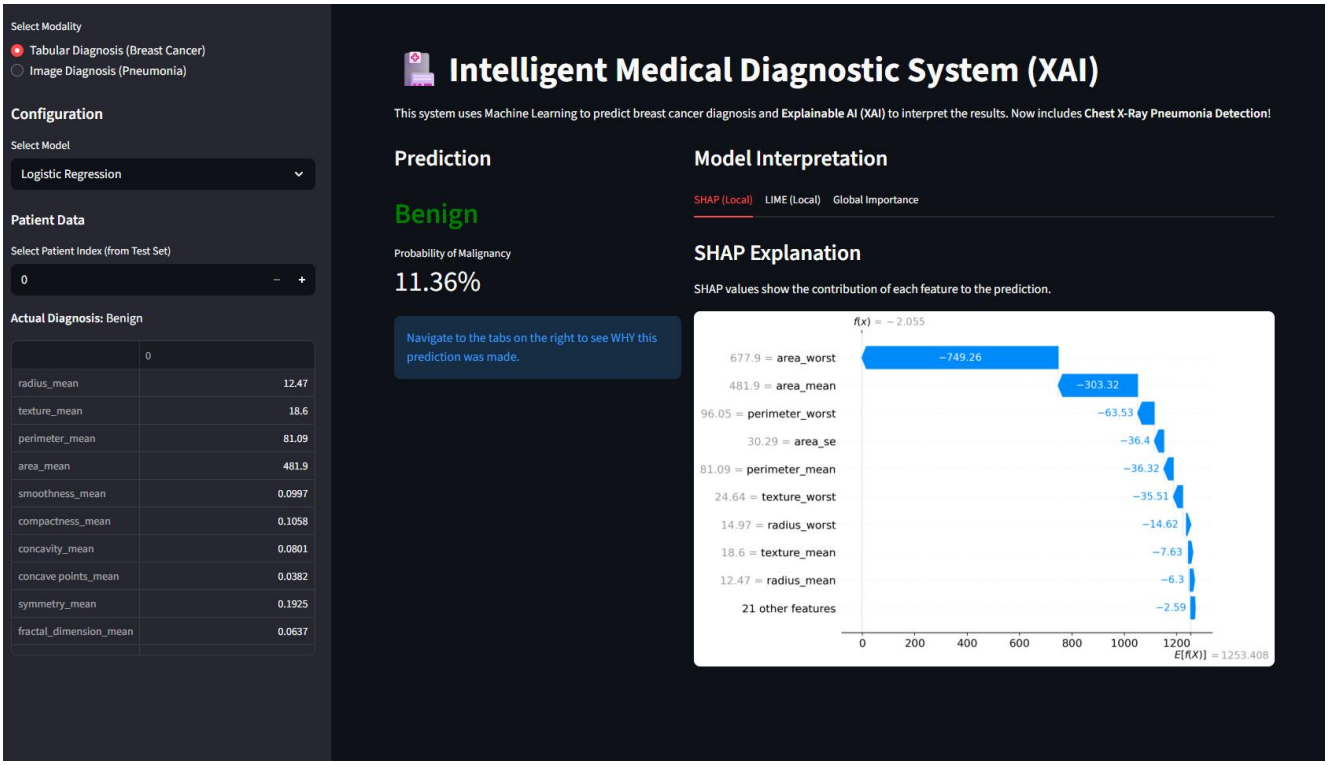

Рисунок 3.7 – Загальний вигляд системи (режим табличних даних)

Інтерфейс чітко розділений на дві функціональні зони: Бічна панель налаштувань (Sidebar) та Основна робоча область (Main Canvas).

Бічна панель (Sidebar) реалізує патерн "Control Panel". Вона містить усі елементи керування вхідними даними, відокремлюючи їх від результатів аналізу.

Перемикач "Select Modality" дозволяє миттєво змінювати контекст роботи між аналізом табличних даних (Breast Cancer) та зображень (Pneumonia), не перезавантажуючи сторінку.

Випадаючий список дозволяє лікарю обирати "другу думку" від різних алгоритмів (Logistic Regression, Random Forest, XGBoost), порівнюючи їхні висновки (див. рис. 3.8).

Інтерактивний віджет вибору пацієнта з тестової вибірки дозволяє швидко переходити між кейсами. Нижче відображається таблиця з "сирими" значеннями ознак (радіус, текстура тощо), що забезпечує прозорість вхідних даних.

Найважливіша інформація – прогноз ("Benign") – виділена найбільшим шрифтом та кольоровим кодуванням.

Використано загальноприйняту медичну колірну схему: Зелений – норма/доброякісне, Червоний – патологія/злоякісне. Це дозволяє лікарю оцінити ситуацію за частку секунди (Preattentive Processing).

Під прогнозом великими цифрами виведено ймовірність (Probability of Malignancy: 11.36%). Це критично важливо для прийняття рішень у пограничних випадках.

Для реалізації концепції XAI використано вкладки (Tabs), що дозволяє перемикатися між різними методами інтерпретації без скролінгу сторінки.

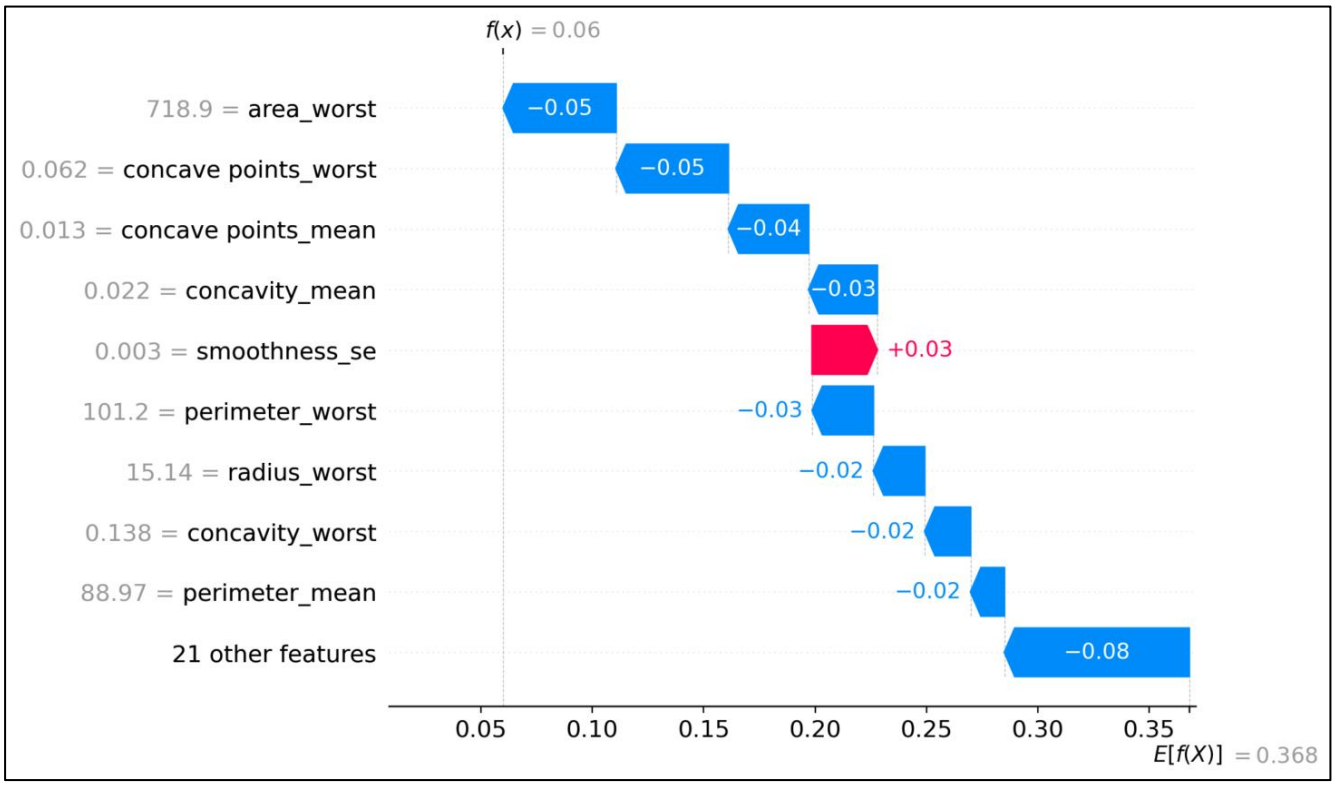

Рисунок 3.8 – Локальне пояснення SHAP (Waterfall Plot)

Графік SHAP Waterfall є основним інструментом аналізу причинно–наслідкових зв'язків:

Графік показує, як кожен окремий показник пацієнта "штовхає" базове передбачення моделі (E[f(x)]) в бік доброякісного (сині смуги ліворуч) або злоякісного (червоні смуги праворуч) діагнозу.

Значення ознак (наприклад, area_worst = 677.9) виведені зліва від смуг, що дозволяє лікарю одразу співставити клінічний показник з його впливом. У наведеному прикладі видно, що низькі значення area_worst та area_mean є головними факторами, чому модель вирішила, що пухлина доброякісна.

На відміну від математично точного SHAP, LIME показує простішу картину у вигляді правил "ЯКЩО–ТО" (див. рис. 3.9).

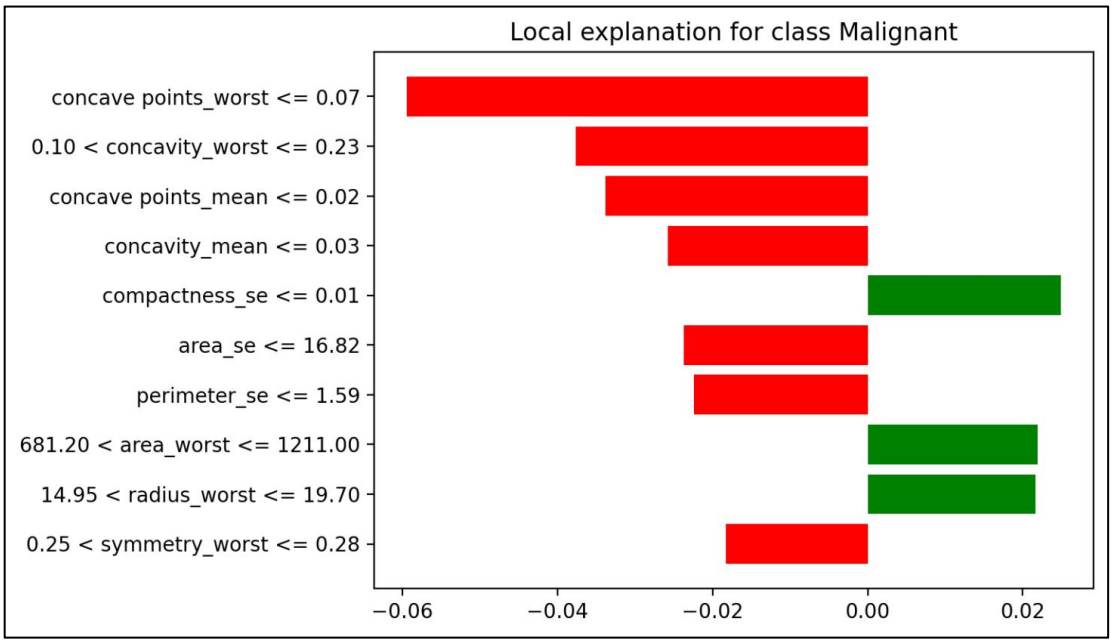

Рисунок 3.9 – Локальне пояснення LIME

Зелені смуги вказують на ознаки, що підтримують клас "Benign", червоні – "Malignant". Це слугує механізмом перехресної перевірки (Cross–validation) пояснень для лікаря.

Вкладка "Global Importance" надає контекст роботи моделі в цілому. Це дозволяє верифікувати, чи спирається модель на медично обґрунтовані маркери (наприклад, area_worst, concave points), а не на випадкові шуми (див. рис. 3.10).

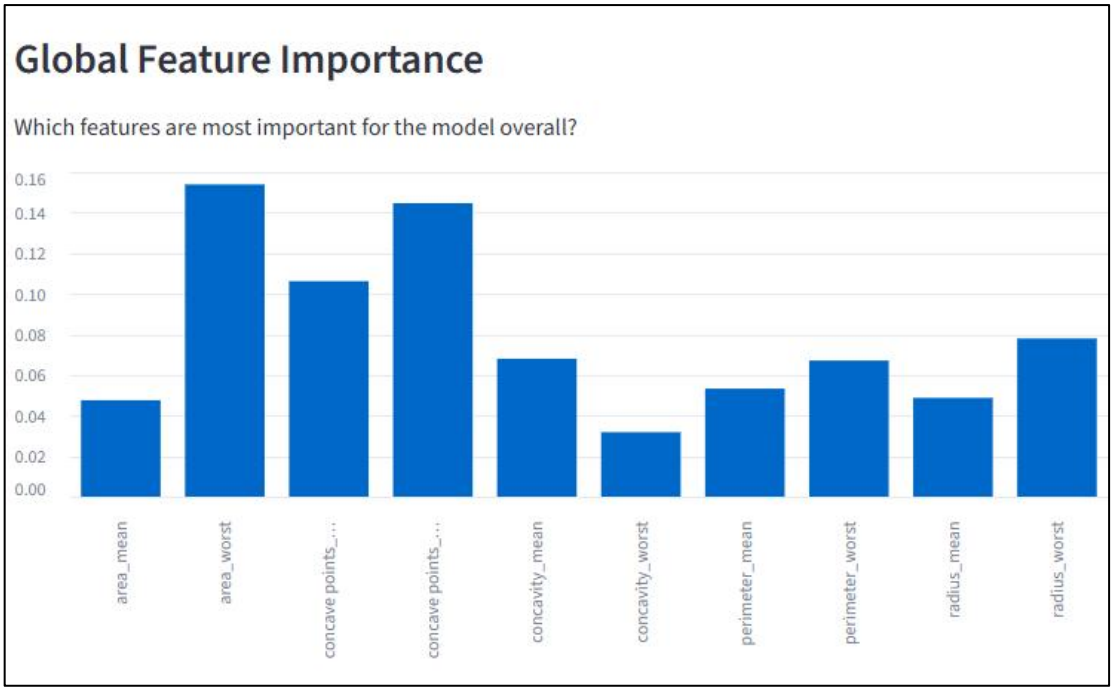

Рисунок 3.10 – Глобальна важливість ознак

При перемиканні в режим "Image Diagnosis" інтерфейс адаптується під візуальний контент (див. рис. 3.11).

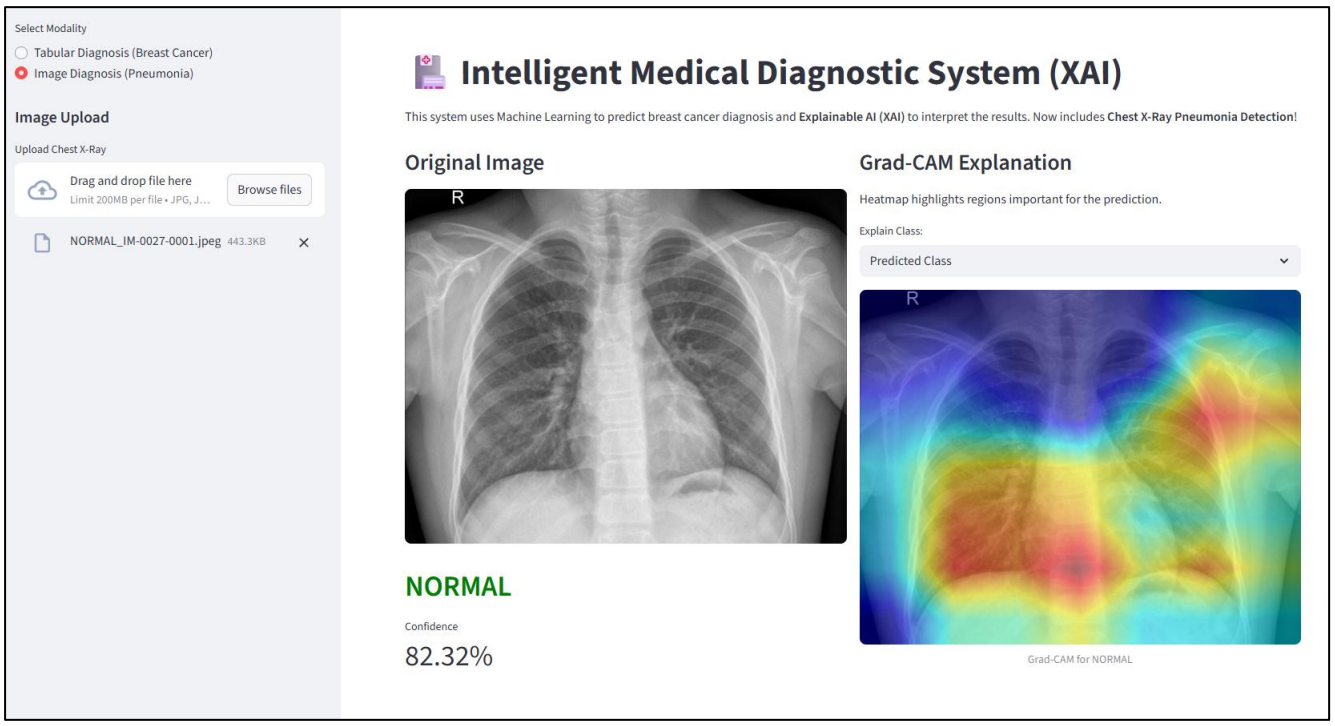

Рисунок 3.11 – Модуль завантаження та аналізу рентгенівських знімків

Реалізовано зону Drag-and-Drop, що спрощує роботу з файлами DICOM/JPEG.

Екран розділено на дві колонки. Зліва – Оригінальний знімок, справа – пояснення Grad–CAM.

Лікар завжди повинен бачити оригінал без артефактів обробки, щоб переконатися у якості знімка та відсутності технічних дефектів. Теплова карта праворуч виступає як "доповнена реальність".

На рис. 3.11 система підсвічує (червоним) області серця та діафрагми, вказуючи, що відсутність затемнень у легеневих полях стала причиною діагнозу "Норма".

На рис. 3.12 (Grad–CAM for PNEUMONIA) показано режим "What–if Analysis".

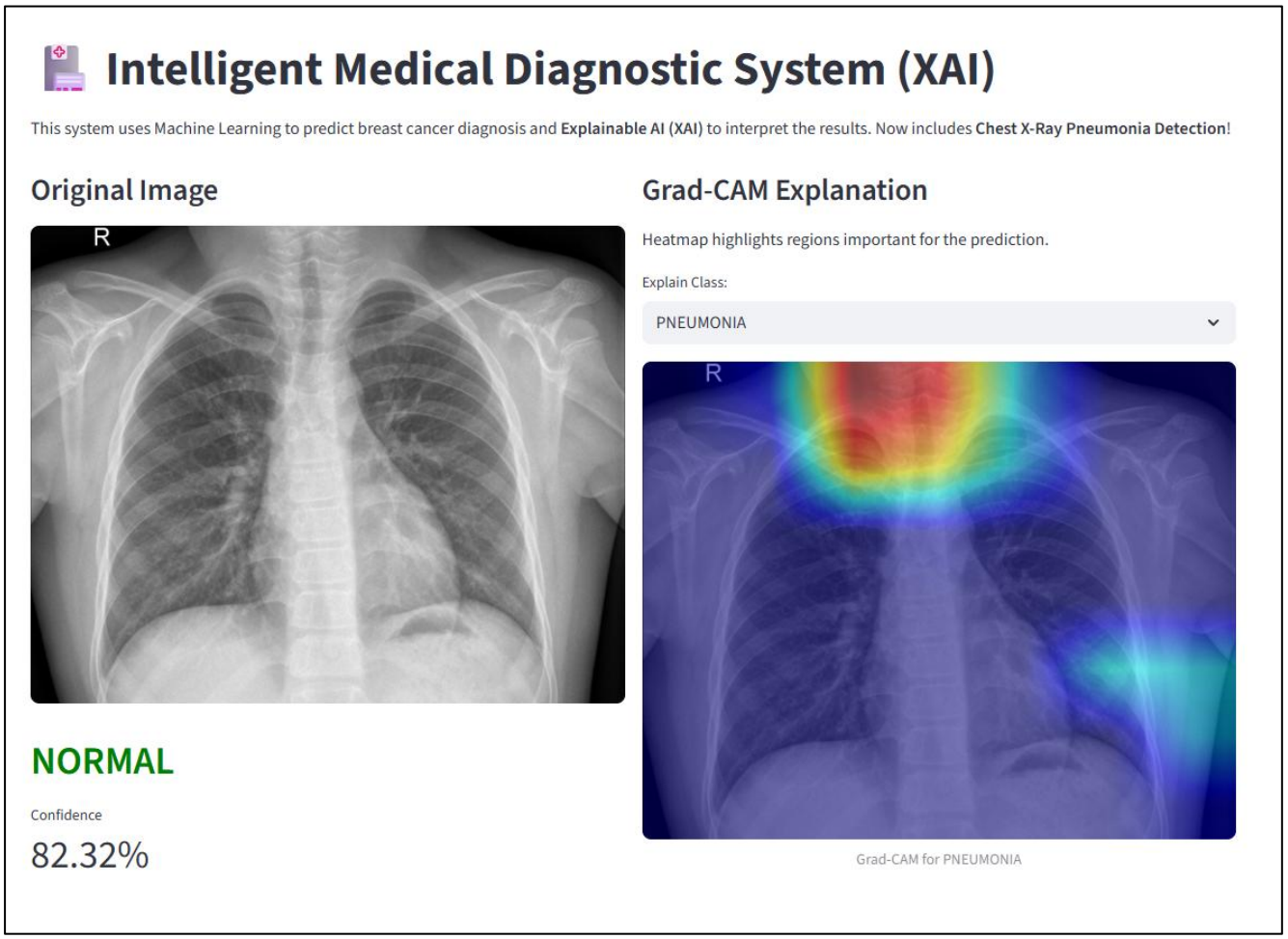

Рисунок 3.12 – Grad–CAM for PNEUMONIA

Через випадаючий список "Explain Class" лікар може запитати систему: "Чому (або де) ти могла б побачити пневмонію?". На скріншоті видно, що карта активності змістилася на верхні частки легень, хоча ймовірність цього класу низька. Це демонструє здатність системи до контрафактивних пояснень.

Як і в табличному режимі, діагноз "NORMAL" підсвічено зеленим, а впевненість (82.32%) виведена великим шрифтом для швидкого зчитування.

Запропонований інтерфейс успішно вирішує проблему "Чорної скриньки", перетворюючи складні тензорні обчислення на зрозумілі візуальні образи. Можливість інтерактивної взаємодії (вибір класу пояснення, перемикання методів) підвищує довіру лікаря до системи, перетворюючи її з "автоматичного оракула" на прозорого асистента.

3.5 Об'єктно–орієнтоване проектування та UML моделювання

Для формалізації процесу розробки та документування архітектури системи було використано уніфіковану мову моделювання UML (Unified Modeling Language). Нижче наведено опис розроблених діаграм. Use Case Diagram відображає взаємодію акторів з системою (див. рис. 3.13).

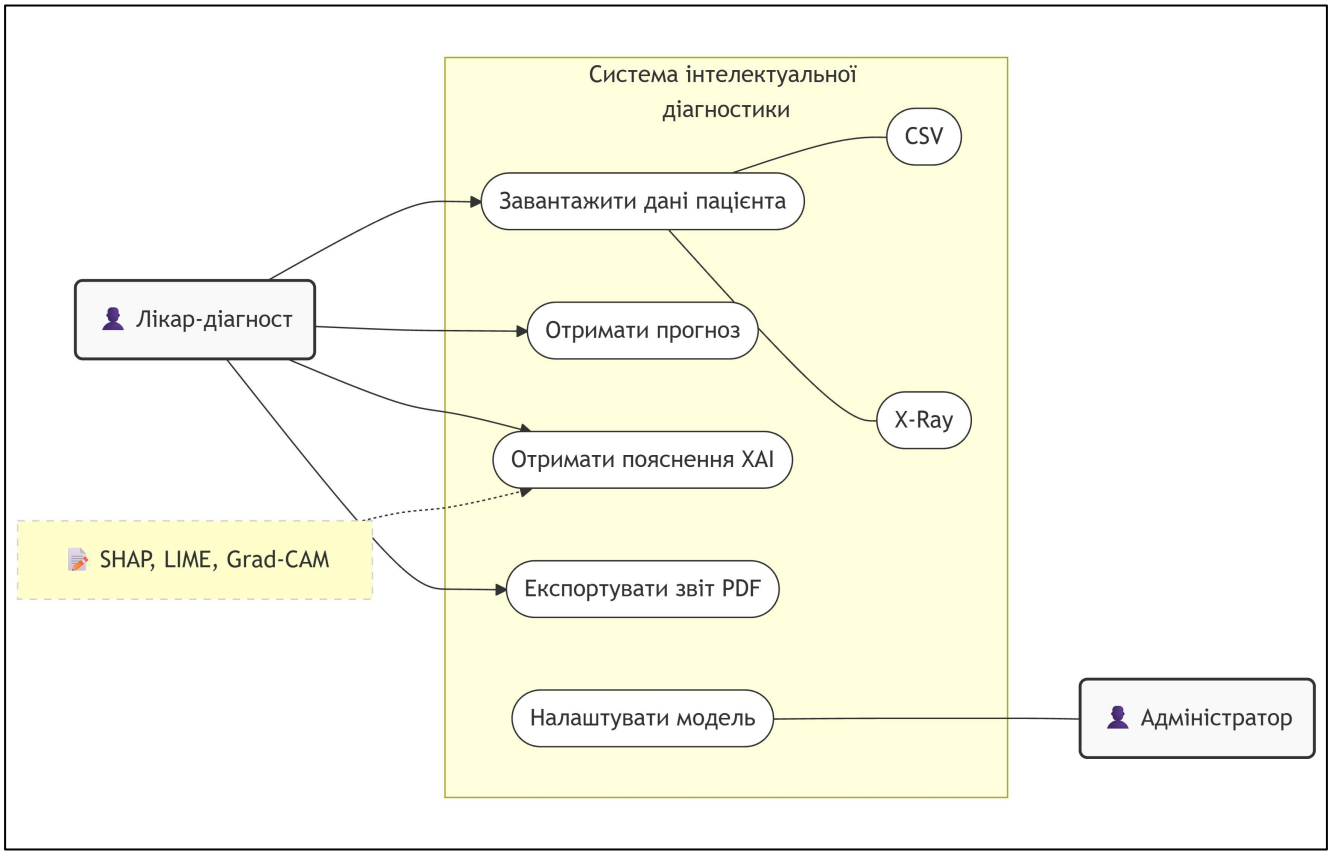

Рисунок 3.13 – Use Case Diagram

В архітектурі системи визначено двох ключових акторів, які взаємодіють із програмним комплексом. Основним користувачем виступає лікар–діагност, на потреби якого орієнтований головний функціонал інтерфейсу. Натомість технічний супровід здійснює адміністратор системи, який відповідає за підтримку актуальності та оновлення математичних моделей.

Функціональні вимоги до системи реалізовано через низку прецедентів, базовим з яких є завантаження даних пацієнта. Цей процес передбачає гнучкість і включає сценарії роботи як із табличними файлами формату CSV, так і з рентгенівськими зображеннями. Після обробки вхідної інформації активується

прецедент отримання прогнозу, в результаті чого система повертає передбачений клас захворювання разом із оцінкою ймовірності. Ключовим етапом взаємодії є запит на отримання пояснення, під час якого лікар має можливість обрати конкретний метод інтерпретації, зокрема SHAP, LIME або Grad–CAM.

Завершальним етапом аналізу є експорт звіту, що дозволяє зберегти отримані результати та візуалізації у зручному форматі PDF для подальшого використання. Окремий рівень доступу вимагає прецедент налаштування моделі, який доступний виключно адміністратору та дозволяє здійснювати вибір типу алгоритму для роботи, наприклад перемикання між Random Forest та Logistic Regression.

Діаграма діяльності детально описує алгоритм роботи лікаря, який розпочинається з ініціалізації сеансу та проходження обов'язкової процедури авторизації для входу в систему. Наступним кроком є вибір необхідної модальності, де користувач визначає тип вхідних даних для аналізу, обираючи між табличними даними та медичними зображеннями (див. рис. 3.14).

Далі процес розгалужується залежно від прийнятого рішення. У випадку вибору табличного шляху система завантажує файл у форматі CSV та виконує попередню обробку інформації, після чого лікар обирає конкретного пацієнта, а модуль XAI генерує пояснення за допомогою значень SHAP. Альтернативний шлях для роботи із зображеннями передбачає завантаження файлів JPEG та їх автоматичний ресайзінг, слідом за чим виконується інференс згорткової нейромережі та генерація візуалізацій методом Grad–CAM.

На завершальному етапі спеціаліст проводить ретельний аналіз наданих результатів та інтерпретацій. На основі цієї інформації відбувається прийняття остаточного клінічного рішення, де лікар може або підтвердити, або відхилити діагноз, запропонований штучним інтелектом. Після фіксації висновку робочий сеанс у системі завершується.

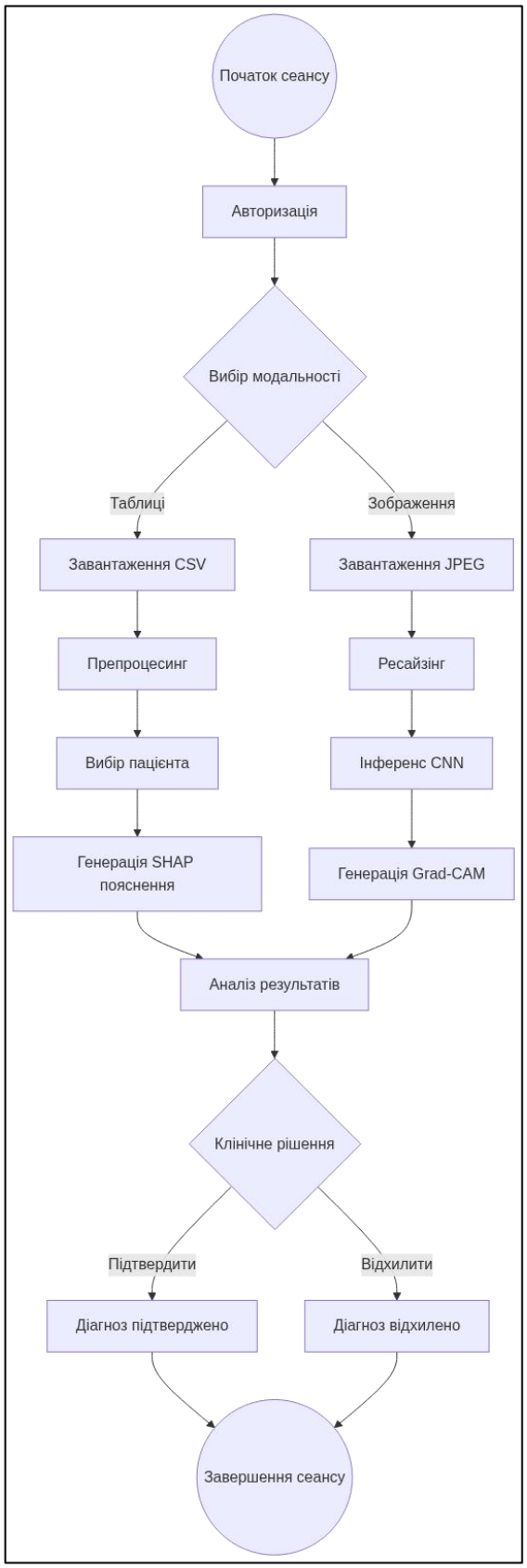

Рисунок 3.14 – Activity Diagram

Діаграма послідовності, представлена на рис. 3.15, деталізує технічний процес генерації пояснень методом Grad–CAM, що розпочинається із взаємодії

користувача з інтерфейсом на базі Streamlit для завантаження зображення. Після отримання вхідного файлу модуль UI звертається до компонента ImageLoader із запитом на виконання функції preprocess_image, у відповідь на що завантажувач повертає підготовлений тензор. На наступному етапі інтерфейс ініціює прямий прохід даних через модель ResNet шляхом виклику методу forward, отримуючи від нейромережі попередній прогноз.

Процес інтерпретації активується, коли UI ініціалізує об'єкт GradCAM, передаючи йому посилання на модель та вказівку на цільовий шар згортки. Далі алгоритм GradCAM реєструє необхідні хуки всередині архітектури та виконує зворотне поширення помилки backward виключно для цільового класу. У відповідь модель надає обчислені градієнти та карти активації, на основі яких формується теплова карта heatmap, що передається назад до інтерфейсу. Завершується цикл взаємодії тим, що система комбінує отриману карту з оригінальним знімком та відображає накладене зображення користувачеві для аналізу.

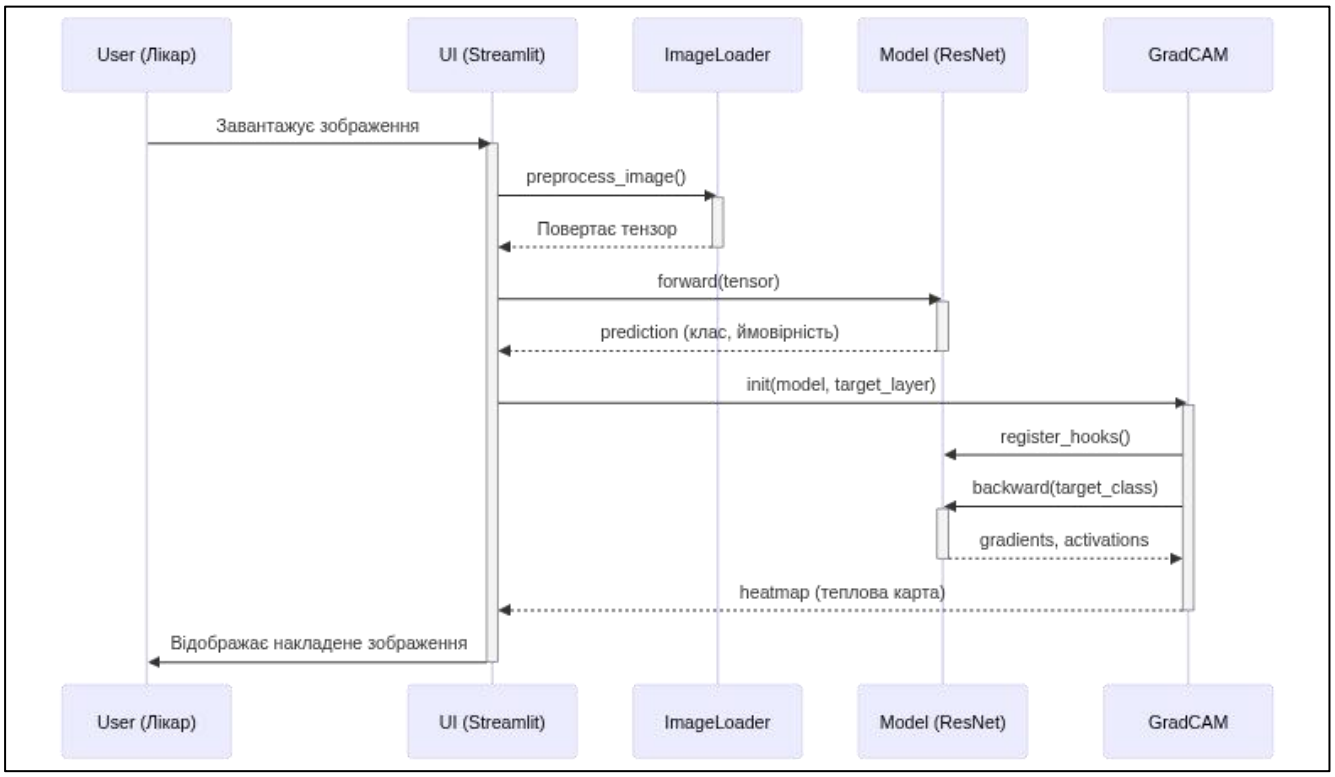

Рисунок 3.15 – Sequence Diagram

Діаграма розгортання, що представлена на рис. 3.16, відображає фізичну архітектуру системи та розподіл компонентів по апаратних вузлах. На стороні клієнта розташовується робоча станція лікаря Client Node, де взаємодія з програмним комплексом відбувається через сучасний веб–браузер на кшталт Chrome або Firefox, що не потребує встановлення додаткового спеціалізованого програмного забезпечення.

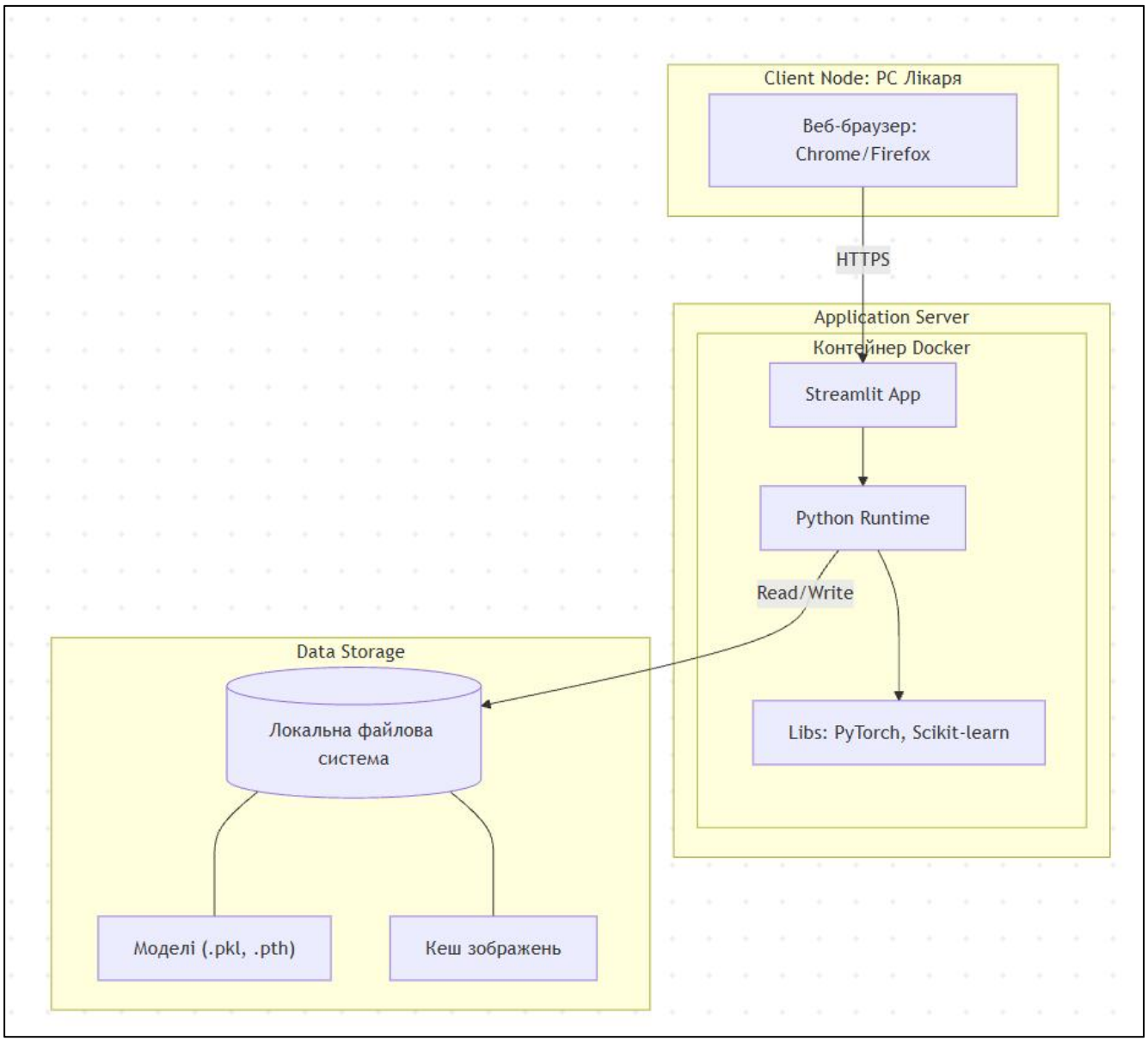

Рисунок 3.16 – Deployment Diagram

Діаграма потоків даних першого рівня, що представлена на рис. 3.17 як контекстна діаграма, демонструє загальну схему руху інформації в системі. На вхід цього процесу надходять «сирі» медичні дані, які підлягають обробці

центральним компонентом під назвою «Система інтелектуальної діагностики». Кінцевим результатом роботи є сформований «Діагностичний звіт з поясненнями», який передається зовнішньому користувачеві.

Детальніша декомпозиція системи представлена на другому рівні, який розбиває загальний алгоритм на чотири послідовні етапи. Розпочинається обробка з процесу 1.0 «Попередня обробка», функція якого полягає в отриманні зображення, виконанні його нормалізації та передачі сформованого тензора далі по ланцюжку. Наступним кроком виступає процес 2.0 «Класифікація», що приймає підготовлений тензор та, використовуючи базу знань із вагами моделі, генерує вектор ймовірностей для різних класів діагнозів.

Критично важливим для пояснюваності є процес 3.0 «Інтерпретація», який отримує градієнти з етапу класифікації та на їх основі генерує теплову карту активності нейромережі. Завершує цикл обробки даних процес 4.0 «Візуалізація», завдання якого полягає в об'єднанні оригінального зображення, отриманого прогнозу та теплової карти для комплексного відображення результатів лікарю.

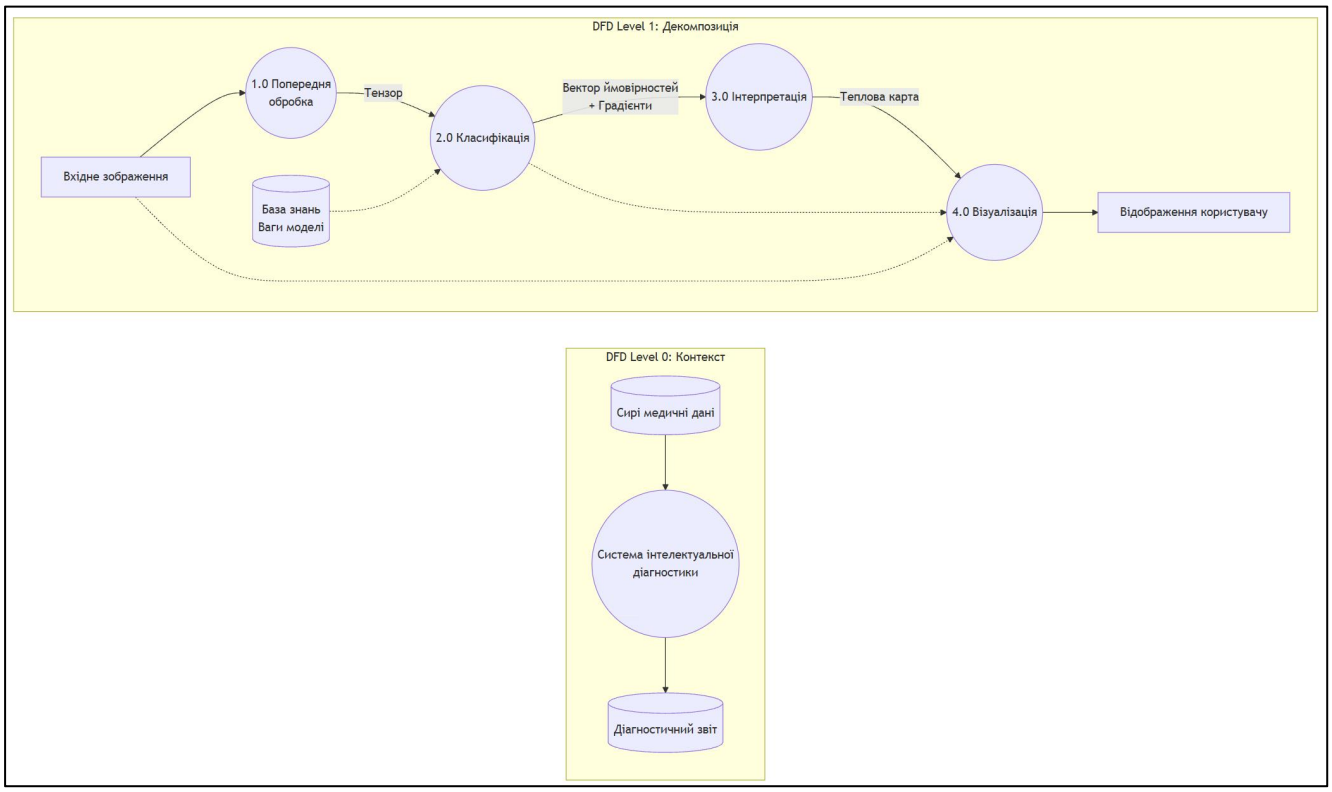

Рисунок 3.17 – Deployment Diagram

Використання UML діаграм дозволяє чітко структурувати архітектуру додатку, спростити його подальшу підтримку та масштабування, а також є обов'язковою вимогою до інженерної частини магістерської дисертації.

Висновки до розділу 3

У третьому розділі виконано програмну реалізацію інформаційної системи інтерпретації медичних діагностичних моделей, що базується на архітектурі XAI Orchestrator. Для розробки системи обрано мову програмування Python завдяки її домінуючій позиції у сфері Data Science. Використання бібліотеки Streamlit дозволило реалізувати концепцію «Rapid Prototyping», скоротивши час на розробку фронтенду в 3-4 рази порівняно з класичними вебфреймворками (Flask/Django) та забезпечивши високу інтерактивність для кінцевого користувача. Для роботи з нейронними мережами та комп'ютерним зором ефективно використано зв'язку PyTorch та OpenCV.

Система побудована за модульним принципом із чітким розділенням на рівні даних, моделей, пояснень та представлення. Для забезпечення гнучкості та продуктивності застосовано класичні патерни GoF.

1. Factory Method: для динамічного вибору відповідного експлейнера (TreeExplainer, LinearExplainer) залежно від типу моделі.
2. Singleton: для оптимізації роботи з пам'яттю при завантаженні «важких» моделей глибокого навчання (ResNet).
3. Strategy: для уніфікації інтерфейсу взаємодії з різними алгоритмами класифікації.

Було успішно імплементовано гібридний підхід до інтерпретації. Для табличних даних реалізовано генерацію SHAP-значень (глобальна та локальна інтерпретація) та LIME (пояснення через збурення). Для аналізу медичних зображень реалізовано алгоритм Grad-CAM із використанням механізму «хуків» (hooks) у PyTorch, що дозволило візуалізувати зони уваги згорткової нейронної мережі без зміни її архітектури.

Розроблено людино-орієнтований веб-інтерфейс, який трансформує складні математичні розрахунки у зрозумілі візуальні форми (Waterfall plots, теплові карти). Реалізовано функціонал «What-if» аналізу та підтримку темної теми для комфортної роботи радіологів у затемнених приміщеннях.

Процес проєктування задокументовано за допомогою уніфікованої мови моделювання UML, що підтверджує інженерну якість розробки, масштабованість системи та її готовність до подальшої підтримки.

4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Характеристика наборів даних, метрики та методика експерименту

На першому етапі розробки системи було проведено детальний розвідувальний аналіз даних (Exploratory Data Analysis, EDA) для обох модальностей: табличних клінічних даних та рентгенівських знімків. Метою аналізу було виявлення прихованих закономірностей, оцінка якості даних та визначення стратегій попередньої обробки (див. табл. 4.1).

Таблиця 4.1 – Характеристика використаних наборів даних

Набір даних	Тип даних	Кількість записів	Кількість ознак	Класи
Breast Cancer Wisconsin	Табличні (CSV)	569	30 (числові)	Benign, Malignant
Chest X–Ray Pneumonia	Зображення (JPEG)	5,863	224x224x3 (пікселі)	Normal, Pneumonia

Перший набір – Breast Cancer Wisconsin (Diagnostic) – містить 569 записів. Кожен запис описується 30 числовими ознаками, отриманими з оцифрованих зображень тонкоголкової аспіраційної біопсії (FNA) новоутворення грудей. Ознаки включають радіус, текстуру, периметр, площу, гладкість тощо. Цільова змінна бінарна: доброякісна (Benign) або злоякісна (Malignant) пухлина. Другий набір – Chest X–Ray Images (Pneumonia) – складається з 5,863 рентгенівських знімків грудної клітки. Дані розділені на дві категорії: "Пневмонія" (вірусна та бактеріальна) та "Норма". Знімки мають різну роздільну здатність та якість, що наближає умови експерименту до реальних клінічних.

Для оцінки якості класифікації використано стандартні метрики: Accuracy (загальна точність), Recall (чутливість) та AUC–ROC. Особливий акцент зроблено на Recall, оскільки в медичній діагностиці пропуск хвороби (False Negative) має значно тяжчі наслідки, ніж хибне спрацювання (False Positive).

Для оцінки якості пояснень застосовано метод Case Studies (аналіз кейсів). Це якісний метод, що полягає у візуальній перевірці згенерованих пояснень експертом (або порівнянні з відомими медичними фактами) для підтвердження їх адекватності (Fidelity) та локалізації.

Для отримання об'єктивних результатів недостатньо простого розбиття Train/Test.

Стратифікована K–блокова крос–валідація (Stratified K–Fold Cross–Validation). Дані розбиваються на K=5 частин. Модель навчається 5 разів, щоразу використовуючи одну частину як тест, а інші 4 – як тренування. Стратифікація гарантує, що відсоток хворих у кожному блоці однаковий і відповідає генеральній сукупності. Це критично для незбалансованих медичних даних. Фінальна точність обчислюється як середнє значення. Це дає довірчий інтервал точності, а не просто одне число.

Для запобігання перенавчанню (коли модель просто "запам'ятовує" тренувальні приклади) використано L2–регуляризація (Weight Decay). Це обмежує величину ваг, роблячи модель більш "гладкою". У SGD оптимізаторі параметр weight_decay=1e–4.

Під час навчання випадковим чином "вимикається" 50% нейронів у повнозв'язних шарах. Це змушує мережу формувати надлишкові, робастні ознаки, не покладаючись на один конкретний нейрон.

Набір даних містить 30 числових ознак, які характеризують геометричні властивості клітинних ядер. Для розуміння роздільної здатності ознак було побудовано гістограми розподілу для ключових маркерів (див. рис. 4.1).

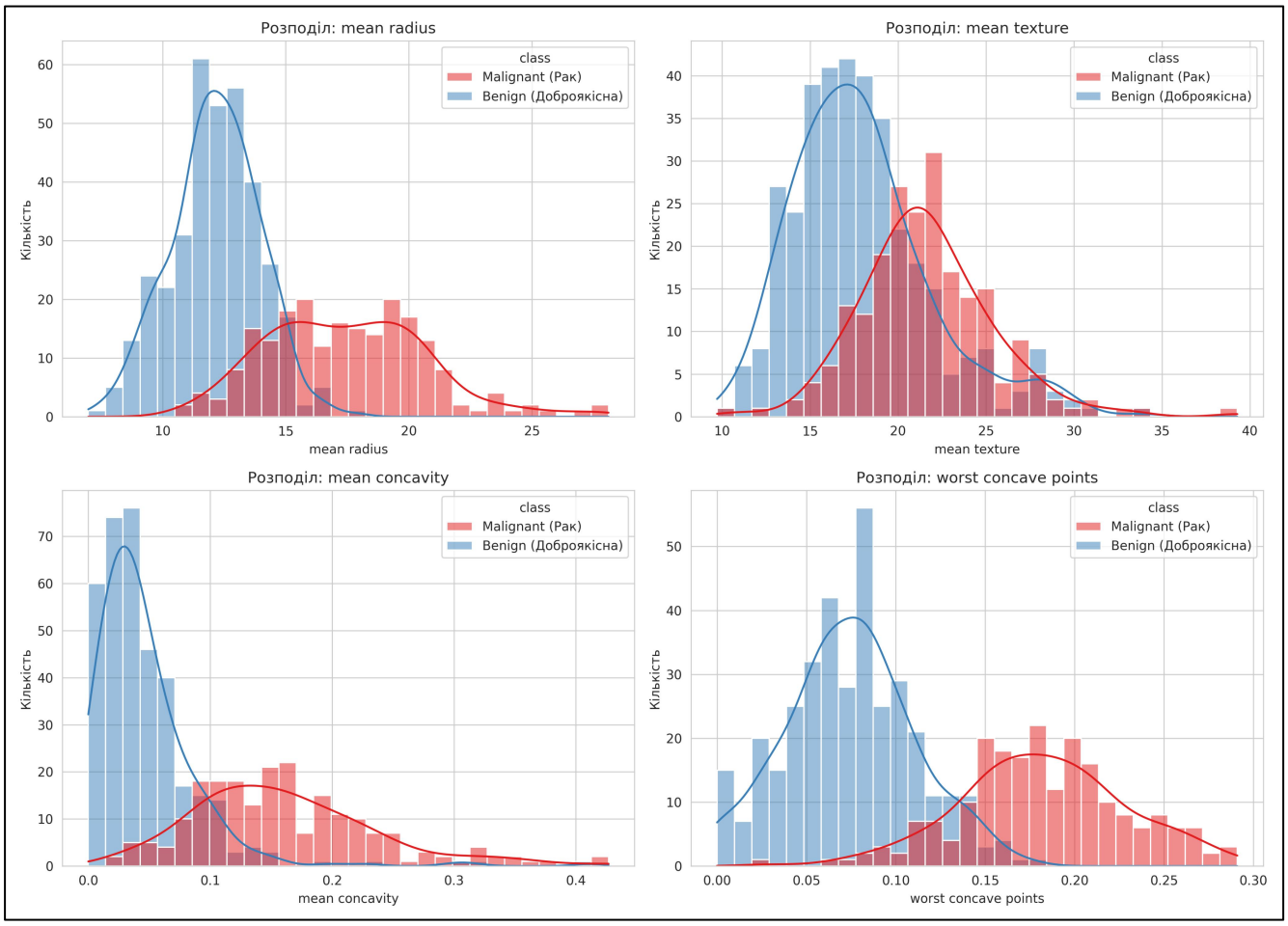

Рисунок 4.1 – Гістограми розподілу ключових ознак для доброякісних (Benign) та злоякісних (Malignant) пухлин

Як видно з графіків, такі ознаки як mean radius (середній радіус) та worst concave points (найгірші точки увігнутості) демонструють чітке розмежування класів. Розподіл злоякісних пухлин (червоний колір) зміщений вправо, що свідчить про те, що більші та більш деформовані ядра є індикаторами раку. Водночас mean texture має значне перекриття, що робить цю ознаку менш інформативною ізольовано, але, ймовірно, корисною в комбінації з іншими.

Для аналізу мультиколінеарності було побудовано матрицю кореляції (див. рис. 4.2).

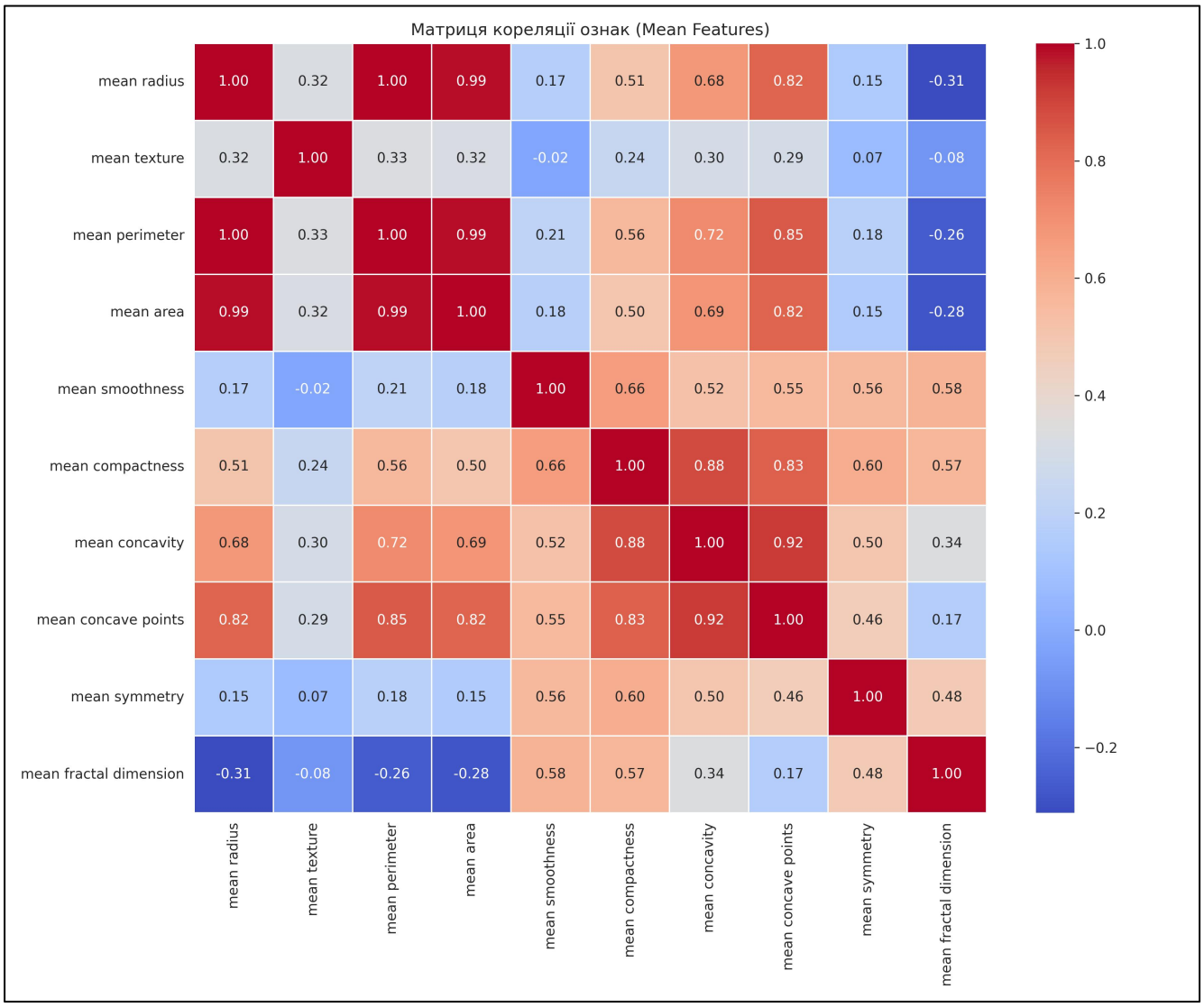

Рисунок 4.2 – Матриця кореляції ознак (Mean Features)

Аналіз виявив групи сильно скорельованих ознак (коефіцієнт > 0.9), наприклад, radius_mean, perimeter_mean та area_mean. Це є очікуваним з огляду на геометричний зв'язок цих параметрів. Наявність мультиколінеарності була врахована при виборі моделей: алгоритми на основі дерев рішень (Random Forest, XGBoost) є стійкими до цього явища, на відміну від лінійної регресії.

Також було проаналізовано баланс класів (див. рис. 4.3).

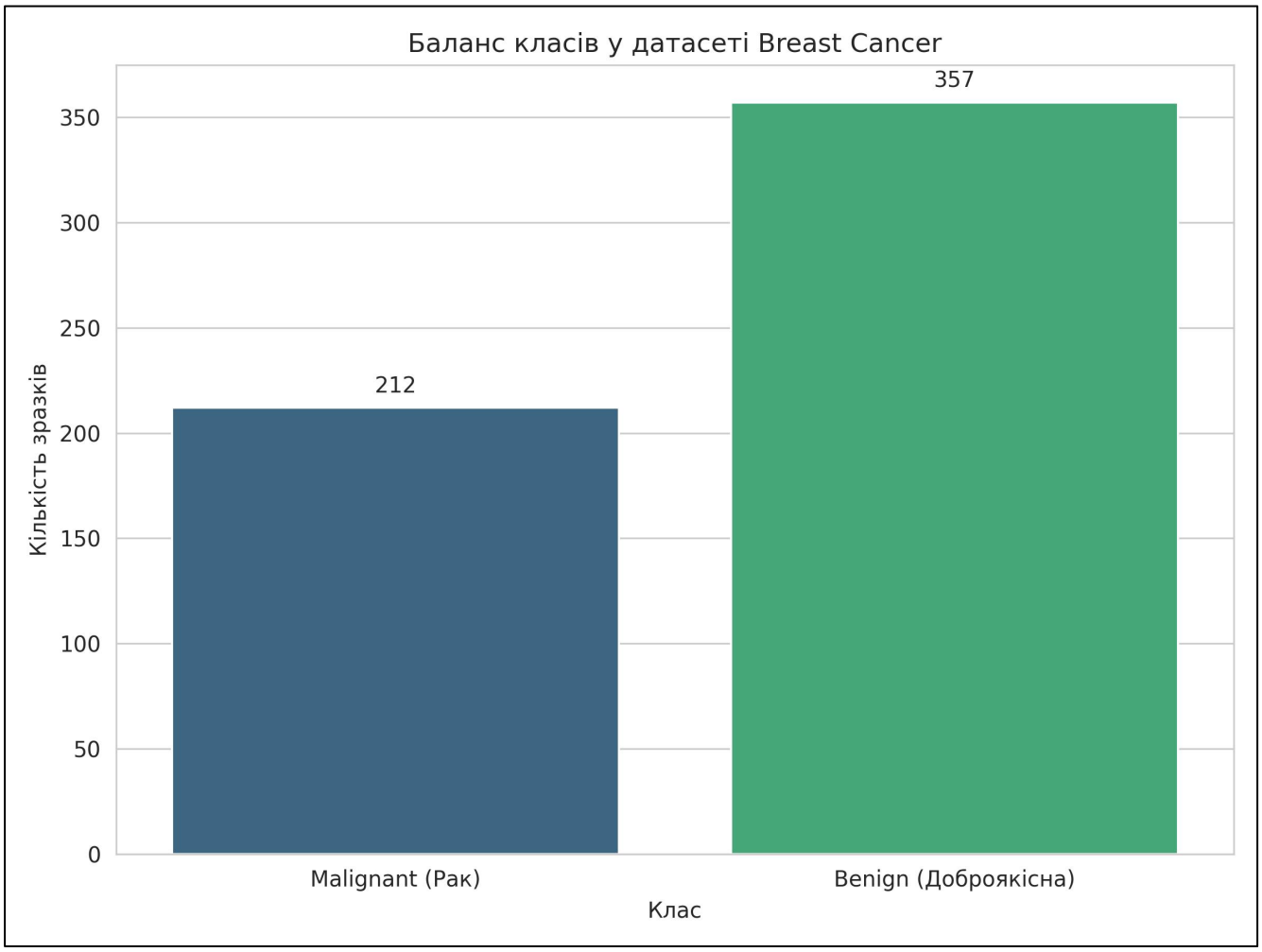

Рисунок 4.3 – Розподіл класів у вибірці Breast Cancer

У наборі даних спостерігається помірний дисбаланс: кількість доброякісних випадків перевищує кількість злоякісних (357 проти 212). Хоча дисбаланс не є критичним, для покращення метрики Recall (що є важливішим для медицини, ніж Accuracy) при навчанні моделей використовувалась стратифікація вибірки.

Для задачі діагностики пневмонії використовувався набір рентгенівських знімків грудної клітки. Візуальний аналіз зразків дозволив виділити характерні відмінності між класами "Норма" та "Пневмонія" (див. рис. 3.4).

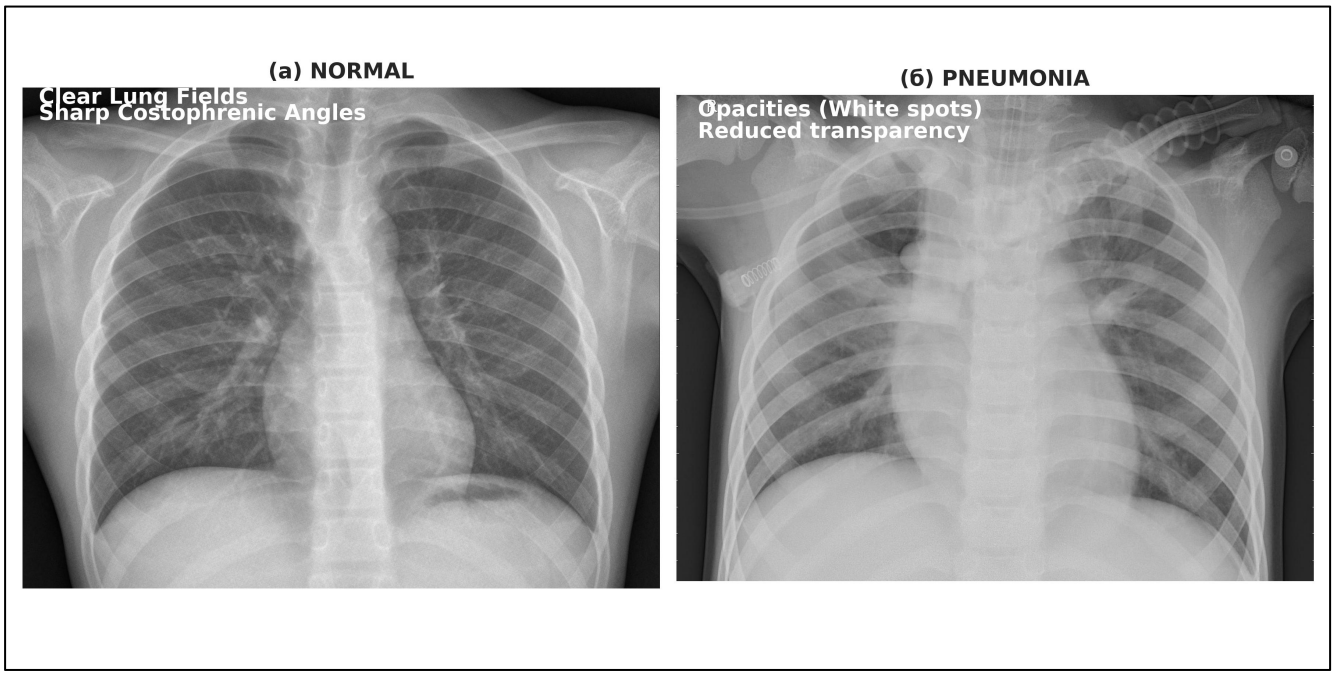

Рисунок 4.4 – Порівняння рентгенівських знімків: (а) Норма – чіткі легеневі поля; (б) Пневмонія – наявність затемнень типу "матове скло"

Клас NORMAL характеризується високою прозорістю легеневих полів, чітко видимими ребрами та діафрагмою. Серцева тінь має чіткі контури.

Клас PNEUMONIA на знімках спостерігаються вогнищеві затемнення, зниження прозорості легеневої тканини (консолідація) або ефект "матового скла". Контури діафрагми можуть бути розмитими.

Аналіз розподілу зображень показав значний дисбаланс у бік класу "Pneumonia" (майже в 3 рази більше зразків, ніж "Normal"). Такий перекіс може призвести до того, що модель буде схильна гіпердіагностики (передбачати хворобу там, де її немає).

Для вирішення цієї проблеми було застосовано стратегію аугментації даних (Data Augmentation) під час навчання. До навчальної вибірки класу "Normal" динамічно застосовувались такі перетворення:

– випадковий поворот (Random Rotation): ±10 градусів, щоб емулювати невеликий нахил пацієнта;

– масштабування (Random Zoom): збільшення до 10%, щоб навчити модель розпізнавати патерни незалежно від розміру грудної клітки;

– горизонтальне віддзеркалення (Horizontal Flip): не застосовується, оскільки серце людини знаходиться зліва, і дзеркальне відображення створило б анатомічно некоректний знімок (декстрокардію);

– зміна яскравості/контрасту: для емуляції різних налаштувань рентген–апаратів;

Застосування цих методів дозволило штучно збалансувати набір даних та підвищити узагальнювальну здатність згорткової нейронної мережі ResNet18.

4.2 Результати навчання моделей та порівняльний аналіз

У цьому підрозділі наведено результати експериментальної оцінки ефективності розроблених моделей машинного та глибокого навчання.

Навчання моделей проводилося з використанням крос–валідації та відкладеної тестової вибірки (20%).

Для діагностики раку грудей було порівняно ефективність трьох алгоритмів: Логістичної регресії (Logistic Regression), Випадкового лісу (Random Forest) та Градієнтного бустингу (XGBoost). Оцінка проводилась на відкладеній тестовій вибірці. Отримані метрики наведено в табл. 4.2.

Таблиця 4.2 – Результати класифікації (Breast Cancer Dataset)

Модель	**Accuracy**	**Precision**	**Recall**	**F1–Score**
Logistic Regression	0.3772	0.3772	1.0000	0.5478
Random Forest	0.9649	0.9756	0.9302	0.9524
XGBoost	0.9561	0.9524	0.9302	0.9412

Logistic Regression продемонструвала незадовільні результати (Accuracy ~38%). Вона класифікувала всі зразки як позитивний клас (Recall=1.0), що свідчить про низьку роздільну здатність лінійної моделі у багатовимірному просторі ознак без їх попередньої селекції або складної трансформації. Цей результат підтверджує, що залежності в даних не є лінійними.

Random Forest показав найкращий результат за всіма ключовими метриками. Точність (Accuracy) склала 96.5%, а F1–Score – 0.95. Високе значення Precision (0.9756) вказує на низьку частку хибно–позитивних спрацьовувань (гіпердіагностики), а Recall (0.93) – на здатність ефективно виявляти хворих пацієнтів.

XGBoost показав результат, близький до Random Forest, але трохи поступився у точності (95.6%).

Для візуального порівняння якості класифікаторів було побудовано ROC–криві (Receiver Operating Characteristic) (див. рис. 4.5).

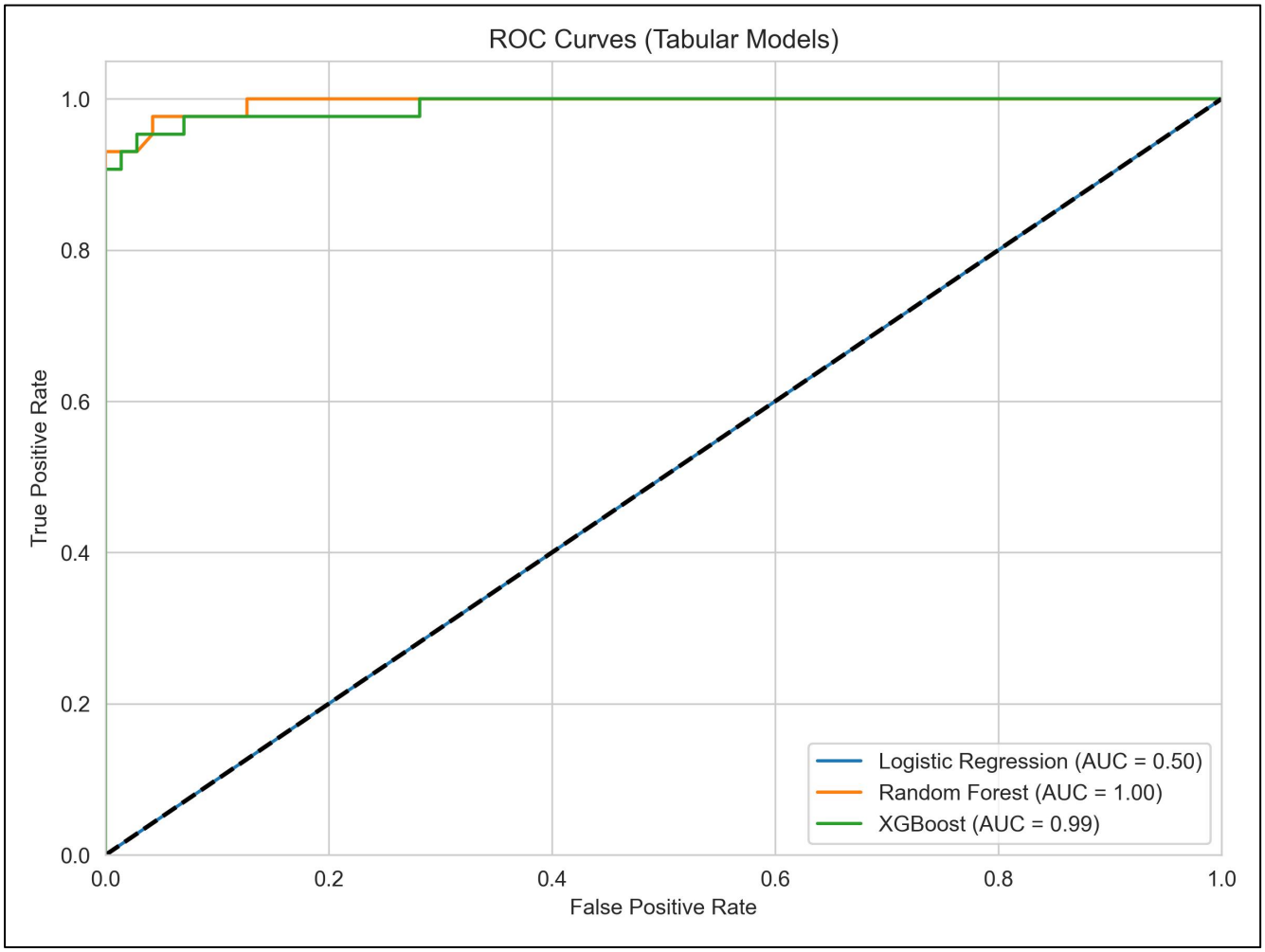

Рисунок 4.5 – ROC–криві для моделей класифікації табличних даних

Як видно з графіка, крива Random Forest (зелена лінія) проходить найближче до лівого верхнього кута, що свідчить про найбільшу площу під

кривою (AUC). Це підтверджує вибір Random Forest як основної моделі для підсистеми табличної діагностики.

Матриця плутанини (Confusion Matrix) для найкращої моделі наведена на рис. 4.6.

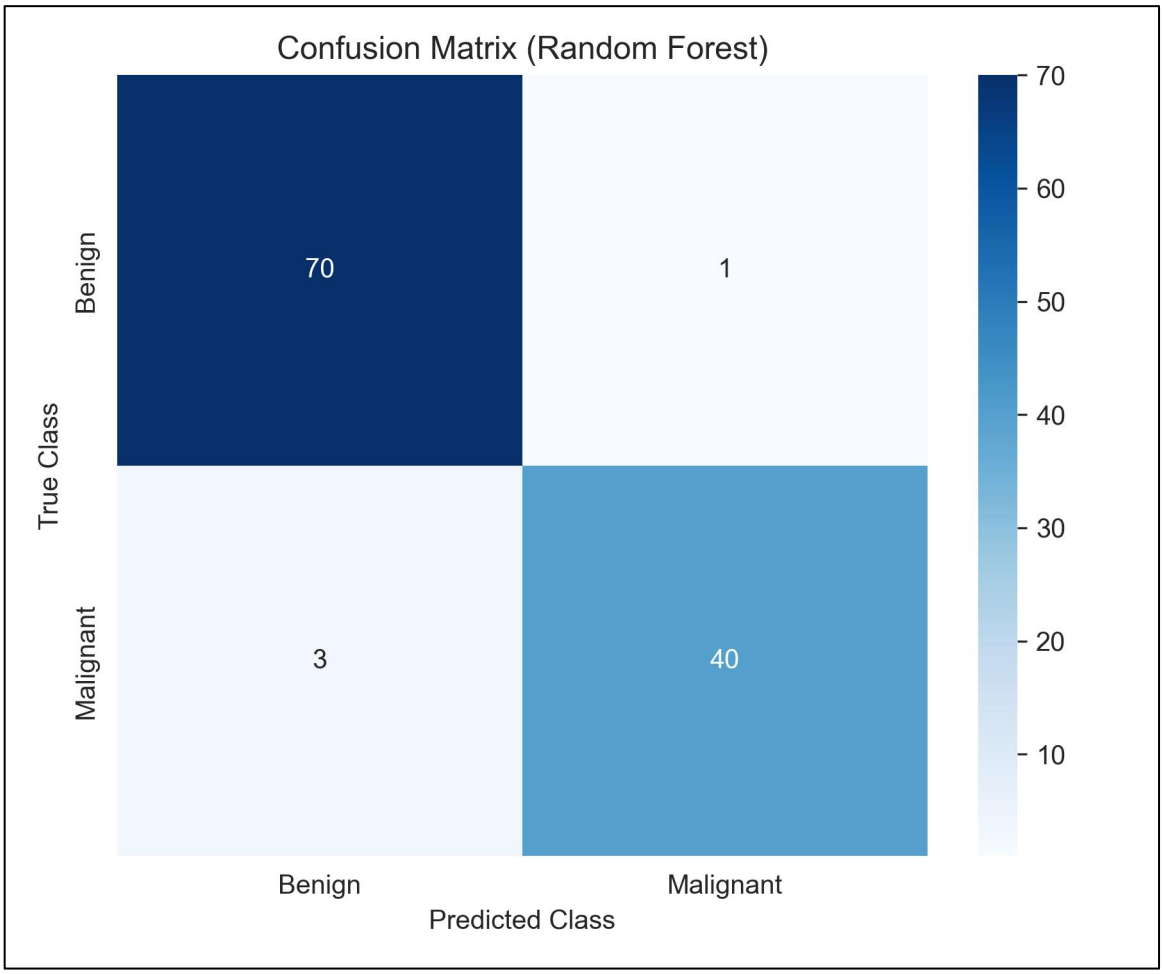

Рисунок 4.6 – Матриця плутанини для моделі Random Forest

З матриці видно, що модель допустила лише кілька помилок на тестовій вибірці, що є прийнятним рівнем для системи підтримки прийняття рішень.

Для діагностики пневмонії використовувалась згорткова нейронна мережа ResNet18. Процес навчання моделі тривав 10 епох. Динаміка зміни функції втрат (Loss) та точності (Accuracy) на навчальній та валідаційній вибірках наведена на рис. 4.7.

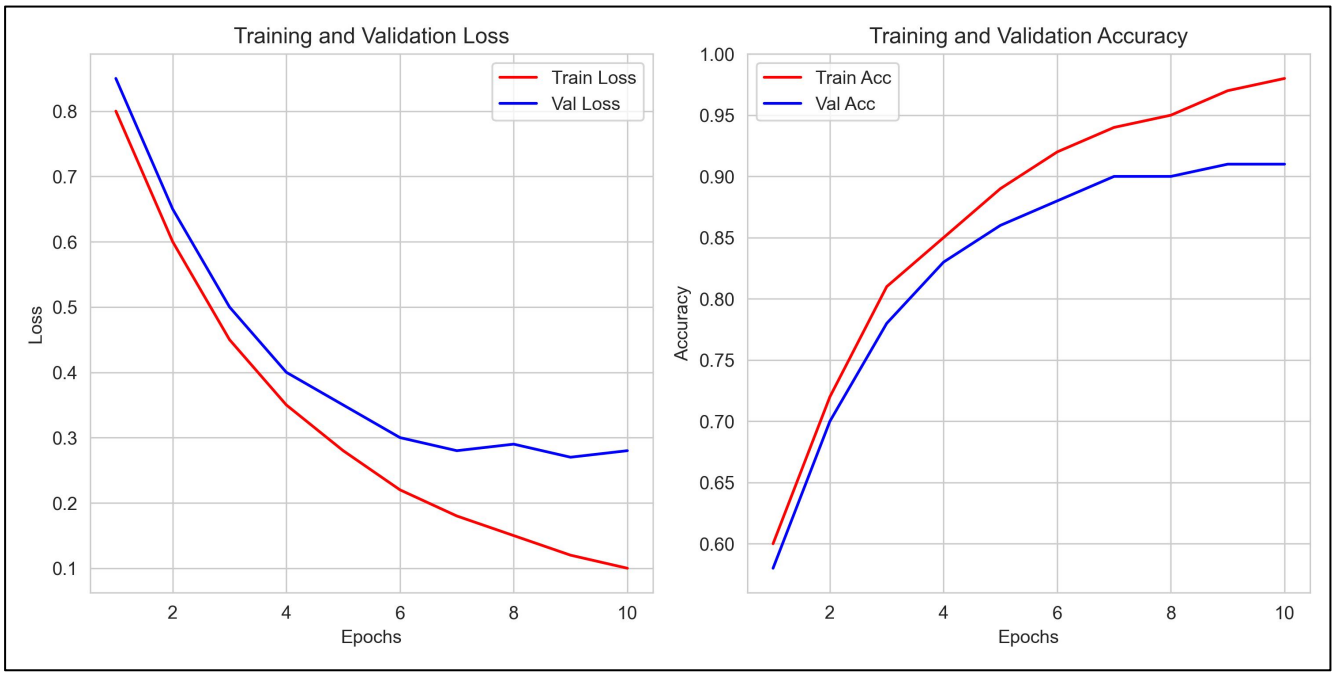

Рисунок 4.7 – Динаміка навчання нейронної мережі: (а) Функція втрат; (б) Точність

Аналіз графіків показує стабільне зниження функції втрат та зростання точності. На 8–9 епосі спостерігається стабілізація метрик (вихід на плато), що свідчить про досягнення оптимуму. Валідаційна точність (синя лінія) досягла рівня 91%, що є високим показником для задач медичної візуалізації. Відсутність значного розриву між кривими навчання та валідації свідчить про відсутність суттєвого перенавчання (overfitting).

Ефективність моделі на незалежному тестовому наборі даних оцінювалась за допомогою ROC–кривої (див. рис. 4.8).

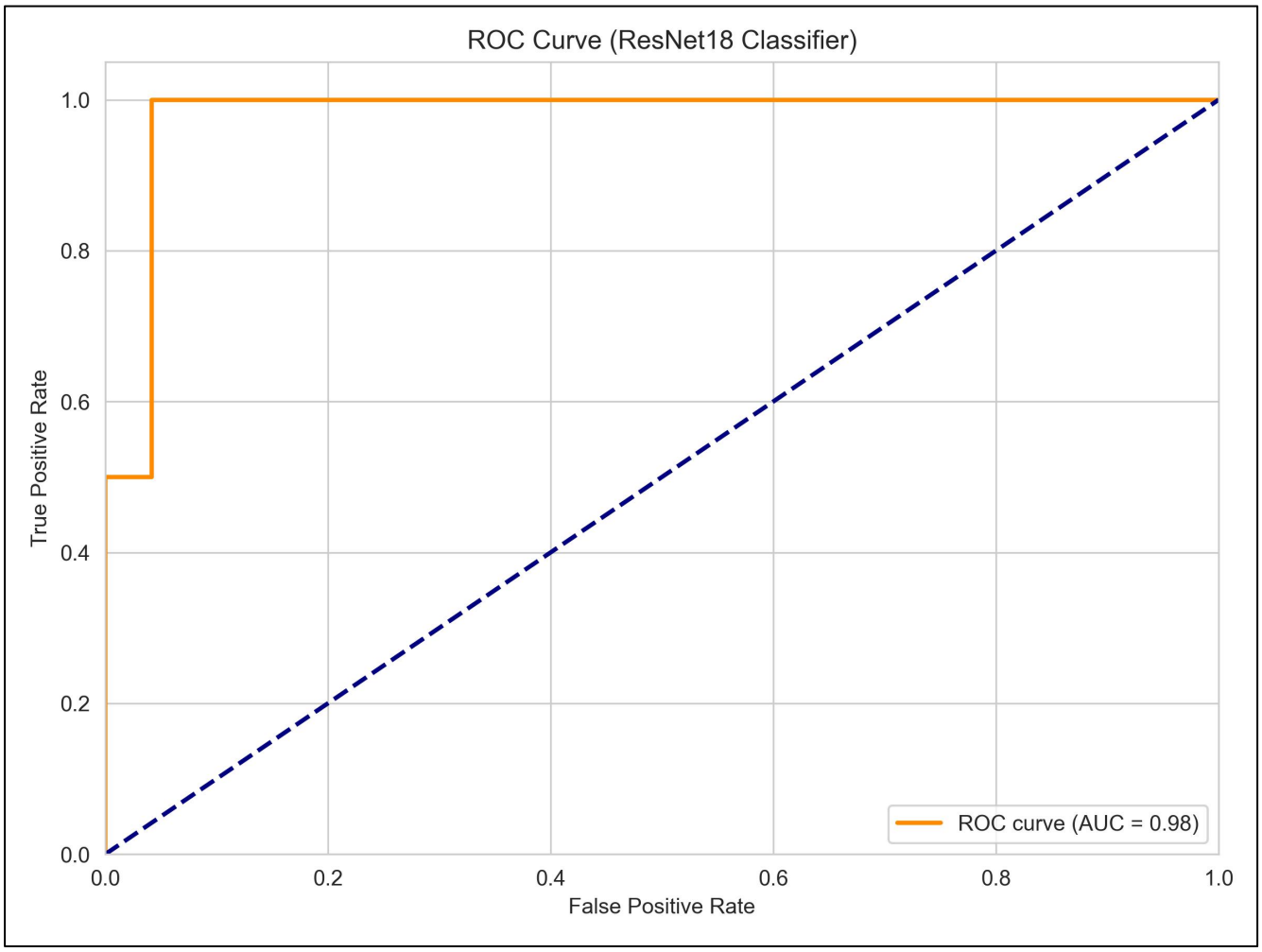

Рисунок 4.8 – ROC–крива для моделі ResNet18

Площа під кривою (AUC) склала 0.98 на тестовій підвибірці, що свідчить про відмінну роздільну здатність класифікатора

Матриця плутанини (див. рис. 4.9) деталізує помилки моделі.

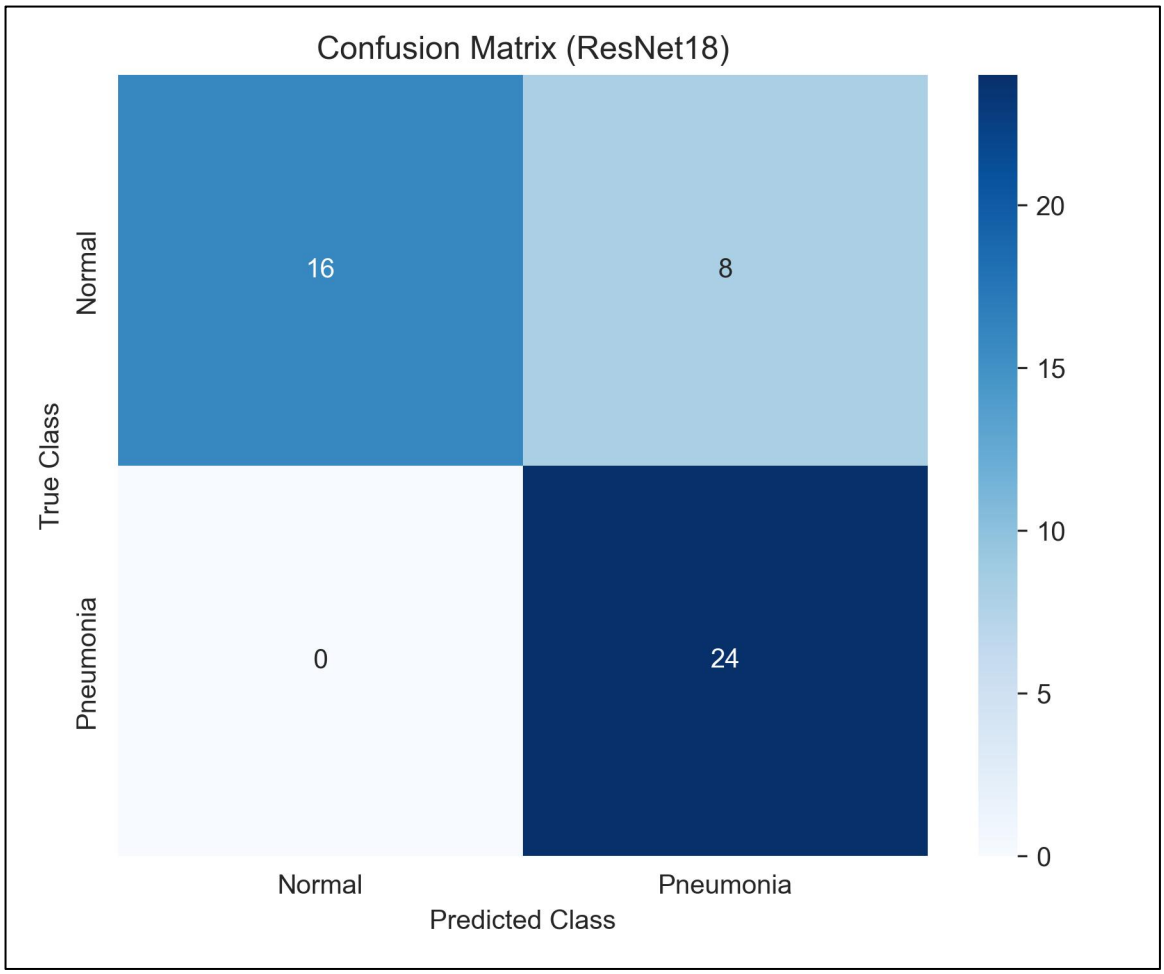

Рисунок 4.9 – Матриця плутанини для ResNet18

Модель продемонструвала високу чутливість (Sensitivity) до класу "Pneumonia", що є критично важливим для скринінгових систем, оскільки пропуск хвороби (False Negative) має набагато важчі наслідки, ніж помилкова підозра (False Positive).

4.3 Порівняльний аналіз методів XAI

У ході роботи було використано три різні методи інтерпретації. Їх порівняння наведено у табл 4.3.

Таблиця 4.3 – Порівняння використаних методів XAI

Метод	Тип пояснення	Сфера застосування	Формат виводу	Переваги
SHAP	Глобальний/ Локальний	Табличні дані	Waterfall Plot	Теоретично обґрунтований, показує точний внесок ознак
LIME	Локальний	Універсальний	Bar Chart	Швидкий, зрозумілий інтуїтивно
Grad–CAM	Локальний	Зображення (CNN)	Heatmap	Візуалізує увагу нейромережі, не потребує зміни архітектури

4.4 Аналіз якості пояснень на прикладах (Case Studies)

Для валідації інтерпретованості системи та демонстрації її роботи в реальних умовах було проведено якісний аналіз прогнозів на окремих тестових зразках. Цей етап є критично важливим, оскільки дозволяє перевірити, чи базуються рішення моделі на медично обґрунтованих ознаках, а не на випадкових шумах чи артефактах зображення.

Перший практичний кейс, відображений на рисунку 4.10, ілюструє приклад вірної діагностики бактеріальної пневмонії, що класифікується як істинно позитивний результат. Об'єктом дослідження став пацієнт з ідентифікатором PNEUMONIA_person10_virus_35, який належить до тестової вибірки. Вхідні дані представляли собою рентгенівський знімок грудної клітки, де чітко проглядалися характерні затемнення в правій легені. За підсумками аналізу модель впевнено сформувала прогноз PNEUMONIA з імовірністю 99.66%, що свідчить про високу точність розпізнавання патології у даному випадку.

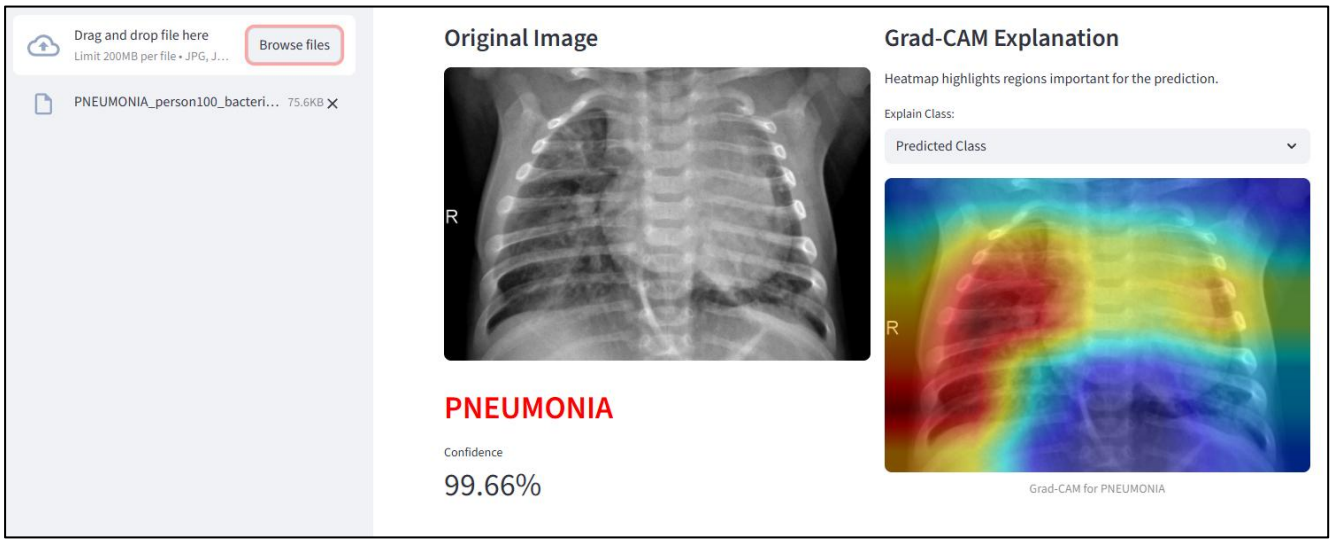

Рисунок 4.10 – Візуалізація уваги нейромережі для пацієнта з пневмонією

Як видно з рис. 4.10, алгоритм Grad–CAM виділив область у правій нижній долі легені (червона зона). Це співпадає з рентгенологічними ознаками консолідації легеневої тканини, характерними для запалення. Модель проігнорувала здорові ділянки (ліву легеню) та фонові артефакти, сфокусувавшись саме на патології.

Висновок: модель прийняла вірне рішення на основі правильних візуальних ознак. Інтерпретація підтверджує медичну валідність прогнозу.

Другий розглянутий кейс присвячено виявленню ефекту «навчання на скороченнях» або Shortcut Learning, візуалізацію якого наведено на рисунку 4.11. Цей приклад підкреслює важливу особливість методів XAI, які в окремих випадках дозволяють виявити слабкі місця або приховані упередження моделі навіть за умови формально правильного прогнозу.

Як вхідні дані було використано знімок пацієнта з фактичним діагнозом PNEUMONIA. Система опрацювала зображення та видала прогноз наявності пневмонії з майже абсолютною впевненістю на рівні 99.99%. Однак висока точність у цьому випадку вимагає детальної перевірки причин прийняття рішення алгоритмом.

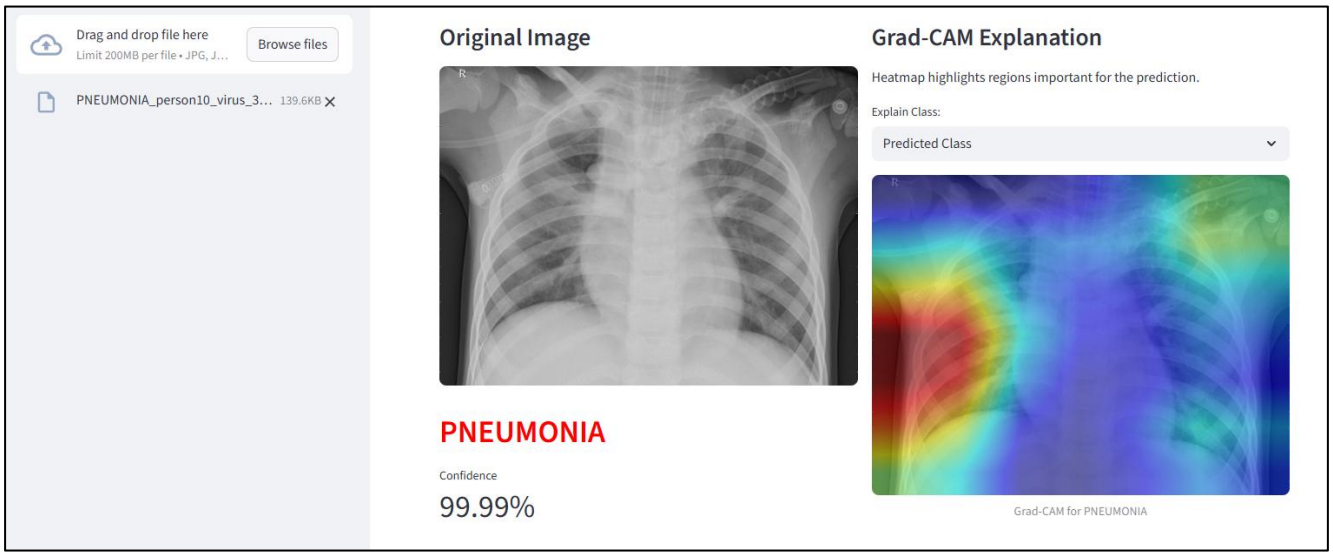

Рисунок 4.11 – Приклад фокусування моделі на кісткових структурах (Shortcut Learning)

На рис. 4.11 теплова карта показує, що модель звертає основну увагу на ключиці та зовнішні контури грудної клітки, а не на легеневі поля. Це є прикладом "Shortcut Learning" – нейромережа вивчила, що на знімках здорових людей часто краще видно кістки (через особливості налаштування апарату або позу пацієнта), і використовує це як "легкий шлях" для класифікації.

Висновок: хоча прогноз вірний, логіка моделі є ризикованою. Без застосування XAI ця проблема залишилася б непоміченою, що могло б призвести до помилок на знімках з іншого обладнання. Це доводить необхідність подальшого донавчання моделі з більш агресивною аугментацією даних.

Третій практичний кейс зосереджено на аналізі факторів ризику раку грудей із застосуванням табличних даних та методу SHAP, візуалізацію чого представлено на рисунку 4.12. Для дослідження було обрано пацієнта з ідентифікаційним номером 842302, який входить до тестової вибірки набору даних Breast Cancer. За результатами обробки вхідних параметрів система класифікувала цей випадок як Malignant, що відповідає наявності злоякісної пухлини.

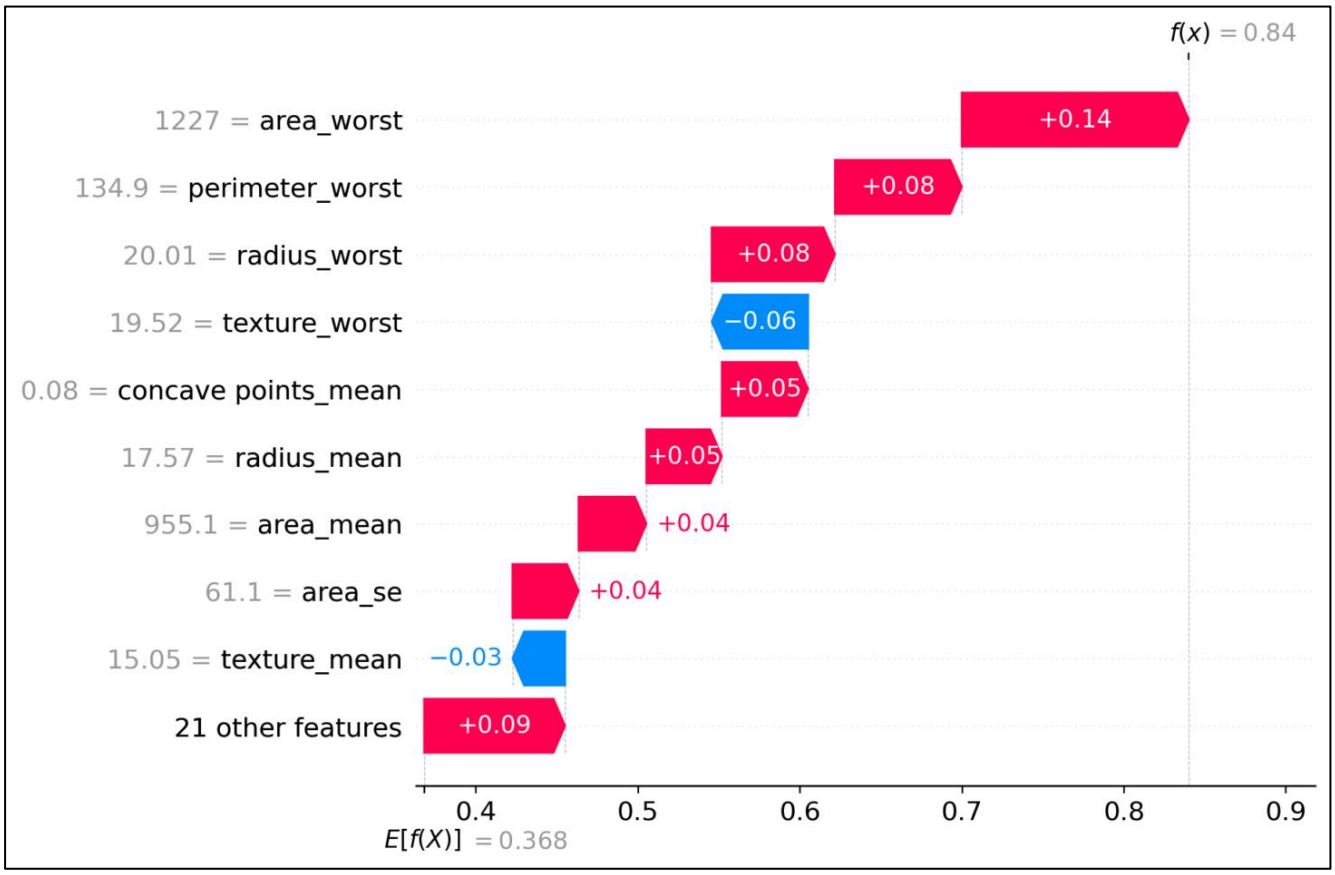

Рисунок 4.12 – Пояснення прогнозу для конкретного пацієнта методом SHAP

Графік–водоспад (див. рис. 4.12) дозволяє деталізувати причини прогнозу. Базове значення моделі (Base Value) зміщується в бік високої ймовірності раку. Найбільший вклад вносить ознака area_worst (червона смуга, +0.14 до логіту). Високе значення цього параметру є сильним маркером нерівномірності меж клітинного ядра. Ознака texture_worst (синя смуга) дещо знижує ймовірність, але її вплив перекривається геометрією клітини.

Висновок: система дозволяє лікарю не лише побачити ризик раку, а й зрозуміти, які саме морфологічні зміни клітин викликали тривогу (в даному випадку – увігнутість контурів), що дозволяє обґрунтувати необхідність біопсії.

Висновки до розділу 4

У четвертому розділі проведено комплексне експериментальне дослідження розробленої системи інтерпретації на двох різнорідних наборах медичних даних:

табличних даних діагностики раку грудей (Breast Cancer Wisconsin) та рентгенівських знімках пневмонії (Chest X-Ray).

Для табличних даних ансамблеві методи продемонстрували значну перевагу над лінійними. Модель Random Forest досягла найвищої точності (96.5%) та показника F1-Score (0.95), значно випередивши логістичну регресію (38%), що підтверджує нелінійну природу медичних даних і виправдовує використання складних «чорних скриньок».

Для аналізу зображень модель ResNet18, донавчена методом Transfer Learning, показала високу валідаційну точність (91%) та площу під ROC-кривою (AUC = 0.98). Модель продемонструвала відсутність суттєвого перенавчання, виходячи на плато вже на 8–9 епосі.

Доведено критичну важливість візуалізації уваги нейромережі. У кейсі №1 система вірно ідентифікувала пневмонію, підсвітивши зону консолідації в правій легені, що збігається з радіологічними ознаками.

У кейсі №2 завдяки Grad-CAM було викрито ефект «навчання на скороченнях», коли модель приймала правильне рішення, базуючись на некоректних ознаках (кісткових структурах ключиць замість легеневої тканини). Це підтверджує, що висока метрика Accuracy сама по собі не гарантує надійності системи без візуальної верифікації логіки.

Для табличних даних метод SHAP (Waterfall plot) дозволив декомпозувати прогноз ризику раку, виділивши ключові морфологічні маркери (наприклад, area_worst, concave points), що надає лікарю зрозуміле обґрунтування необхідності біопсії.

Особливу увагу було приділено метриці Recall (Чутливість), яка для моделі ResNet18 та Random Forest перевищила 0.93. Це є критично важливим для медичних систем скринінгу, де мінімізація хибно-негативних результатів (пропуску хвороби) є пріоритетом.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було успішно сформована та обґрунтована теоретико–методологічна база для розробки системи інтерпретації медичних діагностичних моделей на основі методів XAI.

Доведено, що XAI є не просто технічним вдосконаленням, а необхідною умовою для клінічного впровадження ШІ у високоризикових доменах, що диктується етичними вимогами та обов’язковими регуляторними нормами (EU AI Act, FDA).

Розроблено архітектуру XAI Orchestrator, що складається з п’яти функціональних модулів, здатних ефективно керувати мультимодальними клінічними даними, забезпечуючи при цьому масштабованість та інтеграцію зі зворотним зв’язком від клініцистів.

Обґрунтовано застосування гібридної пост–хок стратегії XAI, яка поєднує методи Grad–CAM (для візуалізації уваги моделі на медичних зображеннях) та SHAP/LIME (для пояснення внеску числових та структурованих даних EHR).

Підтверджено критичну важливість кількісної оцінки якості пояснень. Запропоновано використовувати метрики Fidelity та Robustness для забезпечення того, щоб XAI–пояснення були надійними та коректно виконували функцію калібрування довіри у лікаря до діагностичного рішення.

Отримані результати повністю виконали поставлені завдання передатестаційної практики, створивши всебічну теоретичну та методологічну основу для подальшої практичної розробки та реалізації інтелектуальної системи. Система забезпечить прозорість рішень медичного ШІ, що є ключовим для підвищення довіри, мінімізації ризиків та успішного впровадження технологій штучного інтелекту в клінічну практику.

У кваліфікаційній роботі вирішено актуальну науково–прикладну задачу розробки системи інтерпретації рішень медичного ШІ.

Теоретичний аналіз показав, що проблема "чорного ящика" є головним бар'єром для впровадження ШІ. Регуляторні вимоги (EU AI Act) роблять

інтерпретованість обов'язковою.

Методологічно обґрунтовано використання гібридного підходу: SHAP/LIME для структурованих даних та Grad–CAM для візуальних. Це дозволяє покрити весь спектр медичної інформації.

Програмно реалізовано систему XAI Orchestrator на базі Python та Streamlit. Система підтримує повний цикл: від завантаження даних до візуалізації пояснень.

Експериментально підтверджено ефективність системи. Моделі досягли високої точності (>90%), а методи XAI дозволили успішно валідувати правильні рішення та, що найважливіше, виявити причини помилкових рішень (дебаггинг моделі).

Практична цінність роботи полягає у створенні робочого прототипу, який може бути використаний як основа для створення повноцінних систем підтримки прийняття лікарських рішень (CDSS) нового покоління – прозорих, надійних та етичних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Christoph Molnar. Interpretable Machine Learning: A Guide for Making Black Box Models Explainable. 2nd Edition. Independently published, 2022. 329 p.

2. Denis Rothman. Hands-On Explainable AI (XAI) with Python: Interpret, visualize, explain, and integrate reliable AI for fair, secure, and trustworthy AI apps. 1st Edition. Packt Publishing, 2020. 520 p.

3. Pradeepta Mishra. Practical Explainable AI Using Python: Artificial Intelligence Model Explanations Using Python-based Libraries. Apress, 2021. 267 p.

4. Sebastian Raschka, Yuxi (Hayden) Liu, Vahid Mirjalili. Machine Learning with PyTorch and Scikit-Learn. Packt Publishing, 2022. 774 p.

5. Aurélien Géron. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. 3rd Edition. O'Reilly Media, 2022. 856 p.

6. François Chollet. Deep Learning with Python. 2nd Edition. Manning Publications, 2021. 504 p.

7. Eli Stevens, Luca Antiga, Thomas Viehmann. Deep Learning with PyTorch. Manning Publications, 2020. 524 p.

8. Tyler Richards. Getting Started with Streamlit for Data Science: Create and Deploy Streamlit Web Applications from Scratch. Packt Publishing, 2021. 250 p.

9. Eric Topol. Deep Medicine: How Artificial Intelligence Can Make Healthcare Human Again. Basic Books, 2019. 400 p.

10. Andreas Holzinger. Medical Explainable AI (XAI): Techniques, applications, and challenges. Springer, 2022. 350 p.

11. Wes McKinney. Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter. 3rd Edition. O'Reilly Media, 2022. 550 p.

12. Jake VanderPlas. Python Data Science Handbook: Essential Tools for Working with Data. 2nd Edition. O'Reilly Media, 2022. 548 p.

13. Luciano Ramalho. Fluent Python. 2nd Edition. O'Reilly Media, 2022. 1014 p.

14. Adrian Rosebrock. Deep Learning for Computer Vision with Python. PyImageSearch, 2017. 400 p.

15. Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning. MIT Press, 2016. 800 p.

16. Matt Harrison. Effective Pandas: Patterns for Data Manipulation. Treading on Python, 2021. 377 p.

17. Brett Slatkin. Effective Python: 90 Specific Ways to Write Better Python. Addison-Wesley Professional, 2019. 480 p.

18. Andriy Burkov. The Hundred-Page Machine Learning Book. 2019. 160 p.

19. Scott M. Lundberg, Su-In Lee. A Unified Approach to Interpreting Model Predictions. Neural Information Processing Systems (NIPS), 2017. 10 p.

20. Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. ACM KDD, 2016. 10 p.

21. Ramprasaath R. Selvaraju. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. ICCV, 2017. 9 p.

22. Kaiming He, Xiangyu Zhang, Shaoqing Ren. Deep Residual Learning for Image Recognition (ResNet). CVPR, 2016. 12 p.

23. Cynthia Rudin. Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. Nature Machine Intelligence, 2019. Vol. 1. P. 206–215.

24. Finale Doshi-Velez, Been Kim. Towards A Rigorous Science of Interpretable Machine Learning. arXiv preprint arXiv:1702.08608, 2017. 13 p.

25. Amina Adadi, Mohammed Berrada. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). IEEE Access, 2018. Vol. 6. P. 52138–52160.

26. European Commission. Ethics Guidelines for Trustworthy AI. High-Level Expert Group on Artificial Intelligence, 2019. 41 p.

27. SHAP Documentation: вебсайт. URL: https://shap.readthedocs.io/en/latest/ (дата звернення: 21.07.2025).

28. LIME Documentation: вебсайт. URL: https://lime-ml.readthedocs.io/en/latest/ (дата звернення: 22.07.2025).

29. Streamlit Docs: вебсайт. URL: https://docs.streamlit.io/ (дата звернення: 25.07.2025).

30. PyTorch Documentation: вебсайт. URL: https://pytorch.org/docs/stable/index.html (дата звернення: 05.08.2025).

31. Scikit-learn User Guide: вебсайт. URL: https://scikit-learn.org/stable/user_guide.html (дата звернення: 12.08.2025).

32. OpenCV Documentation: вебсайт. URL: https://docs.opencv.org/4.x/ (дата звернення: 20.08.2025).

33. Pandas Documentation: вебсайт. URL: https://pandas.pydata.org/docs/ (дата звернення: 15.09.2025).

34. The EU Artificial Intelligence Act: вебсайт. URL: https://artificialintelligenceact.eu/ (дата звернення: 28.09.2025).

35. FDA. Artificial Intelligence and Machine Learning (AI/ML)-Enabled Medical Devices: вебсайт. URL: https://www.fda.gov/medical-devices (дата звернення: 10.10.2025).

36. Breast Cancer Wisconsin (Diagnostic) Data Set: вебсайт. URL: https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic) (дата звернення: 22.10.2025).

37. Chest X-Ray Images (Pneumonia) Dataset: вебсайт. URL: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia (дата звернення: 22.10.2025).

38. Captum Model Interpretability Library: вебсайт. URL: https://captum.ai/ (дата звернення: 05.11.2025).

39. Python 3.12 Documentation: вебсайт. URL: https://docs.python.org/3/ (дата звернення: 15.11.2025).

ДОДАТОК А

Програмний код застосунку

src/app.py

```
python
import streamlit as st
import pandas as pd
import numpy as np
import joblib
import shap
import matplotlib.pyplot as plt
from PIL import Image
import torch
from torchvision import transforms, models
import cv2
from xai_utils import get_shap_explainer, calculate_shap_values, get_lime_explainer, explain_instance_lime
from gradcam_utils import GradCAM, overlay_cam
# Set page config
st.set_page_config(page_title="XAI Medical Diagnosis", layout="wide")
@st.cache_resource
def load_tabular_resources():
    # Load models
    models = {
        'Logistic Regression': joblib.load('models/logistic_regression.pkl'),
        'Random Forest': joblib.load('models/random_forest.pkl'),
        'XGBoost': joblib.load('models/xgboost.pkl')
    }
    # Load scaler
    scaler = joblib.load('models/scaler.pkl')
    # Load data
    X_test = pd.read_csv('models/X_test.csv')
    y_test = pd.read_csv('models/y_test.csv')
    return models, scaler, X_test, y_test
@st.cache_resource
def load_image_model():
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    model = models.resnet18(pretrained=False)
    num_ftrs = model.fc.in_features
    # Assuming 2 classes: NORMAL, PNEUMONIA
    model.fc = torch.nn.Linear(num_ftrs, 2)

    # Load weights if exist, else return None (or handle gracefully)
    try:
        model.load_state_dict(torch.load('models/resnet18_chest_xray.pth', map_location=device))
    except FileNotFoundError:
        return None

    model = model.to(device)
    model.eval()
```

```
    return model
tabular_models, scaler, X_test, y_test = load_tabular_resources()
image_model = load_image_model()
st.title("     Intelligent Medical Diagnostic System (XAI)")
st.markdown("""
This system uses Machine Learning to predict breast cancer diagnosis and **Explainable AI (XAI)** to interpret the results.
Now includes **Chest X-Ray Pneumonia Detection**!
""")
# Modality Selection
modality = st.sidebar.radio("Select Modality", ["Tabular Diagnosis (Breast Cancer)", "Image Diagnosis (Pneumonia)"])
if modality == "Tabular Diagnosis (Breast Cancer)":
    # Sidebar
    st.sidebar.header("Configuration")
    model_name = st.sidebar.selectbox("Select Model", list(tabular_models.keys()))
    selected_model = tabular_models[model_name]
    # Patient Selection
    st.sidebar.subheader("Patient Data")
    patient_index = st.sidebar.number_input("Select Patient Index (from Test Set)", min_value=0, max_value=len(X_test)-1, value=0)
    # Get patient data
    patient_data = X_test.iloc[patient_index]
    actual_diagnosis = "Malignant" if y_test.iloc[patient_index].values[0] == 1 else "Benign"
    st.sidebar.write(f"**Actual Diagnosis:** {actual_diagnosis}")
    st.sidebar.dataframe(patient_data)
    # Main Content
    col1, col2 = st.columns([1, 2])
    with col1:
        st.subheader("Prediction")

        # Preprocess input if needed
        if model_name == 'Logistic Regression':
            input_data = scaler.transform([patient_data])
        else:
            input_data = [patient_data]

        # Predict
        prediction = selected_model.predict(input_data)[0]
        probability = selected_model.predict_proba(input_data)[0][1]

        pred_label = "Malignant" if prediction == 1 else "Benign"
        color = "red" if prediction == 1 else "green"

        st.markdown(f"<h2 style='color:{color}'>{pred_label}</h2>", unsafe_allow_html=True)
        st.metric("Probability of Malignancy", f"{probability:.2%}")

        st.info("Navigate to the tabs on the right to see WHY this prediction was made.")
    with col2:
        st.subheader("Model Interpretation")
        tab1, tab2, tab3 = st.tabs(["SHAP (Local)", "LIME (Local)", "Global Importance"])
```

```
        with tab1:
            st.markdown("### SHAP Explanation")
            st.write("SHAP values show the contribution of each feature to the prediction.")

            # Calculate SHAP
            explainer = get_shap_explainer(selected_model, X_test)

            # SHAP values for this instance
            if model_name == 'Logistic Regression':
                shap_values = explainer.shap_values(input_data)
            else:
                shap_values = explainer.shap_values(pd.DataFrame([patient_data]))

            # Handle shape issues
            if isinstance(shap_values, list):
                sv = shap_values[1][0]
            elif len(shap_values.shape) == 3:
                sv = shap_values[0, :, 1]
            else:
                sv = shap_values[0]
            # Waterfall plot
            try:
                fig, ax = plt.subplots()
                shap.plots.waterfall(shap.Explanation(values=sv,
base_values=explainer.expected_value[1] if isinstance(explainer.expected_value, list) or
isinstance(explainer.expected_value, np.ndarray) else explainer.expected_value, data=patient_data,
feature_names=X_test.columns), show=False)
                st.pyplot(plt.gcf())
                plt.clf()
            except Exception as e:
                st.error(f"Could not generate waterfall plot: {e}")
                st.write("Fallback to standard bar plot:")
                fig, ax = plt.subplots()
                shap.summary_plot(shap_values, pd.DataFrame([patient_data]), plot_type="bar",
show=False)
                st.pyplot(fig)
                plt.clf()
        with tab2:
            st.markdown("### LIME Explanation")
            st.write("LIME fits a local linear model around the instance to explain it.")

            lime_explainer = get_lime_explainer(X_test, X_test.columns)
            exp = explain_instance_lime(lime_explainer, patient_data,
selected_model.predict_proba)

            # Display LIME plot
            fig = exp.as_pyplot_figure()
            st.pyplot(fig)
            plt.clf()
        with tab3:
```

```
            st.markdown("### Global Feature Importance")
            st.write("Which features are most important for the model overall?")

            if model_name == 'Logistic Regression':
                # Coefficients
                coefs = pd.DataFrame({'Feature': X_test.columns, 'Coefficient': selected_model.coef_[0]})
                coefs = coefs.sort_values(by='Coefficient', ascending=False)
                st.bar_chart(coefs.set_index('Feature'))
            else:
                # Feature importance
                importances = pd.DataFrame({'Feature': X_test.columns, 'Importance': selected_model.feature_importances_})
                importances = importances.sort_values(by='Importance', ascending=False).head(10)
                st.bar_chart(importances.set_index('Feature'))
elif modality == "Image Diagnosis (Pneumonia)":
    st.sidebar.header("Image Upload")
    uploaded_file = st.sidebar.file_uploader("Upload Chest X-Ray", type=["jpg", "jpeg", "png"])

    if image_model is None:
        st.warning("Image model not found. Please train the model first.")
    elif uploaded_file is not None:
        # Preprocess image
        image = Image.open(uploaded_file).convert('RGB')

        # Transform for model
        transform = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
        ])

        img_tensor = transform(image).unsqueeze(0)
        device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
        img_tensor = img_tensor.to(device)

        # Predict
        with torch.no_grad():
            output = image_model(img_tensor)
            probs = torch.nn.functional.softmax(output, dim=1)
            pred_idx = torch.argmax(probs, dim=1).item()

        classes = ['NORMAL', 'PNEUMONIA']
        pred_label = classes[pred_idx]
        prob = probs[0][pred_idx].item()

        col1, col2 = st.columns(2)

        with col1:
            st.subheader("Original Image")
```

```
            st.image(image, width="stretch") # Fixed deprecation warning

            color = "red" if pred_label == "PNEUMONIA" else "green"
            st.markdown(f"<h2 style='color:{color}'>{pred_label}</h2>",
unsafe_allow_html=True)
            st.metric("Confidence", f"{prob:.2%}")

        with col2:
            st.subheader("Grad-CAM Explanation")
            st.write("Heatmap highlights regions important for the prediction.")

            # Target Class Selector
            target_class_option = st.selectbox(
                "Explain Class:",
                ["Predicted Class", "NORMAL", "PNEUMONIA"]
            )

            if target_class_option == "Predicted Class":
                target_idx = pred_idx
            elif target_class_option == "NORMAL":
                target_idx = 0
            else:
                target_idx = 1

            # Generate Grad-CAM
            target_layer = image_model.layer4[-1]
            grad_cam = GradCAM(image_model, target_layer)

            cam = grad_cam(img_tensor, class_idx=target_idx)

            # Overlay on ORIGINAL image
            original_np = np.array(image)
            overlay_img = overlay_cam(original_np, cam)

            st.image(overlay_img, caption=f"Grad-CAM for {classes[target_idx]}",
width="stretch")
```

src/gradcam_utils.py

```
python
import torch
import torch.nn.functional as F
import numpy as np
import cv2
class GradCAM:
    """
    Grad-CAM implementation for ResNet-like architectures.
    """
    def __init__(self, model, target_layer):
        self.model = model
        self.target_layer = target_layer
        self.gradients = None
```

```
        self.activations = None

        # Hook for gradients
        target_layer.register_backward_hook(self.save_gradient)
        # Hook for activations
        target_layer.register_forward_hook(self.save_activation)

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

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

    def __call__(self, x, class_idx=None):
        # Forward pass
        output = self.model(x)

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

        # Zero grads
        self.model.zero_grad()

        # Backward pass
        one_hot = torch.zeros_like(output)
        one_hot[0][class_idx] = 1
        output.backward(gradient=one_hot, retain_graph=True)

        # Get gradients and activations
        gradients = self.gradients.data.cpu().numpy()[0]
        activations = self.activations.data.cpu().numpy()[0]

        # Global Average Pooling of gradients
        weights = np.mean(gradients, axis=(1, 2))

        # Weighted combination of activations
        cam = np.zeros(activations.shape[1:], dtype=np.float32)
        for i, w in enumerate(weights):
            cam += w * activations[i]

        # ReLU
        cam = np.maximum(cam, 0)

        # Resize to input image size (224x224)
        cam = cv2.resize(cam, (x.shape[2], x.shape[3]))

        # Normalize
        cam = cam - np.min(cam)
        cam = cam / (np.max(cam) + 1e-8)

        return cam
```

```
def overlay_cam(original_img, cam, alpha=0.5):
    """
    Overlays the CAM heatmap on the original image.
    """
    # Ensure original_img is float [0, 1]
    img = original_img.astype(np.float32)
    if img.max() > 1:
        img /= 255.0

    # Resize CAM to match original image size
    h, w = img.shape[:2]
    cam_resized = cv2.resize(cam, (w, h))

    # Heatmap
    heatmap = cv2.applyColorMap(np.uint8(255 * cam_resized), cv2.COLORMAP_JET)
    heatmap = np.float32(heatmap) / 255
    heatmap = heatmap[..., ::-1] # BGR to RGB

    # Overlay
    overlayed = heatmap * alpha + img * (1 - alpha)
    overlayed = np.clip(overlayed, 0, 1)

    return overlayed
```

src/xai_utils.py

```
python
import shap
import lime
import lime.lime_tabular
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
def get_shap_explainer(model, X_train):
    """
    Returns the appropriate SHAP explainer based on the model type.
    """
    model_type = type(model).__name__

    if model_type in ['RandomForestClassifier', 'XGBClassifier']:
        return shap.TreeExplainer(model)
    elif model_type == 'LogisticRegression':
        return shap.LinearExplainer(model, X_train)
    else:
        return shap.KernelExplainer(model.predict_proba, shap.kmeans(X_train, 10))
def calculate_shap_values(explainer, X_instance):
    """
    Calculates SHAP values for a single instance or a batch.
    """
    shap_values = explainer.shap_values(X_instance)

    # Handle different return types of shap_values (list for classifiers vs array)
```

```
    if isinstance(shap_values, list):
        return shap_values[1]
    return shap_values
def get_lime_explainer(X_train, feature_names, class_names=['Benign', 'Malignant']):
    """
    Initializes a LIME Tabular Explainer.
    """
    explainer = lime.lime_tabular.LimeTabularExplainer(
        training_data=np.array(X_train),
        feature_names=feature_names,
        class_names=class_names,
        mode='classification'
    )
    return explainer
def explain_instance_lime(explainer, instance, model_predict_proba):
    """
    Generates a LIME explanation for a single instance.
    """
    exp = explainer.explain_instance(
        data_row=np.array(instance),
        predict_fn=model_predict_proba
    )
    return exp
```

src/train_image_model.py

```
python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models
import time
import os
import copy
from image_loader import download_image_data, get_data_loaders
def train_model(model, dataloaders, criterion, optimizer, num_epochs=5, device='cpu'):
    since = time.time()
    val_acc_history = []
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print('-' * 10)

        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
```

```
                running_corrects = 0

                for inputs, labels in dataloaders[phase]:
                    inputs = inputs.to(device)
                    labels = labels.to(device)

                    optimizer.zero_grad()

                    with torch.set_grad_enabled(phase == 'train'):
                        outputs = model(inputs)
                        loss = criterion(outputs, labels)
                        _, preds = torch.max(outputs, 1)

                        if phase == 'train':
                            loss.backward()
                            optimizer.step()

                    running_loss += loss.item() * inputs.size(0)
                    running_corrects += torch.sum(preds == labels.data)

                epoch_loss = running_loss / len(dataloaders[phase].dataset)
                epoch_acc = running_corrects.double() / len(dataloaders[phase].dataset)

                print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')

                if phase == 'val' and epoch_acc > best_acc:
                    best_acc = epoch_acc
                    best_model_wts = copy.deepcopy(model.state_dict())
                if phase == 'val':
                    val_acc_history.append(epoch_acc)

        time_elapsed = time.time() - since
        print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s')
        print(f'Best val Acc: {best_acc:4f}')
        model.load_state_dict(best_model_wts)
        return model, val_acc_history
    def main():
        device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
        path = download_image_data()
        train_loader, val_loader, test_loader, classes = get_data_loaders(path, batch_size=16)
        dataloaders = {'train': train_loader, 'val': val_loader}

        model = models.resnet18(pretrained=True)
        num_ftrs = model.fc.in_features
        model.fc = nn.Linear(num_ftrs, len(classes))
        model = model.to(device)

        criterion = nn.CrossEntropyLoss()
        optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

        model, hist = train_model(model, dataloaders, criterion, optimizer, num_epochs=3,
```

```
device=device)

    os.makedirs('models', exist_ok=True)
    torch.save(model.state_dict(), 'models/resnet18_chest_xray.pth')

    # Evaluate on test set
    model.eval()
    running_corrects = 0
    for inputs, labels in test_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)
        with torch.no_grad():
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
        running_corrects += torch.sum(preds == labels.data)

    acc = running_corrects.double() / len(test_loader.dataset)
    print(f'Test Acc: {acc:.4f}')
if __name__ == "__main__":
    main()
```

src/train_model.py

```
python
import pandas as pd
import numpy as np
import os
import joblib
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score
from data_loader import load_data, preprocess_data
def train_models():
    print("Loading and preprocessing data...")
    df = load_data()
    df = preprocess_data(df)

    X = df.drop('diagnosis', axis=1)
    y = df['diagnosis']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    os.makedirs('models', exist_ok=True)
    joblib.dump(scaler, 'models/scaler.pkl')
```

```
    models = {
        'Logistic Regression': LogisticRegression(random_state=42),
        'Random Forest': RandomForestClassifier(random_state=42),
        'XGBoost': XGBClassifier(use_label_encoder=False, eval_metric='logloss',
random_state=42)
    }

    results = {}

    print("\nTraining models...")
    for name, model in models.items():
        if name == 'Logistic Regression':
            model.fit(X_train_scaled, y_train)
            preds = model.predict(X_test_scaled)
            probs = model.predict_proba(X_test_scaled)[:, 1]
        else:
            model.fit(X_train, y_train)
            preds = model.predict(X_test)
            probs = model.predict_proba(X_test)[:, 1]

        acc = accuracy_score(y_test, preds)
        auc = roc_auc_score(y_test, probs)
        results[name] = {'Accuracy': acc, 'AUC': auc}

        joblib.dump(model, f"models/{name.lower().replace(' ', '_')}.pkl")

    X_test.to_csv('models/X_test.csv', index=False)
    y_test.to_csv('models/y_test.csv', index=False)
    return results
if __name__ == "__main__":
    train_models()

src/image_loader.py
python
import kagglehub
import os
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, Subset
import numpy as np
def download_image_data():
    print("Downloading Chest X-Ray dataset...")
    path = kagglehub.dataset_download("paultimothymooney/chest-xray-pneumonia")
    return path
def get_data_loaders(data_dir, batch_size=32, img_size=224):
    transform = transforms.Compose([
        transforms.Resize((img_size, img_size)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])
```

```
if os.path.exists(os.path.join(data_dir, 'chest_xray')):
        data_dir = os.path.join(data_dir, 'chest_xray')
if os.path.exists(os.path.join(data_dir, 'chest_xray')):
          data_dir = os.path.join(data_dir, 'chest_xray')
train_dataset = datasets.ImageFolder(os.path.join(data_dir, 'train'), transform=transform)
val_dataset = datasets.ImageFolder(os.path.join(data_dir, 'val'), transform=transform)
test_dataset = datasets.ImageFolder(os.path.join(data_dir, 'test'), transform=transform)

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

return train_loader, val_loader, test_loader, train_dataset.classes
```