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

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

ІНФОРМАЦІЙНА СИСТЕМА ДІАГНОСТИКИ ХВОРОБ СІЛЬСЬКОГОСПОДАРСЬКИХ КУЛЬТУР ІЗ ВИКОРИСТАННЯМ ТРАНСФОРМЕРІВ ЗОРУ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Інтелектуальні інформаційні системи»

Здобувач _____ Данило МУРЗАКОЙ

«___»_____ 2025 р.

Керівник канд. пед. наук, доцент _____ Надія БОЛЮБАШ

«___»_____ 2025 р.

Миколаїв – 2025

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

Факультет	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

Завідувач кафедри інтелектуальних інформаційних систем

_____ Юрій КОНДРАТЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

_____ Мурзакоя Данила Васильовича_____
((прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Інформаційна система діагностики хвороб сільськогосподарських культур із використанням трансформерів зору».

Керівник роботи: Болюбаш Надія Миколаївна, доцент кафедри інтелектуальних інформаційних систем, канд. пед. наук, доцент.

Затверджена наказом ЧНУ ім. Петра Могили від «7» липня 2025 р. № 184.

2. Строк представлення кваліфікаційної роботи «16» грудня 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: інформаційна система діагностики хвороб сільськогосподарських культур із використанням трансформерів зору (Vision Transformers, ViT) для фермерських господарств, яка забезпечує автоматичне розпізнавання стану листя рослин за зображенням та класифікацію його як здорового або ураженого певною хворобою на основі

механізму самоуваги моделей ViT; набори даних із зображеннями листя різних видів сільськогосподарських рослин, класифікованих за їх станом здоров’я.

4. Перелік питань, що підлягають розробці: дослідження теоретичних засад діагностики хвороб сільськогосподарських культур, аналіз існуючих досліджень та програмних рішень для автоматизованого виявлення хвороб рослин в аграрній галузі; обґрунтування вибору моделі Vision Transformer, набору даних і методів для її донавчання та інструментальних засобів розробки інформаційної системи; проведення донавчання моделі ViT на підготовленому наборі даних та дослідження налаштувань її гіперпараметрів для отримання високої точності розпізнавання хвороб; моделювання, проєктування, здійснення програмної реалізації системи діагностики хвороб сільськогосподарських культур та оцінка її якості.

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

Керівник роботи _____ Надія БОЛЮБАШ
(Особистий підпис) *(Власне ім’я ПРІЗВИЩЕ)*

Здобувач _____ Данило МУРЗАКОЙ
(Особистий підпис) *(Власне ім’я ПРІЗВИЩЕ)*

Дата видачі завдання «07» липня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Інформаційна система діагностики хвороб сільськогосподарських культур із використанням трансформерів зору

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	29.06.2025	30.06.2025	Виконано
2	Аналіз предметної області та постановка задачі	1.09.2025	10.09.2025	Виконано
3	Огляд літературних джерел за темою кваліфікаційної роботи, аналіз публікацій та існуючих підходів до автоматизації ідентифікації і класифікації хвороб рослин за зображеннями листя	11.09.2025	21.09.2025	Виконано
4	Огляд існуючих моделей ViT та наборів даних і методів для їх донавчання, аналіз та вибір стеку технологій розробки	22.09.2025	25.10.2025	Виконано
5	Реалізація обраних технологій із аналізом отриманих результатів	26.10.2025	24.11.2025	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	25.11.2025	25.11.2025	Виконано
7	Корегування роботи за результатами попереднього захисту	26.11.2025	08.12.2025	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	09.12.2025	09.12.2025	Виконано
9	Доробка та остаточне оформлення КР	10.12.2025	13.02.2025	Виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	14.12.2025	15.12.2025	Виконано

Керівник роботи _____ Надія БОЛЮБАШ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Данило МУРЗАКОЙ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«7» липня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи здобувача групи 601м ЧНУ ім. П. Могили

Мурзакой Данила Васильовича

на тему: «**ІНФОРМАЦІЙНА СИСТЕМА ДІАГНОСТИКИ ХВОРОБ СІЛЬСЬКОГОСПОДАРСЬКИХ КУЛЬТУР ІЗ ВИКОРИСТАННЯМ ТРАНСФОРМЕРІВ ЗОРУ**»

Кваліфікаційна робота присвячена розробці та програмній реалізації інформаційної системи, яка здатна автоматизовано ідентифікувати хвороби сільськогосподарських рослин на основі аналізу зображень листів із використанням технологій штучного інтелекту – трансформерів зору. Що дозволяє вирішити актуальну проблему своєчасної діагностики хвороб малими та середніми фермерськими господарствами без залучення висококваліфікованих експертів.

Об’єкт дослідження – процес діагностики хвороб сільськогосподарських культур.

Предмет дослідження – моделі трансформерів зору, методи їх навчання для класифікації захворювань рослин та програмні засоби їх реалізації у вигляді інформаційних систем.

Мета дослідження – підвищення точності розпізнавання хвороб сільськогосподарських рослин та інтерпретованості результатів за допомогою карт уваги шляхом створення інформаційної системи із використанням механізму самоуваги трансформерів зору.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків та додатків. У першому розділі проаналізовано теоретичні засади діагностики хвороб сільськогосподарських культур. У другому розділі розкрито сучасні моделі і методи комп’ютерного зору, обґрунтовано вибір моделі трансформера зору ViT. У третьому розділі описано створення і донавчання на спеціалізованому Data Set моделі ViT та дослідження з метою підбору оптимальних гіперпараметрів. У четвертому розділі описано стек технологій розробки інформаційної системи діагностики хвороб сільськогосподарських культур, її моделювання, проєктування,а програмну реалізацію інформаційної системи та оцінку якості.

Кваліфікаційна робота містить 100 сторінок, 20 рисунків, 11 таблиць, 42 джерела, 4 додатки.

Ключові слова: *трансформери зору*, *механізм самоуваги, багатоголова самоувага, карти уваги, підбір параметрів, класифікаційна голова*.

ABSTRACT

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

Murzakoi Danyla Vasylovycha

on the subject: «**INFORMATION SYSTEM FOR DIAGNOSTICS OF AGRICULTURAL CROPS DISEASES USING VISION TRANSFORMERS**»

The master's qualification work is devoted to the development and software implementation of an information system capable of automatically identifying diseases of agricultural plants based on leaf image analysis using artificial intelligence technologies - vision transformers. This allows solving the urgent problem of timely diagnosis of diseases by small and medium-sized farms without involving highly qualified experts.

Object of research – the process of diagnosing crop diseases.

Subject of research – models of vision transformers, methods of their training for the classification of plant diseases and software tools for their implementation in the form of information systems.

The purpose of the study is to increase the accuracy of recognizing agricultural plant diseases and the interpretability of results using attention maps by creating an information system using the self-attention mechanism of vision transformers.

The master's qualification work consists of an introduction, four sections, conclusions and appendices. The first section analyzes the theoretical foundations of crop disease diagnostics. The second section reveals modern models and methods of computer vision, justifies the choice of the ViT vision transformer model. The third section describes the creation and further training of the ViT model on a specialized Data Set and research to select optimal hyperparameters. The fourth section describes the stack of technologies for developing an information system for crop disease diagnostics, its modeling, design, and software implementation of the information system and assessed its quality.

The master’s thesis contains 100 pages, 20 figures, 11 tables, 42 sources, 4 appendices.

Key words: *vision transformers, self-attention, multi-head self-attention, attention maps, fine-tuning, classification head.*

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ 4
ВСТУП 5
1 ТЕОРЕТИЧНІ ЗАСАДИ ДІАГНОСТИКИ ХВОРОБ СІЛЬСЬКОГОСПОДАРСЬКИХ КУЛЬТУР 8
1.1 Предметна сфера діагностики хвороб рослин у агроіндустрії 8
1.2 Традиційні методи та сучасні підходи до виявлення хвороб сільськогосподарських культур 12
1.3 Аналіз досліджень та публікацій застосування Vision Transformers для розпізнавання хвороб рослин 16
1.4 Огляд програмних рішень для діагностики хвороб у аграрній галузі 20
1.5 Проблеми та обмеження існуючих підходів 23
1.6 Постановка задачі 26
Висновки до розділу 1 29
2 ДІАГНОСТИКА ХВОРОБ СІЛЬСЬКОГОСПОДАРСЬКИХ КУЛЬТУР ІЗ ВИКОРИСТАННЯМ МОДЕЛЕЙ VISION TRANSFORMERS 31
2.1 Архітектура моделей Vision Transformer 31
2.2 Механізм багатоголової уваги моделей ViT 34
2.3 Методи попередньої обробки та аугментації зображень 37
2.3 Fine-tuning моделей ViT для класифікації хвороб 41
2.4 Метрики оцінки якості моделей Vision Transformers 47
Висновки до розділу 2 51
3 НАВЧАННЯ ТА ОЦІНКА ЯКОСТІ МОДЕЛЕЙ VISION TRANSFORMER 53
3.1 Підготовка даних і моделі до навчання 53
3.2 Базова моделі Vision Transformer та стратегії її донавчання 56
3.3 Метод підбору гіперпараметрів моделі Random Search 58
3.4 Методика дослідження впливу гіперпараметрів та стратегій донавчання на якість класифікації 62

3.3 Аналіз отриманих результатів 67
Висновки до розділу 3 75
4 РОЗРОБКА ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДІАГНОСТИКИ ХВОРОБ СІЛЬСЬКОГОСПОДАРСЬКИХ КУЛЬТУР..... 77
4.1 Моделювання та проєктування інформаційної системи..... 77
4.2 Інструментальні засоби розробки..... 81
4.3 Реалізація серверної частини: Node.js API-шлюз та Python FastAPI сервіс інференсу..... 84
4.4 Реалізація клієнтської частини на React/Vite та взаємодія з REST API 86
4.5 Програмна реалізація вебзастосунку 88
4.6 Інтерфейс користувача та сценарій роботи з системою 96
4.7 Тестування та оцінка якості інформаційної системи 102
Висновки до розділу 4 104
ВИСНОВКИ..... 106
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ..... 109
ДОДАТОК А Характеристика датасет для донавчання базової моделі ViT 114
ДОДАТОК Б JavaScript-скрипт upload.js для визначення серверного маршруту зображення..... 115
ДОДАТОК В JavaScript-скрипт App.jsx для формування макету сторінки та розміщення блоків..... 116
ДОДАТОК Г JavaScript-скрипт UploadForm.jsx для завантаження файлу та відображення результатів класифікації 118

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ШІ – штучний інтелект

ІС – інформаційна система

AI – Artificial Intelligence

ViT – Vision transformer

CNN – Convolutional Neural Networks

ML – Machine Learning

ВСТУП

Актуальність. В умовах розбудови інформаційного суспільства передові технології штучного інтелекту (ШІ) значно покращують сільськогосподарську практику. Вирішальне значення у балансуванні економічних, соціальних і екологічних аспектів для підтримки прийняття рішень та довгострокової продуктивності у сфері агроіндустрії має виявлення хвороб сільськогосподарських рослин. Методи комп'ютерного зору, такі як трансформери зору (англ. Vision transformers, ViTs), продемонстрували значний потенціал в автоматизації виявлення хвороб рослин, забезпечуючи їх ранню та точну ідентифікацію. Що дозволяє вирішити актуальну проблему своєчасної діагностики хвороб малими та середніми фермерськими господарствами без залучення висококваліфікованих експертів.

Традиційні методи виявлення хвороб спираються на лабораторні методи та візуальний огляд рослин експертами, який включає обстеження на наявність видимих симптомів хвороб, таких як ураження, зміна кольору, деформації тощо. Ці методи мають обмеження щодо масштабованості та ефективності, оскільки потребують спеціалізованого обладнання та навченого персоналу. Технології комп'ютерного зору дозволяють автоматизувати ідентифікацію хвороб шляхом аналізу зображень рослин, забезпечуючи більш ефективну та менш трудомістку альтернативу ручному моніторингу.

Значно підвищує точність діагностики хвороб сільськогосподарських культур використання згорткових нейронних мереж (англ. Convolutional Neural Networks, CNN), які є безумовним лідером у класифікації зображень рослин. Що дозволяє автоматизувати процеси ідентифікації та класифікації хвороб за зображеннями листя. Однак більшість моделей CNN мають обмежену здатність до узагальнення та інтерпретації і низьку гнучкість при перенавчанні на нових даних. Застосування трансформерів зору завдяки механізму самоуваги (англ. Self-Attention) забезпечує кращу продуктивність при великій кількості класів хвороб та

дозволяє більш ефективно пояснювати рішення ViT-моделі шляхом ідентифікації глобальних зв’язків між різними ділянками зображення.

Проблема полягає у відсутності доступних і точних автоматизованих інструментів для діагностики широкого спектра хвороб сільськогосподарських культур, здатних працювати в реальних умовах. Доступні на ринку програмні засоби, які сьогодні використовують для діагностики і розпізнавання хвороб сільськогосподарських культур, забезпечують швидке виявлення та масштабування моніторингу. Проте готові рішення більшою мірою орієнтовані на глобальні набори даних або корпоративні, комерційні інтереси. Моделі, навчені на чистих дата сетах, погано працюють у польових умовах без донавчання і потребують агрономічної верифікації. Тому доцільним є розробка системи, орієнтованої та невеликі фермерські господарства, яка створює рекомендації на основі точного розпізнання хвороб із використанням моделей, що дозволяють адаптуватися під нові культури або змінені умови із врахуванням локальних особливостей: сорти культур, регіональні патогени, агроклімат.

Метою дослідження є підвищення точності розпізнавання хвороб сільськогосподарських рослин та інтерпретованості результатів за допомогою карт уваги шляхом створення інформаційної системи із використанням механізму самоуваги трансформерів зору.

Досягнення поставленої мети обумовлює необхідність вирішення наступних **завдань**:

– дослідити теоретичні засади діагностики хвороб сільськогосподарських культур, проаналізувати існуючих досліджень та програмних рішень для автоматизованого виявлення хвороб рослин в аграрній галузі;

– обґрунтувати вибір моделі Vision Transformer, набору даних і методів для її донавчання та інструментальних засобів розробки інформаційної системи;

– провести донавчання моделі ViT на підготовленому наборі даних та дослідити налаштування її гіперпараметрів для отримання високої точності розпізнавання хвороб;

– здійснити моделювання, проєктування, програмну реалізацію системи діагностики хвороб сільськогосподарських культур та оцінити її якість.

Об’єктом дослідження є процес діагностики хвороб сільськогосподарських культур.

Предметом дослідження є моделі трансформерів зору, методи їх навчання для класифікації захворювань рослин та програмні засоби їх реалізації у вигляді інформаційних систем.

Методологічною основою дослідження є загальнонаукові аналітичні методи, методи комп’ютерного зору, глибинного та машинного навчання, методи попередньої обробки та аугментації зображень, методи оцінки точності прогнозу, які дозволили вивчити предмет і об’єкт дослідження, дослідити розвиток науково-методичних засад, напрямів та шляхів підвищення точності розпізнавання хвороб та їх інтерпретованості.

Результати дослідження обговорювалися на XXVIII Всеукраїнській науково-практичній конференції «Могилянські читання – 2025: Досвід та тенденції розвитку суспільства в Україні: глобальний, національний та регіональний аспекти» (10-14 листопада 2025 року) та отримали схвалення.

Практичне значення отриманих результатів полягає в тому, що розроблена інформаційна система може використовуватися фермерами, агрономами та дослідницькими установами для автоматизованої швидкої і точної діагностики й інтепретованості хвороб сільськогосподарських культур у аграрній галузі, що є особливо важливим для невеликих та середніх фермерських господарств. Система сприяє зменшенню втрат урожаю, скороченню витрат на лабораторні дослідження, оптимізації використання засобів захисту рослин та розвитку концепції розумного землеробства (англ. Smart Agriculture).

Структура кваліфікаційної роботи. Відповідно до мети, завдань і предмета дослідження кваліфікаційна робота складається із вступу, чотирьох розділів, висновку, списку використаних джерел та 4 додатків. Загальний обсяг кваліфікаційної роботи – 100 сторінок, кількість використаних джерел – 42.

1 ТЕОРЕТИЧНІ ЗАСАДИ ДІАГНОСТИКИ ХВОРОБ СІЛЬСЬКОГОСПОДАРСЬКИХ КУЛЬТУР

1.1 Предметна сфера діагностики хвороб рослин у агроіндустрії

Сучасне сільське господарство є однією з ключових галузей економіки, від ефективності якої залежить продовольча безпека держави. Однією з найсерйозніших проблем аграрного виробництва є захворювання рослин, які призводять до значних втрат урожаю, погіршення якості продукції та збільшення витрат на її вирощування [1]. Традиційні методи діагностики хвороб культурних рослин потребують участі фахівців-агрономів, займають багато часу та залежать від людського фактору, що ускладнює їхнє застосування на великих площах.

Ефективність агровиробництва значною мірою залежить від здатності своєчасно виявляти та запобігати розвитку хвороб сільськогосподарських культур. За оцінками міжнародних аграрних організацій, втрати врожаю через різноманітні патології можуть досягати до 40 % загального обсягу виробництва, а у випадках епідемічного поширення - навіть більше.

Причинами ураження рослин є широкий спектр біотичних (грибкових, бактеріальних, вірусних) та абіотичних (нестача елементів живлення, стресові погодні умови, забруднення) чинників. Візуальні прояви захворювань зазвичай відображаються на листках у вигляді плям, змін кольору, деформацій, некрозів чи в’янення (рис. 1.1, рис. 1.2). Традиційна діагностика базується на візуальному огляді агрономом або фітопатологом, іноді – на лабораторному аналізі. Такий підхід є ефективним лише в обмежених масштабах і вимагає значних людських ресурсів та часу.

З розвитком технологій штучного інтелекту (англ. Artificial Intelligence, AI), комп’ютерного зору та машинного навчання (англ. Machine Learning, ML) з’явилися можливості для автоматизації процесів розпізнавання хвороб за зображеннями листя. Автоматизована діагностика дає змогу оперативно визначати

тип захворювання, підвищує точність оцінки стану рослин і знижує залежність від людського фактора.

Рисунок 1.1 – Візуальне зображення листа хворої рослини

Рисунок 1.2 – Листя хворої рослини у польових умовах

Предметна сфера таких систем охоплює сукупність методів і технологій, що забезпечують:

– збір та підготовку даних - створення навчальних датасетів із зображень листя культур;

– обробку та нормалізацію зображень - усунення шумів, вирівнювання освітлення, масштабування;

– навчання моделей глибокого навчання для розпізнавання типів хвороб;

– інтеграцію результатів у інформаційні системи та веб-застосунки для кінцевих користувачів - фермерів, агрономів, дослідників.

На сьогодні поширеними є системи, побудовані на згорткових нейронних мережах CNN – таких як ResNet, EfficientNet або MobileNet. Проте останніми роками у світовій науці відбувся перехід до використання трансформерів зору (Vision Transformers, ViT) - архітектур, що походять з успішних мовних моделей (табл. 1.1). ViT моделі використовують механізм самоуваги (self-attention), який дозволяє ефективно аналізувати глобальні просторові зв’язки між фрагментами зображення [2]. Це робить їх особливо корисними у випадках, коли симптоми хвороби проявляються в різних ділянках листка або мають складну текстурну структуру.

У наукових дослідженнях останніх років доведено, що ViT-моделі та їх гібриди з CNN забезпечують вищу точність класифікації у порівнянні з класичними підходами, особливо у задачах із малою кількістю розмічених зразків або за умов польових знімків із шумом та неоднорідним фоном [2-4]. Крім того, використання self-supervised методів (MAE, DINO) і fine-tuning на локальних наборах даних дозволяє створювати високоробустні рішення для практичного аграрного застосування.

Значення автоматизованої діагностики хвороб рослин полягає у:

– оперативному виявленні хвороб та можливості раннього втручання, що мінімізує втрати;

– зниженні потреби в експертному персоналі та скороченні часу обстеження;

– оптимізації використання засобів захисту рослин та скороченні витрат;

– підвищенні точності прогнозування стану посівів і планування агротехнічних заходів;

– екологічній безпеці завдяки точковому застосуванню препаратів.

Розвиток таких систем сприяє створенню інтелектуальних агроплатформ, здатних інтегрувати дані з дронів, супутників і мобільних пристроїв, формувати карти стану полів і забезпечувати підтримку прийняття рішень на основі аналізу реальних зображень. Власні дослідження та розробки у цьому напрямі мають стратегічне значення, адже дозволяють адаптувати моделі під локальні умови вирощування**,** специфічні сорти культур та регіональні патогени, що не охоплюються глобальними комерційними продуктами, такими як Plantix, Taranis, PlantVillage та інші [3].

Таблиця 1.1 – Порівняння ViT та CNN у розпізнаванні хвороб рослин

Ознака	Convolutional Neural Network	Vision Transformer
Принцип	Витягує локальні ознаки (через фільтри)	Аналізує глобальний контекст через самоувагу
Сфера бачення	Обмежена сусідніми пікселями	Кожен патч взаємодіє з усіма іншими
Навчання	Потребує великої кількості даних	Може донавчатися на менших наборах
Інтерпретація	«Чорна скринька»	Карти уваги показують, куди дивиться модель
Гнучкість	Важко адаптувати під нові культури	Легше fine-tuning під локальні умови
Точність	Висока у стабільному домені	Вища узагальнюваність між культурами/умовами

Отже, предметна сфера автоматизованої діагностики хвороб рослин є міждисциплінарною областю, що поєднує знання з агрономії, інформатики, машинного навчання та комп'ютерного зору. Її значення полягає у створенні технологічного підґрунтя для цифрової трансформації сільського господарства, підвищення продуктивності, ефективності використання ресурсів і забезпечення сталого розвитку агросектору.

1.2 Традиційні методи та сучасні підходи до виявлення хвороб сільськогосподарських культур

Діагностика хвороб рослин є одним із ключових процесів у системі управління агровиробництвом. Від своєчасності та точності визначення захворювання залежить ефективність заходів захисту культур, розмір економічних втрат і якість кінцевої продукції [4]. Протягом тривалого часу в аграрній практиці застосовувалися переважно традиційні методи визначення хвороб, що ґрунтувалися на візуальних спостереженнях та лабораторному аналізі. Однак стрімкий розвиток інформаційних технологій, сенсорики та штучного інтелекту зумовив перехід до сучасних автоматизованих підходів, які істотно підвищують швидкість і точність діагностики.

Традиційні методи діагностики. Класичні методи визначення хвороб рослин базуються на візуальній оцінці стану рослини фахівцем-агрономом або фітопатологом. Експерт аналізує морфологічні ознаки ураження – зміну кольору, форми, наявність плям, нальоту, некрозів, деформацій листя чи плодів [5]. Для уточнення діагнозу часто застосовують мікроскопічні та лабораторні дослідження, що дозволяють виявити збудників хвороб - грибки, бактерії, віруси або паразитів.

До основних традиційних підходів належать:

– візуальний фітопатологічний огляд - суб'єктивний метод, що залежить від досвіду спеціаліста;

– біохімічні та імунологічні тести (наприклад, ІФА, ПЛР-аналіз) - високоточні, але потребують лабораторних умов, реактивів і часу;

– морфометричні аналізи (визначення розмірів уражених ділянок, площі некрозу тощо);

– агрометеорологічне прогнозування ризику хвороб за температурою, вологістю, фазами розвитку культури.

Попри свою наукову обґрунтованість, традиційні методи мають низку недоліків: вони є трудомісткими, повільними, потребують значних людських і фінансових ресурсів, а результати часто залежать від кваліфікації спеціаліста. Крім того, вони не дозволяють забезпечити масштабний моніторинг стану посівів у реальному часі.

Сучасні підходи до автоматизованої діагностики. З появою технологій цифрової обробки зображень і машинного навчання з’явилася можливість переходу до автоматизованих систем розпізнавання хвороб рослин [6]. Сучасні методи базуються на аналізі фотографій листя, зібраних з полів, дронів, супутників або мобільних пристроїв, із подальшим використанням алгоритмів штучного інтелекту для визначення типу захворювання.

Основними напрямами розвитку автоматизованої діагностики є:

– методи комп’ютерного зору (Computer Vision) – перетворення, фільтрація, сегментація та аналіз візуальних ознак (текстура, колір, форма плям тощо);

– машинне навчання (Machine Learning) – використання класичних моделей (Support Vector Machine, Random Forest, k-NN) для класифікації за витягнутими ознаками;

– глибоке навчання (Deep Learning) – застосування згорткових нейронних мереж (CNN), які автоматично виділяють ознаки із зображення без ручного програмування;

– моделі трансформерів зору (Vision Transformers, ViT) – новий клас моделей, що використовують механізм самоуваги для аналізу просторових взаємозв’язків у зображенні.

Перші автоматизовані рішення у сфері аграрної діагностики базувалися на традиційних машинних алгоритмах, які вимагали ручного створення ознак (feature extraction). Пізніше з'явилися системи на базі CNN, що значно підвищили точність класифікації (до 95-98% на структурованих наборах PlantVillage). Сьогодні ж активно впроваджуються гібридні підходи CNN + ViT, які поєднують локальну чутливість CNN та глобальну контекстну увагу трансформерів.

Упродовж останнього десятиліття методи комп'ютерного зору (Computer Vision) та глибинного навчання (Deep Learning) стали ключовими технологічними складовими цифрової трансформації аграрного сектору. Вони забезпечують можливість автоматичного аналізу зображень рослин, визначення патологічних змін, оцінювання стану посівів і навіть прогнозування урожайності. Завдяки цьому сільське господарство поступово переходить від традиційних ручних процедур до інтелектуальних систем спостереження та прийняття рішень.

Комп'ютерний зір охоплює широкий спектр алгоритмів, які дозволяють комп'ютеру "бачити" та інтерпретувати об'єкти на зображеннях. Основними етапами обробки є фільтрація, сегментація, виділення ознак та класифікація. В аграрних застосуваннях ці методи використовують для: виявлення уражених ділянок листя (локалізація хвороб); підрахунку кількості плодів або листків; оцінки рівня стиглості чи дефіциту поживних речовин; аналізу структури ґрунту, бур'янів або шкідників.

Класичні алгоритми комп'ютерного зору, такі як SIFT, SURF, HOG, LBP, забезпечують виділення ключових ознак текстури, кольору та форми, які потім подаються на вхід моделей машинного навчання (SVM, k-NN, Random Forest) [7]. Хоча ці методи демонструють задовільні результати, вони мають обмежену здатність узагальнювати складні варіації зображень, що властиво реальним польовим умовам (різне освітлення, тіні, фонові об'єкти).

Поява глибоких нейронних мереж (Deep Neural Networks) стала переломним моментом у розвитку комп'ютерного зору. Завдяки можливості автоматичного виділення високорівневих ознак із сирих пікселів, глибоке навчання забезпечує

істотне підвищення точності розпізнавання хвороб рослин. Основу сучасних систем становлять згорткові нейронні мережі (Convolutional Neural Networks, CNN). Їхня архітектура дозволяє послідовно виділяти локальні патерни (краї, контури, плями), формуючи ієрархічні представлення ознак. Найбільш відомими є моделі AlexNet, VGGNet, ResNet, DenseNet, EfficientNet, які застосовуються для класифікації листя різних культур (яблуня, картопля, пшениця, кукурудза).

Приклади ефективного використання CNN в аграрній сфері включають:

– PlantVillage dataset - база з понад 50 000 зображень, на якій було створено численні CNN-моделі для ідентифікації понад 30 типів хвороб;

– PlantDoc dataset - зображення, зібрані в польових умовах із природним фоном, що стали основою для тестування моделей у реальних сценаріях;

– MobileNet та EfficientNet - легкі CNN-архітектури, оптимізовані для мобільних додатків (наприклад, Plantix, Nuru).

Останніми роками відбувся перехід від CNN до архітектур трансформерів зору (Vision Transformers, ViT), які завдяки механізму самоуваги (self-attention) здатні ефективно враховувати глобальні взаємозв’язки між пікселями. На відміну від CNN, що працюють локально, трансформери аналізують усе зображення цілісно, що особливо важливо при розпізнаванні схожих симптомів різних хвороб.

Ключові моделі, що набули поширення протягом останніх років:

– ViT (Vision Transformer, Dosovitskiy et al.): базова архітектура, яка показала результативність на рівні ResNet при меншій кількості параметрів;

– DeiT (Data-efficient Image Transformer): оптимізована версія для роботи з малими датасетами, що часто зустрічається в аграрному домені;

– MobileViT та PMVT (Lightweight ViT): спрощені моделі для мобільних пристроїв і дронів;

– Swin Transformer: ієрархічна архітектура, яка комбінує локальні та глобальні ознаки;

– MAE (Masked Autoencoder) та DINO: self-supervised підходи, що дозволяють навчати моделі без великих розмічених наборів.

Дослідження показують, що такі моделі демонструють високу робустність у польових умовах (з шумом, різним освітленням, частковими пошкодженнями листків), а також здатні ефективно працювати при невеликій кількості прикладів хвороб (few-shot learning).

Отже, еволюція від традиційних методів до сучасних інтелектуальних систем діагностики знаменує собою новий етап розвитку агротехнологій, у якому ключову роль відіграють методи комп'ютерного зору, глибокі нейронні мережі та трансформери зору. Використання моделей ViT у сфері агротехнологій відкриває нові перспективи для створення інтелектуальних систем моніторингу стану рослин та своєчасного виявлення патологій. Саме ці підходи ляжуть в основу подальшого дослідження, описаного у наступних підрозділах.

1.3 Аналіз досліджень та публікацій застосування Vision Transformers для розпізнавання хвороб рослин

Останні роки характеризуються активним упровадженням трансформерних архітектур у сферу комп'ютерного зору. Якщо раніше домінували згорткові нейронні мережі, то нині Vision Transformer та їх гібридні варіації стають провідним напрямом досліджень у розпізнаванні візуальних об'єктів, зокрема у завданнях класифікації хвороб сільськогосподарських культур [8]. Цей інтерес пояснюється високою точністю ViT, здатністю ефективно працювати з невеликими вибірками даних і можливістю адаптації під польові умови.

У наукових публікаціях останніх років доведено, що Vision Transformer та його модифікації (MobileViT, MAE-fViT, Swin Transformer) демонструють найкращі результати в умовах реальних знімків, де є шум, різне освітлення, фонові об'єкти. Це робить їх особливо перспективними для впровадження в польових умовах, у мобільних та веб-застосунках для фермерів [2-4].

Серед загальних тенденцій досліджень у роботах останніх років відзначається кілька спільних напрямів розвитку Vision Transformers у сільськогосподарських задачах:

– легкі архітектури (MobileViT, PMVT) для мобільних і польових пристроїв;

– гібридні моделі CNN + ViT: поєднують локальні та глобальні ознаки [9];

– самонавчальні (self-supervised) підходи, зокрема MAE (англ. Masked Autoencoder) і DINO, для роботи з малими обсягами розмічених даних;

– адаптація моделей під польові умови (англ. in-the-wild), де враховуються шум, тіні, варіації освітлення та фонові об’єкти;

– дослідження fine-tuning попередньо навчених моделей на спеціалізованих датасетах рослинних культур (PlantVillage, PlantDoc, Wheat Disease).

Далі розглянемо найбільш впливові публікації, що сформували наукову базу для розробки ViT у аграрному домені.

Li Y., Zhang J., Liu D. nf Wang C. Представили полегшену модель PMVT (Plant Mobile Vision Transformer)**,** розроблену спеціально для мобільних і вбудованих пристроїв [2]. Автори продемонстрували, що PMVT досягає високої точності класифікації хвороб листя при значно меншій кількості параметрів, ніж стандартні ViT або ResNet. Особливістю підходу є поєднання блоків самоуваги з оптимізованими MobileNet-конволюціями, що дозволяє моделі ефективно працювати в реальному часі [10]. Робота є важливою для створення польових систем, які не потребують потужного серверного обладнання.

Thakur A., Singh P., Jain R. у своєму дослідженні запропонували гібридну архітектуру CNN + ViT, у якій згорткові шари виділяють локальні текстурні ознаки, а трансформерна частина аналізує глобальні просторові взаємозв’язки [3]. Модель продемонструвала підвищення точності класифікації на кількох відкритих наборах даних (PlantVillage, PlantDoc). Важливо, що автори виконали аналіз інтерпретованості результатів за допомогою Grad-CAM, що дозволило візуально підтвердити, які області листя використовує модель для прийняття рішення. Це підсилює довіру до використання ViT у практичних аграрних системах.

Ullah A., Ahmad M., Zafar A. та Habib S. присвятили роботу класифікації хвороб листя яблуні за допомогою гібридної моделі AppViT, що поєднує переваги згорткових блоків та Vision Transformer [4]. AppViT показала стабільні результати навіть на даних, зібраних у польових умовах (in-the-wild), з точністю понад 98%. Дослідження підтверджує, що ViT ефективно узагальнює візуальні патерни навіть при наявності шуму, варіацій освітлення та часткових ушкоджень листків. Отримані результати мають безпосередню практичну цінність для мобільних і веб-рішень у сфері агромоніторингу.

Zhao H., Xu R., Tang Y. і Wang X. у своїй публікації представили підхід, заснований на self-supervised навчанні, де модель MAE (Masked Autoencoder) спочатку навчається відновлювати приховані частини зображення, а потім донавчається на невеликому спеціалізованому наборі даних картоплі [5]. Результати довели, що така стратегія попереднього навчання суттєво підвищує робустність і узагальнюючу здатність ViT при обмеженій кількості даних. Це особливо важливо для малих фермерських господарств, які не мають великих баз зображень для навчання моделей.

У дослідженні Zhao Z. запропоновано великий мультимодальний датасет, який включає не лише зображення, а й текстові описи симптомів хвороб [7]. Автори порівняли результати різних архітектур, зокрема CLIP-ViT, CNN і Swin Transformer, у складних реальних умовах. Робота підкреслює важливість мультимодальних і мультиджерельних підходів, де поєднання зображень, тексту й метеоданих забезпечує суттєве підвищення точності. Цей напрям демонструє перспективність використання ViT у системах, здатних інтегрувати різні типи інформації.

Модель CLIP-ViT (Contrastive Language-Image Pretraining)**,** створена компанією OpenAI, також активно застосовується у задачах класифікації хвороб рослин [6]. Її перевага – у мультимодальності**:** модель навчається на парах «зображення та текстовий опис». У дослідженнях доведено, що CLIP-ViT може використовуватись у zero-shot режимі, тобто без додаткового навчання, якщо класи мають текстові описи («іржа

пшениці», «фітофтороз картоплі» тощо). Це відкриває нові можливості для створення систем, які розуміють семантику назв хвороб.

У цілому можна відмітити практичну релевантність – у багатьох роботах ViT адаптують під польові умови (шум, фон, різне освітлення) та під мобільні платформи (PMVT, MobileViT-based) [11]. Характерним є застосування методів для малої кількості міток: self-supervised (MAE, DINO) під час fine-tuning показують кращу робустність при обмежених даних, що часто буває у сільському господарстві. Поєднання CNN (локальні ознаки) та ViT (глобальний контекст) дає стабільний приріст у задачах розрізнення схожих класів хвороб.

Узагальнюючи результати сучасних публікацій, можна виділити такі тенденції:

– ViT-моделі демонструють високу точність (до 98-99%) у завданнях класифікації хвороб листя різних культур;

– гібридні архітектури (CNN + ViT) поєднують локальні та глобальні ознаки, що покращує стабільність моделі;

– Self-supervised методи (MAE, DINO) знижують потребу у великих розмічених наборах;

– MobileViT та інші lightweight варіації роблять можливим використання AI безпосередньо на мобільних пристроях;

– мультимодальні підходи (CLIP-ViT) дозволяють здійснювати zero-shot класифікацію на основі текстових описів.

Методи комп’ютерного зору та глибинного навчання сьогодні становлять основу автоматизованих систем моніторингу рослин. Від класичних CNN до сучасних Vision Transformer спостерігається постійне підвищення точності, гнучкості та здатності працювати в реальних умовах. Ці підходи формують наукове підґрунтя для подальших досліджень у сфері створення інтелектуальних систем діагностики хвороб сільськогосподарських культур, що є предметом даної роботи.

Таким чином, Vision Transformers стають фундаментальною технологією для побудови сучасних систем автоматизованої діагностики хвороб рослин. Їх

поєднання з методами донавчання (fine-tuning), мультимодальним аналізом і легкими мобільними реалізаціями відкриває шлях до створення адаптивних, швидких і точних інформаційних систем у сфері розумного землеробства (англ. Smart Farming) [12].

1.4 Огляд програмних рішень для діагностики хвороб у аграрній галузі

Окрім академічних досліджень, у світі активно розвиваються прикладні рішення, які вже реалізують подібні технології. Серед доступних на ринку готових рішень, які сьогодні використовують для діагностики і розпізнавання хвороб сільськогосподарських культур, можна виділити наступні види програмного забезпечення:

- консультаційні / торгові платформи, поєднані з діагностикою;
- мобільні діагностичні додатки;
- платформи precision-ag / enterprice;
- дрон-супутникові аналітичні сервіси.

Однак в цілому класифікація готових програмних рішень чіткого розмежування не має, оскільки консультаційні платформи, поєднані з діагностикою можуть бути мобільними застосунками. А корпоративні платформи можуть передбачати інтеграцію даних з дронів, супутників та мобільні фото. Сучасні аграрні IT-рішення інтегрують комп’ютерний зір із вебзастосунками і мобільними платформами, створюючи комплексні інформаційні системи для підтримки прийняття рішень.

Консультаційні / торгові платформи, поєднані з діагностикою є платформами, що дають діагноз та продають рішення або підключають постачальників послуг. До них можна віднести мобільний застосунок для миттєвої класифікації захворювань за фото Plantix: сервіс діагностики та рекомендацій захисту (https://plantix.net/en/?utm_source=chatgpt.com). При фотографуванні користувачем ураженої рослини застосунок швидко класифікує проблему

(хвороба, шкідник, дефіцит) і повертає діагноз та рекомендації лікування. Система навчена на великій базі користувацьких фото, що підвищує її адаптивність. Розроблена з використанням мобільних CNN-архітектур, бекенд збирає метадані (геолокацію), підсилюючи рекомендації [13]. Перевагами є швидкий зворотний зв'язок і простий UX для фермерів.

Мобільні діагностичні застосунки дозволяють проводити швидку діагностику з телефону: фото → відповідь. Часто мають офлайн-режим або серверну перевірку, фокусуючи увагу на польових умовах (розмиті, частково закриті листки). Розроблені з використанням технологій TensorFlow Lite або мобільні PyTorch-моделі (MobileNet, EfficientNet-lite, легкі ViT/DeiT-версії), а також object detection (TensorFlow Object Detection API) для виділення листа на фото. Перевагами є орієнтація на фермерів, відкриті/академічні рекомендації, простота використання. Прикладом є застосунок PlantVillage Nuru: освітня платформа FAO, що використовує моделі TensorFlow Lite.

Платформи precision-ag / enterprise є корпоративними рішеннями, які орієнтовані на агрохолдинги. Напрями їх застосування – аналітика полів, планування, рекомендації. Реалізовані з використанням технологій: високопродуктивні CV-пайплайни (CNN, detection models – YOLO/Faster-RCNN; іноді власні attention-архітектури), аналітика time-series для раннього виявлення. Використовують spectral indices (NDVI) та машинного навчання ML. Перевагами є масштабність, висока точність для великих площ, інтеграція з агрономічними процесами.

До таких платформ відносять Taranis та xarvio FIELD MANAGER: enterprise-платформи для агрохолдингів, що поєднують дані дронів, супутників і AI-аналіз. Taranis (crop intelligence): аналітична платформа для моніторингу посівів із дронів, яка здійснює аналіз на наявність хвороб, шкідників, бур'янів (https://www.taranis.com/blogs/ai-powered-crop-intelligence/?utm_source=chatgpt.com), дає field-level карти ризиків і рекомендації для застосування інтервенцій. Xarvio FIELD MANAGER: система управління

полями з прогнозуванням ризику захворювань, надає поле-орієнтовані рекомендації (сіяння, захист, добрива) з елементами прогнозування ризику хвороб, spray-timers для оптимізації внесення фунгіцидів. Поєднує rule-based агрономічних моделей та машинне навчання (ML) для прогнозу захворювань на основі метео і карт рослинності. Це практичний інструмент для планування захисту, має інтеграцію з агрономічними процесами (https://ag.xarvio.com/global/field-manager.html?utm_source=chatgpt.com).

Дрон-/супутникові аналітичні сервіси працюють від карти поля до локалізованих інтервенцій. Прикладами сервісів для формування NDVI-карт та автоматичного виявлення зон стресу є DroneDeploy та AgroScout: аналітичні сервіси для моніторингу полів і виявлення уражень з повітря. Як працює DroneDeploy: оператор планує політ дрона → DroneDeploy збирає 2D/3D orthomosaics → plant-health інструменти (NDVI, custom indices) та ML-аналіз для виявлення зон стресу, генеруючи генеруються prescription maps (VRA) (https://www.dronedeploy.com/solutions/agriculture?utm_source=chatgpt.com).
Перевагами є охоплення великих площ; раннє виявлення; інтеграція з VRA/агрегацією. AgroScout (https://agro-scout.com/?utm_source=chatgpt.com) комбінує дрон/супутникові дані та ground photos, здійснює аналіз з використанням AI для раннього попередження про шкідників/хвороби, формує агрономічні звіти. Команда також надає агрономічну верифікацію результатів. Переваги: висока точність завдяки агрономічній валідації, підходить для контрактних сервісів.

До переваг наявних готових програмних рішень можна віднести швидке виявлення та масштабування моніторингу (особливо дрони/супутники), можливість раннього втручання та зниження втрат, отримання кращого результату за рахунок комбінування джерел даних [14]. Загалом сучасні підходи дозволяють перейти від реактивної до превентивної моделі управління посівами, коли система не лише розпізнає симптоми хвороб, а й прогнозує ризики їх виникнення. Впровадження таких технологій є складовою стратегії «розумного землеробства» (англ. smart farming) та «цифрового сільського господарства» (англ. digital

agriculture). Такі системи поєднують алгоритми розпізнавання з геоінформаційними технологіями (GIS), хмарними обчисленнями та мобільними інтерфейсами, що дозволяє виконувати аналіз у реальному часі та оперативно реагувати на зміни стану культур.

1.5 Проблеми та обмеження існуючих підходів

Попри значні досягнення у сфері автоматизованої діагностики хвороб сільськогосподарських культур, існуючі рішення мають низку наукових, технічних і практичних обмежень, які ускладнюють їх широке впровадження в аграрну практику. Проблеми стосуються як якості даних, так і архітектурних властивостей моделей, а також економічних і організаційних чинників використання інтелектуальних систем у реальних умовах [15]. Розглянемо їх детальніше.

1.*Нестача якісних та репрезентативних даних*. Більшість публічно доступних наборів даних (PlantVillage, PlantDoc, Mendeley Wheat Disease тощо) містять зображення, зроблені в контрольованих лабораторних умовах із чистим фоном, рівним освітленням і чіткими ознаками хвороб. Такі зображення істотно відрізняються від польових фото, отриманих у реальних умовах, де присутні:

- шум, нерівномірне освітлення, тіні, фонова рослинність;
- варіації сортів, стадій росту, географічних умов;
- неоднорідні прояви однієї і тієї ж хвороби.

Це явище відоме як domain gap – розрив між даними, на яких навчалась модель, і тими, що зустрічаються під час практичного застосування. У результаті модель, яка демонструє точність 98-99% у лабораторії, може знижувати показники до 70-80% у польових умовах.

2. *Обмеженість анотацій і труднощі розмітки*. Розмітка зображень потребує участі фахівців-агрономів або фітопатологів, оскільки неправильне маркування може призвести до спотворення навчання. Для багатьох культур існує брак чітких візуальних описів хвороб, а окремі збудники мають подібні симптоми.

Це ускладнює створення якісних датасетів із прецизійними мітками. Дослідження 2024-2025 років вказують, що для рідкісних або регіонально специфічних хвороб часто є лише кілька десятків зображень, чого недостатньо для класичного supervised-навчання. Цю проблему частково вирішують методи self-supervised або few-shot learning (MAE, DINO), однак вони потребують додаткових обчислювальних ресурсів і налаштування гіперпараметрів [16].

3. *Висока вимогливість моделей до апаратних ресурсів*. Більшість сучасних архітектур, зокрема ViT-Large або Swin-Transformer, мають велику кількість параметрів (до сотень мільйонів) і потребують потужних графічних процесорів (GPU) для навчання та інференсу. Це ускладнює їх використання на мобільних пристроях або в умовах обмежених обчислювальних ресурсів (малі фермерські господарства, польові станції). Хоча розроблені полегшені версії (MobileViT, PMVT, DeiT-Tiny) частково зменшують цю проблему, компроміс між швидкодією та точністю залишається суттєвим викликом. Крім того, складність моделей ускладнює їх пояснюваність та аналіз результатів для користувача-агронома.

4. *Недостатня інтерпретованість результатів*. Попри високі метрики точності, результати глибинних моделей залишаються для користувача “чорним ящиком”. У реальних аграрних умовах фермер або агроном потребує не лише класифікації хвороби, але й пояснення, на основі яких ознак зроблено висновок (наприклад, які ділянки листка є критичними). Багато дослідників (Thakur et al., 2023; Ullah et al., 2024) застосовують візуалізаційні методи Grad-CAM або Attention Map для покращення прозорості моделей, проте такі рішення поки що не є стандартом у готових програмних продуктах. Відсутність інтерпретованості знижує рівень довіри користувачів до автоматизованої системи [17].

5. *Проблеми генералізації та перенавчання*. Моделі, навчені на вузькому наборі даних (наприклад, лише яблуневі або томатні листки), часто демонструють низьку узагальнювальну здатність при застосуванні до інших культур або регіонів. Це пов’язано з перенавчанням (overfitting), коли модель запам’ятовує специфічні патерни навчального набору, замість формування універсальних ознак. У зв’язку з

цим виникає потреба у fine-tuning моделей ViT на локальних даних, що дозволяє підлаштовувати систему під конкретні сорти культур, регіональні патогени та кліматичні умови [18].

6. *Висока вартість та закритість комерційних рішень*. Багато сучасних систем (Plantix, xarvio FIELD MANAGER, Taranis, AgroScout) є комерційними платформами з обмеженим доступом до внутрішніх моделей і даних. Їх використання передбачає оплату підписки, зберігання даних у хмарі та відсутність контролю над моделлю або результатами. Для освітніх, дослідницьких або регіональних задач це створює бар'єр, оскільки відсутня можливість гнучкої адаптації під локальні потреби. Саме тому актуальним напрямом є створення власних відкритих систем, що забезпечують контроль над даними, моделями та процесом навчання [19].

7. *Неповна інтеграція з агрономічними процесами*. Більшість існуючих рішень обмежуються функцією класифікації зображень і не забезпечують подальшу підтримку прийняття рішень – рекомендації щодо лікування, прогноз поширення або економічну оцінку ризиків. Це знижує практичну цінність автоматизованої діагностики. Для підвищення ефективності системи мають інтегрувати агрономічні бази знань, геоінформаційні карти та аналітичні модулі, що дозволять фермеру не лише ідентифікувати проблему, а й отримати конкретні дії для її вирішення [20].

8. *Етичні та безпекові аспекти*. Збирання й передавання зображень із геотегами через хмарні сервіси створює ризики для конфіденційності користувачів і потенційного витоку аграрних даних. Також виникає питання відповідальності за помилки діагностики, якщо рекомендації моделі призводять до матеріальних збитків [21]. Для подолання цих обмежень необхідно впроваджувати прозорі алгоритми, локальне зберігання даних і аудит моделей, що є важливим напрямом подальших досліджень.

Здійснений аналіз дозволяє зробити насутпні висновки. Сучасні підходи до автоматизованої діагностики хвороб рослин демонструють високий потенціал, але

стикаються з низкою проблем: нестача репрезентативних даних і якісних анотацій; складність і ресурсомісткість моделей; відсутність інтерпретованості результатів; потреба в локальній адаптації та відкритості систем. Моделі, навчені на “чистих” датасетах (PlantVillage), погано працюють у “польових” умовах без донавчання. Вартість enterprise-рішень є високою, в наявності присутні етичні та комерційні ризики при використанні консультаційних платформ. Для надійності рекомендації потребують агрономічної верифікації.

Подолання цих обмежень вимагає створення гнучких, відкритих інформаційних систем, які поєднують потужність трансформерних моделей з практичною орієнтованістю на користувача. Існує потреба у розробці системи, орінтованої на невеликі та середні фермерські господарства, для вирішення своєчасної діагностики хвороб без залучення висококваліфікованих експертів, яка при обмеженій складності та ресурсомісткості підвищує інтерпретованість результатів.

1.6 Постановка задачі

Проведений аналіз підходів до діагностики хвороб сільськогосподарських культур із використанням сучасних технологій штучного інтелекту показав, що існуючі рішення, попри високі результати на експериментальних наборах даних, мають низку обмежень – насамперед відсутність адаптації до локальних умов, нестачу репрезентативних даних і складність інтеграції в зручні інформаційні системи для практичного використання. Виходячи з цього було зроблено висновок про необхідність розробки інформаційної системи, яка забезпечує автоматизовану діагностику хвороб рослин із використанням трансформерів зору, адаптованої під умови реальних аграрних господарств.

Розробка інформаційної системи передбачає створення та донавчання на спеціалізованому Data Set моделі ViT для діагностики хвороб сільськогосподарських культур та її інтеграції до вебзастосунку, який надає

користувачеві зручний інтерфейс для виявлення патологій та візуальної інтерпретації рішень у режимі реального часу. Для цього необхідно необхідно реалізувати наступне:

– проаналізувати предметну сферу діагностики хвороб сільськогосподарських культур з використанням сучасних технологій ШІ, зробити огляд сучасних досліджень у цій сфері та програмних рішень, що реалізують діагностику хвороб рослин із використанням комп'ютерного зору;

– дослідити переваги та недоліки існуючих підходів, обґрунтувати вибір моделі Vision Transformer для вирішення поставленої задачі;

– провести формування, очищення та підготовку навчаючого data set із зображеннями листя сільськогосподарських культур;

– реалізувати процес донавчання (fine-tuning) базової навченої моделі ViT на підготовленому наборі даних, провести дослдіження з метою підбору оптимальних гіперпараметрів, які забезпечують високу точність класифікації та прийнятну швидкість навчання;

– оцінити якість донавченої моделі ViT за допомогою метрик Accuracy, Precision, Recall та F1-score;

– розробити вебзастосунок, що містить інтегровану навчену для розпізнавання хвороб модель ViT та забезпечує інтерфейс для завантаження зображень листя сільськогосподарських культур і отримання результатів діагностики та їх інтерпретації;

– провести тестування системи на реальних зображеннях і проаналізувати результати роботи у практичних сценаріях.

Застосування архітектури Vision Transformer для автоматизованої діагностики хвороб сільськогосподарських культур із власним механізмом fine-tuning на спеціалізованому датасеті вирішено провести, оцінюючи вплив наступних гіперпарамерів параметрів: learning rate, batch size, patch size, кількість attention heads з метою забезпечення високої точності класифікації хвороб у реальних умовах.

Для створення застосунку фермера, який дозволяє у автономному режимі діагностувати хвороби сільськогосподарських культур, було обрано модель ViT-Tiny-Patch16-224, натреновану для розпізнавання хвороб листя чотирьох культур (кукурудза, картопля, рис, пшениця), класифікованих за 14 класами, що представляють різні види рослин та їхній стан здоров'я. Для її донавчання було використано Data Set, у якому представлено 14 сільськогосподарських культур, класифікованих за 55 класами, та досліджено вплив гіперпараметрів fine-tuning ViT із метою визначенні оптимальних параметрів донавчання моделі на доменних даних для підвищення точності розпізнавання хвороб та інтерпретованості результатів за допомогою карт уваги.

Об'єктом дослідження є процес діагностики хвороб сільськогосподарських культур.

Предметом дослідження є моделі трансформерів зору, методи їх навчання для класифікації захворювань рослин та програмні засоби їх реалізації у вигляді інформаційних систем.

Мета дослідження – підвищення точності розпізнавання хвороб сільськогосподарських рослин та інтерпретованості результатів за допомогою карт уваги шляхом створення інформаційної системи із використанням механізму самоуваги трансформерів зору.

Досягнення поставленої мети обумовлює необхідність вирішення наступних **завдань**:

– дослідити теоретичні засади діагностики хвороб сільськогосподарських культур, проаналізувати існуючих досліджень та програмних рішень для автоматизованого виявлення хвороб рослин в аграрній галузі;

– обґрунтувати вибір моделі Vision Transformer, набору даних і методів для її донавчання та інструментальних засобів розробки інформаційної системи;

– провести донавчання моделі ViT на підготовленому наборі даних та дослідити налаштування її гіперпараметрів для отримання високої точності розпізнавання хвороб;

– здійснити моделювання, проєктування, програмну реалізацію системи діагностики хвороб сільськогосподарських культур та оцінити її якість.

Висновки до розділу 1

У першому розділі було проведено теоретичний аналіз предметної області автоматизованої діагностики хвороб сільськогосподарських культур, розглянуто традиційні методи визначення патологій рослин, а також сучасні підходи, засновані на технологіях комп’ютерного зору та глибинного навчання. Встановлено, що традиційні методи діагностики – зокрема візуальні спостереження, біохімічні тести та лабораторні аналізи - є трудомісткими, потребують значних ресурсів і залежать від кваліфікації фахівців. Це обмежує їх ефективність у великих господарствах і не дозволяє здійснювати моніторинг стану рослин у реальному часі.

Аналіз сучасних підходів показав, що впровадження методів комп’ютерного зору, машинного та глибинного навчання дозволяє суттєво підвищити точність і швидкість ідентифікації хвороб рослин. Особливо перспективним напрямом є використання трансформерних архітектур зору (Vision Transformers, ViT), які завдяки механізму самоуваги (self-attention) здатні враховувати глобальні просторові взаємозв’язки на зображеннях.

Огляд наукових публікацій останніх років підтвердив ефективність ViT і його модифікацій (DeiT, Swin, MobileViT, MAE-fViT) у завданнях класифікації хвороб рослин. Водночас визначено низку проблем і обмежень існуючих рішень – насамперед нестачу репрезентативних польових даних, складність адаптації моделей до локальних умов, високі обчислювальні вимоги та недостатню інтерпретованість результатів.

На основі проведеного аналізу сформульовано мету, об’єкт, предмет і завдання дослідження, спрямовані на створення інформаційної системи діагностики хвороб сільськогосподарських культур із використанням моделі Vision Transformer. Запропоновано підхід, що передбачає fine-tuning попередньо навченої

ViT-моделі на спеціалізованому наборі зображень і подальшу інтеграцію результатів у вебзастосунок для користувачів-агрономів і фермерів.

Теоретичні положення, викладені у розділі, становлять наукове підґрунтя для подальшої розробки моделей, алгоритмів і програмної реалізації системи автоматизованої діагностики хвороб рослин, що розглядається в наступному розділі.

2 ДІАГНОСТИКА ХВОРОБ СІЛЬСЬКОГОСПОДАРСЬКИХ КУЛЬТУР ІЗ ВИКОРИСТАННЯМ МОДЕЛЕЙ VISION TRANSFORMERS

2.1 Архітектура моделей Vision Transformer

Поява архітектури Vision Transformer у 2020 році стала одним із ключових проривів у сфері комп'ютерного зору. На відміну від традиційних згорткових нейронних мереж, які обробляють зображення через локальні фільтри, ViT використовує механізм самоуваги, що дозволяє моделі враховувати глобальні взаємозв'язки між усіма частинами зображення. Завдяки цьому трансформери зору демонструють високу точність класифікації, особливо у випадках, коли симптоми хвороб розташовані нерівномірно або проявляються у вигляді складних візуальних патернів.

Архітектура ViT була адаптована з оригінальної моделі Transformer (Vaswani et al., 2017), спочатку створеної для обробки тексту [22]. Основна ідея полягає у представленні зображення не як двовимірної сітки пікселів, а як послідовності векторів – патчів (англ. patches), що дозволяє застосовувати до них ті ж механізми, що й у задачах природної мови. На вхід модель отримує зображення розміром, наприклад, 224×224 пікселі, яке розбивається на фрагменти (патчі) розміром 16×16 пікселів. Кожен патч перетворюється у вектор ознак (англ. embedding), після чого всі вектори подаються в трансформер як послідовність токенів [23].

Архітектура Vision Transformer складається з кількох основних блоків: Patch Embedding Layer, Positional Encoding, Class Token (CLS), Encoder Layers, MLP Head (Classification Layer) (рис. 2.1). Опишемо їх більш детально.

Patch Embedding Layer – шар розбиття зображення на патчі. Кожен патч лінійно перетворюється у вектор фіксованої довжини (наприклад, 768 елементів). Результатом є послідовність векторів, аналогічна токенам у мовних моделях.

Positional Encoding – додавання позиційної інформації. Оскільки трансформер не має вбудованого розуміння просторового розташування елементів,

до кожного векторного представлення додається позиційне кодування, яке містить інформацію про координати патча в межах зображення.

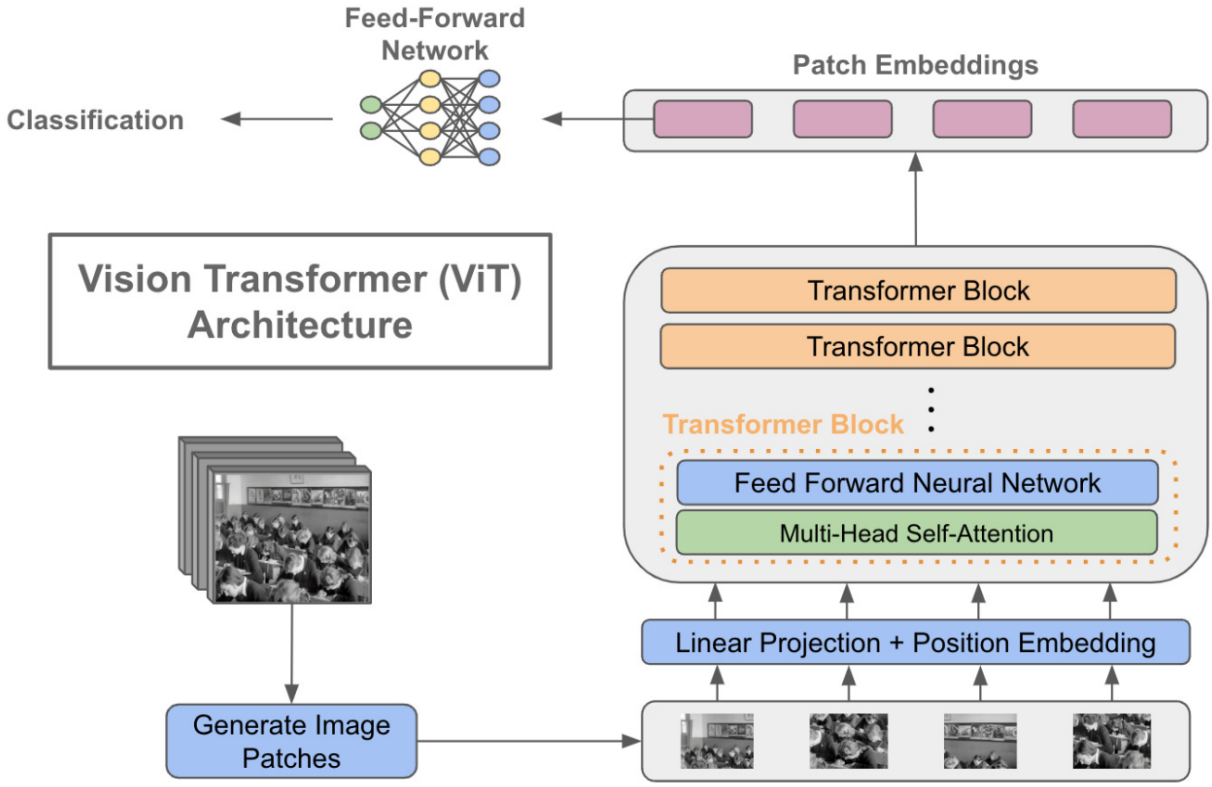

Рисунок 2.1 – Архітектура Vision Transformer

Class Token (CLS) – спеціальний вектор, який додається на початок послідовності та використовується для зчитування узагальненої інформації про все зображення. Після проходження через трансформер саме цей токен подається на фінальний класифікатор, який визначає клас (наприклад, «здоровий листок», «іржа пшениці», «фітофтороз картоплі» тощо) [24].

Encoder Layers (блоки трансформера) – основна частина ViT, що складається з багатьох однакових шарів, кожен з яких містить:

– Multi-Head Self-Attention (MHSA) – механізм, який дозволяє кожному патчу приділяти увагу до всіх інших і обчислювати залежності між ними, це дозволяє моделі одночасно аналізувати глобальні контексти (наприклад, розподіл плям по всьому листку);

– Feed-Forward Network (FFN) – двошарова нейронна мережа, яка перетворює результати уваги в нові представлення ознак;

– Normalization (LayerNorm) та Residual Connections – стабілізують навчання і забезпечують ефективне поширення градієнтів.

MLP Head (англ. Classification Head) – це структура з одного або кількох шарів, яка перетворює вихід моделі (наприклад, CLS-токен у ViT) у прогноз класв – ймовірності належності зображення до певного класу хвороби.

Класифікаційна голова – це модуль, який включає нормалізацію, один або кілька повнозв'язних шарів (Linear), нелінійність (GELU/ReLU), можливо dropout. Класифікаційна голова може містити один лінійний шар (Linear/FC) або кілька шарів (Linear → Dropout → Activation → Linear). Загалом класифікаційний шар (англ. classification layer) – це конкретний шар, зазвичай один лінійний (Fully Connected), є частиною голови, але інколи може бути уся голова у простих моделях.

Використання ViT у сфері агроінформатики має низку суттєвих переваг у задачі діагностики хвороб рослин порівняно з традиційними CNN-моделями:

– глобальне бачення контексту: механізм самоуваги дозволяє одночасно враховувати залежності між усіма частинами листка, що важливо при розподілених або дифузних симптомах хвороб;

– висока узагальнювальна здатність: ViT легше адаптується до нових культур або типів захворювань при донавчанні (fine-tuning);

– менша залежність від кількості параметрів у шарі фільтрації, модель не потребує фіксованих фільтрів, що дозволяє ефективно навчатися на різнорідних даних;

– сумісність із self-supervised навчанням: використання стратегій MAE або DINO дозволяє моделі навчатися навіть на неанотованих наборах польових зображень;

– гнучкість у масштабуванні: існують різні варіанти ViT – Tiny, Small, Base, Large, що дозволяє обирати баланс між швидкодією та точністю під конкретне застосування (сервер, дрон, мобільний пристрій).

Серед наявних моделей ViT можна виділити такі основні різновиди:

– ViT-Base / ViT-Large – базові моделі, навчені на великих наборах даних (ImageNet-21k, JFT-300M); використовуються для високоточної класифікації;

– DeiT (Data-efficient Image Transformer) – адаптований під невеликі вибірки, застосовується при обмежених польових даних;

– Swin Transformer – має ієрархічну структуру з ковзними вікнами (Shifted Windows), що підвищує ефективність для зображень високої роздільної здатності;

– MobileViT / PMVT – полегшені архітектури для мобільних і вбудованих пристроїв; застосовуються в польових умовах і в мобільних додатках;

– MAE (Masked Autoencoder ViT) - self-supervised модель, що відновлює приховані частини зображення; особливо ефективна при нестачі анотованих даних.

Таким чином, архітектура Vision Transformer (ViT) є перспективною основою для створення інтелектуальних систем діагностики хвороб рослин. Її ключова перевага – здатність аналізувати зображення не локально, а в глобальному контексті, що підвищує точність класифікації навіть при складних умовах зйомки.

2.2 Механізм багатоголової уваги моделей ViT

ViT розглядає зображення як сукупність патчів (англ. patches) – ділянок фіксованого розміру, що не перекриваються, на які розбивається вхідне зображення. Кожен патч перетворюється у вектор ознак (англ. embedding), до якого додається позиційне кодування для збереження просторової інформації. Далі ці вектори проходять через трансформенні енкодери, де працює механізм самоуваги (рис. 2.2) [25]. Self-Attention Vision transformer повністю узгоджується з загальною архітектурою трансформера, запозиченою з NLP-моделей обробки природної мови, працюючи замість слів з патчами. Завдяки цьому реалізовано перенесення архітектури трансформера в область комп’ютерного зору, де увага моделює

просторові відносини між частинами зображення, а не лінгвістичні зв’язки між словами, визначаючи просторовий контекст кожного патча серед інших.

У моделях ViT реалізовано механізм багатоголової уваги (англ. Multi-Head Self-Attention, MHSA), де кожна голова навчається фокусуватися на різних ознаках просторових чи семантичних зв’язків між патчами (колір, форма, текстура, контур, плями, краї уражень тощо).

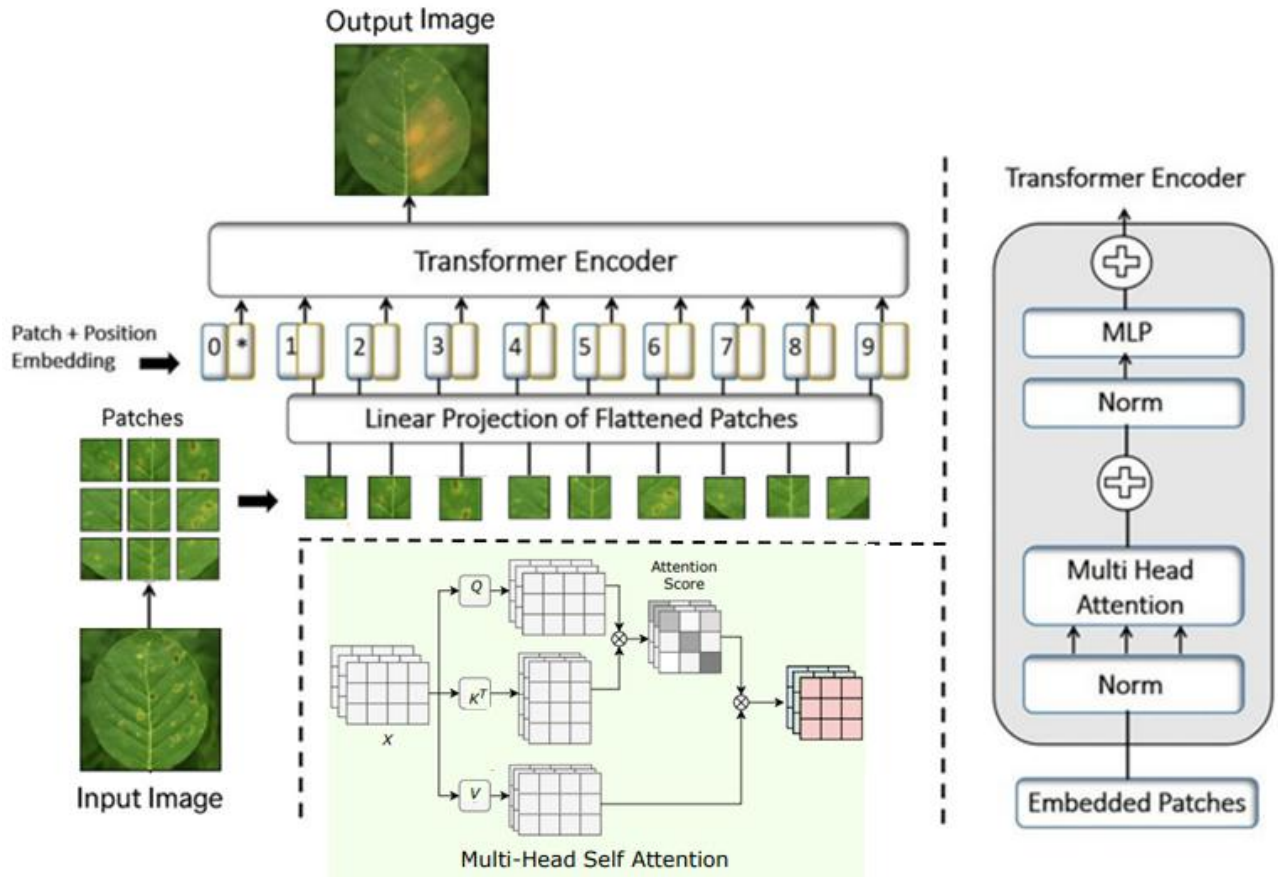

Рисунок 2.2 – Схема роботи self-attention у Vision transformer

Фундаментальним для паралельного вивчення різних аспектів зв’язків між патчами зображення є зв’язок між головами уваги та матрицями Q, K, V, які розраховують для кожної голови:

$$Q = XW_Q\ ,\ K = XW_k\ ,\ V = XW_V\ , \tag{2.1}$$

де X – вхідний вектор патча,

Q (Query) – матриця запитів: векторне представлення поточного патча, який запитує інформацію у решти, використовується для порівняння з іншими Key–векторами (наприклад: ознаки ураження плямою),

K (Key) – матриця ключів: векторні представлення всіх патчів, з якими порівнюється запит, представляє інформацію, яку несе патч (наприклад: колір, текстура, форма, тощо),

V (Value) – матриця ознак: векторні представлення ознак, що передаються, підсумовані з урахуванням ваг уваги (містять корисну інформацію, яку несе патч),

W_Q, W_K, W_V – матриці налаштовуваних під час навчання ваг.

Матриця сумісності містить міри подібності, які визначають, наскільки патч *i* вважає важливим патч *j*, що визначається як скалярний добуток:

$$Score_{ij} = Q_i \cdot K_j^T . \tag{2.2}$$

Ці міри нормалізуються, перетворюючи значення сумісності у ймовірнісні ваги уваги:

$$\alpha_{ij} = soft\max\left(\frac{Q_i \cdot K_j^T}{\sqrt{d_k}}\right), \tag{2.3}$$

де α_{ij} – коефіцієнт уваги (англ. attention weight),

d_k – розмірність матриць ключів і запитів.

Модель комбінує усі значення *V*, зважені на коефіцієнти уваги, представлені для кожної голови у вигляді матриці самоуваги:

$$Attention(Q, K, V) = \sum_j \alpha_{ij} V_j . \tag{2.4}$$

Кожен патч через Q запитує інформацію у інших патчів через K, визначаючи, на які частини звернути увагу [26]. Потім комбінуються значення V, зважені за важливістю, формуючи нове контекстно збагачене представлення ознак. Таким чином на виході кожної голови *h* формується власна карта уваги (англ. Attention Maps) – матриця розмірністю n x n, де n – кількість патчів:

$$Attention\,(Q_h, K_h, V_h) = soft\max\left(\frac{Q_h {K_h}^T}{\sqrt{d_k}}\right) V_h . \qquad (2.5)$$

Конкатенація виходів усіх голів генерує загальну карту уваги, яка є матрицею злиття результатів голів і дає комплексне уявлення про зображення. Це дозволяє фокусувати увагу на уражених ділянках зображення рослин, підсилює розпізнавання патернів (візерунки, форми уражень), ігноруючи фон та враховуючи віддалені зв’язки (наприклад, плями з різних частин листа).

У контексті діагностики хвороб рослин на відміну від CNN моделей, які розпізнають локальні ознаки (плями, краї, текстури), модель ViT бачить картину цілком, розпізнаючи глобальні ознаки (структуру листа, симетрію, взаємозв’язок уражень). У CNN врахування контексту обмежене сусідніми пікселями, а у ViT кожен патч взаємодіє з усіма [27]. Це покращує розуміння хвороби, яка проявляється на різних ділянках. Для адаптації моделі до нових хвороб та культур можна донавчати тільки останні шари, потребуючи менших обчислювальних ресурсів для навчання, що прискорює fine-tuning.

2.3 Методи попередньої обробки та аугментації зображень

У процесі створення інформаційної системи діагностики хвороб сільськогосподарських культур важливе місце займає етап підготовки даних, які буде використано для донавчання моделі ViT. Якість навчального набору

зображень безпосередньо впливає на точність, стабільність і швидкість збіжності моделі при навчанні. Оскільки зображення листя рослин можуть мати різну роздільну здатність, освітлення, орієнтацію та фон, необхідно провести попередню обробку (англ. preprocessing) і аугментацію (англ. augmentation) для підвищення якості даних та збільшення різноманітності прикладів [28].

Попередня обробка – це базовий етап, спрямований на приведення всіх зображень до єдиного формату, видалення шумів і забезпечення сумісності з архітектурою Vision Transformer. Розглянемо основні кроки preprocessing більш детально.

1. Зміна розміру (англ. Resizing). Для навчання моделей ViT усі зображення приводяться до стандартного розміру 224×224 пікселів, який відповідає вхідному шару більшості базових трансформерів (ViT-Base, DeiT). Це гарантує узгодженість патчів під час токенізації зображення.

2. Центрування та обрізання (англ. Cropping). Виконується автоматичне виділення центральної області зображення або обрізання за контуром листка, щоб зменшити вплив фону. Ця операція дозволяє зосередити увагу моделі на об’єкті дослідження - поверхні листя, де проявляються симптоми хвороби.

3. Нормалізація (англ. Normalization). Значення пікселів масштабуються у діапазон [0,1] або нормуються відповідно до середнього mean і стандартного відхилення std, як у датасеті ImageNet:

$$x' = \frac{x - mean}{std} \tag{2.6}$$

Це забезпечує стабільність градієнтів під час навчання та прискорює збіжність моделі.

4. Фільтрація шуму (англ. Denoising). У польових зображеннях часто присутні артефакти - пил, водяні краплі, тіні, розмиття. Для їх усунення

застосовуються медіанні та гаусові фільтри, адаптивне згладжування або двостороння фільтрація (bilateral filtering).

5. Балансування яскравості та контрасту. Різні умови освітлення можуть змінювати колірні характеристики листя. Для компенсації таких ефектів використовуються методи Color Jitter (випадкове коливання яскравості, контрасту, насиченості) або Histogram Equalization, які дозволяють зробити набір даних більш однорідним.

6. Усунення фону (англ. Background Removal). Для підвищення точності класифікації можливе використання алгоритмів сегментації об’єкта (Thresholding, GrabCut, U-Net), що видаляють непотрібний фон і залишають лише листок. Це зменшує вплив зовнішніх факторів, наприклад тіні або ґрунту.

7. Балансування класів. У більшості датасетів спостерігається дисбаланс – деякі класи хвороб представлені значно менше. Для уникнення переваги моделі певного класу застосовують оверсемплінг (дублювання малих класів) або підбір вибірок із ваговими коефіцієнтами (class weights).

Аугментація зображень (англ. augmentation) – це штучне збільшення обсягу навчального набору за рахунок генерування нових зображень на основі існуючих шляхом внесення невеликих випадкових змін. Вона допомагає покращити узагальнювальну здатність моделі та запобігає перенавчанню (overfitting). Основні методи аугментації, застосовані в роботі:

– геометричні перетворення: обертання (Rotation) на випадкові кути (±15–30°); віддзеркалення (Flip) по горизонталі або вертикалі; зсув (Translation) - зміщення зображення в межах кадру; масштабування (Zoom / Resize) - випадкове збільшення або зменшення масштабу; афінні перетворення (Affine Transform) - невелика зміна форми чи нахилу об’єкта;

– колірні та фотометричні перетворення: Random Brightness / Contrast / Saturation - зміна яскравості, контрасту й насиченості кольорів; Hue Shift - варіація відтінків зеленого, щоб модель не прив’язувалася до конкретного кольору листя; Gaussian Blur або Sharpening - легке розмиття чи підсилення

контурів для моделювання різних умов фокусування;

– просторові маскування (Masking / Cutout): частина зображення випадково затемнюється або замінюється шумом, це навчає модель ігнорувати відсутні або пошкоджені ділянки листя;

– Random Erasing: схоже до попереднього методу, але використовується прямокутна ділянка, що замінюється випадковим шумом, підвищує робастність при частковому перекритті листка іншими об’єктами;

– міксування (Mixup, CutMix): методи, коли два зображення накладаються одне на одне, а мітки класів комбінуються пропорційно, це допомагає зменшити надмірну впевненість моделі в передбаченнях і підвищити узагальнення.

Для моделей типу ViT аугментація має особливе значення, оскільки механізм самоуваги (англ. self-attention) аналізує глобальні зв’язки між усіма патчами. Різноманітність вхідних варіацій допомагає моделі краще навчитися залежностям між частинами листка та коректно реагувати на зміну кута, освітлення або фону. Зазвичай застосовують таку комбінацію методів:

– RandAugment – випадкове вибирання набору аугментацій із фіксованої множини;

– Color Jitter + Random Crop + Horizontal Flip – стандартний набір для навчання ViT на зображеннях рослин;

– Normalization + Random Erasing – для зменшення чутливості до шуму та артефактів.

Таким чином, попередня обробка та аугментація є невід’ємною частиною побудови системи автоматизованої діагностики хвороб рослин. Їх застосування формує якісний вхідний набір для навчання трансформерної моделі, забезпечуючи підвищення її точності, стабільності та узагальнювальної здатності. Застосування описаних методів попередньої обробки та аугментації дозволяє: підвищити точність класифікації хвороб рослин на 5–15%; зробити модель стійкішою до

варіацій умов зйомки; зменшити перенавчання на малих датасетах; забезпечити кращу генералізацію при тестуванні на нових польових знімках.

2.3 Fine-tuning моделей ViT для класифікації хвороб

Ефективність розпізнавання хвороб рослин за допомогою моделей ViT значною мірою залежить від правильно побудованого алгоритму навчання та адаптації моделі до специфіки даних [29]. У даній роботі для реалізації класифікації зображень листя використано архітектуру Vision Transformer (ViT), яка попередньо навчена на великому наборі зображень ImageNet-21k і потім донавчається (англ. fine-tuning) на спеціалізованому датасеті хвороб рослин.

Fine-tuning у контексті моделей Vision Transformer – це процес, коли попередньо навчена на великому датасеті модель адаптується до нової, специфічної задачі, такої як класифікація хвороб рослин. Це цілеспрямоване донавчання частини або всієї моделі Vision Transformer на новому доменному датасеті, з використанням попередньо натренованих ваг, щоб швидко адаптувати її до нової задачі, зберігаючи вже набуті представлення ознак [30].

Основні особливості fine-tuning ViT:

– використання попередньо натренованих ваг: ViT спочатку навчена на великій колекції зображень – м одель уже вміє розпізнавати базові форми, текстури та патерни;

– адаптація до нових класів: змінюється лише кінцевий класифікаційний шар (head), щоб відповідати кількості класів нового датасета;

– заморожування або часткове заморожування шарів: можна заморозити ранні шари й тренувати лише верхні або навчати всю модель з меншим learning rate;

– застосування низької швидкості навчання: важливо не «зруйнувати» попередні корисні представлення;

– ефективність при малих датасетах: fine-tuning дозволяє навчити точну модель навіть при обмеженій кількості зображень.

Vision Transformer відрізняється від традиційних згорткових мереж тим, що навчається на послідовностях патчів, обробляючи їх за допомогою механізму багатоголової самоуваги. Навчання моделі полягає в оптимізації вагових коефіцієнтів усіх шарів таким чином, щоб мінімізувати функцію втрат (англ. loss function) між передбаченням моделі та правильними мітками класів [31].

Навчання моделі ViT – це процес, у якому модель адаптує свої параметри, оптимізуючи багатоголову увагу, ембединг патчів та MLP-блоки, щоб мінімізувати функцію втрат і підвищити точність розпізнавання хвороб рослин. Він включає обчислення втрат, зворотнє поширення градієнта та оновлення ваг усіх компонентів: ембедингів патчів, багатогоової уваги, MLP_блоків та класифікаційної голови. Опишемо послідовно основні операції навчання.

1. Вхідні операції навчання:

– розбиття зображення *X* на патчі *pi*:

$$X \to \{p_1, p_2, ..., p_n\}; \tag{2.7}$$

– лінійне перетворення патчів у вектори. Кожен патч перетворюється у вектор ембедингу:

$$z_i \to W_p \cdot p_i + b_p, \tag{2.8}$$

додається positional encoding:

$$z_i' = z_i + \varepsilon_i; \tag{2.9}$$

– формування послідовності для трансформера:

$$Z = [CLS, z_1', z_2', ..., z_n']. \tag{2.10}$$

2. Обчислення самоуваги:

– для кожної голови *h*: $Q = XW_h^Q$, $K = XW_h^K$, $V = XW_h^V$;

– увага: $A_h = soft\max\left(\frac{Q_h \cdot K_h^T}{\sqrt{d_k}}\right)$;

– вихід голови: $A_h = A_h V_h$;

– конкатенація всіх голів: $H = Concat(H_1, H_2, ..., H_h)W^O$.

3. Проходження через MLP-блоки:

$$U = MLP(LayerNorm(H + Z)), \tag{2.11}$$

це повторюється для L шарів трансформера.

4. Класифікаційна голова – береться CLS-токен після останнього шару h_{CLS} для отримання прогнозу класу:

$$\hat{y} = soft\max(W_{cls}h_{CLS} + b_{cls}). \tag{2.12}$$

5. Обчислення функції втрат. Оскільки розпізнавання хвороб рослин – це класифікація, то функцію втрат *L* зазвичай обчислюють за формулою:

$$L = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{i,j}\log(\hat{y}_{i,j}) \tag{2.13}$$

де n – кількість зразків у батчі,

m – кількість класів,

$y_{i,j}$ – істинна мітка (0 або 1),

$\hat{y}_{i,j}$ – передбачена ймовірність належності до класу *j*.

6. Зворотне поширення градієнта: градієнти обчислюються для усіх параметрів – матрриці патч-ембедингу, positional encodimg, усіх матрць Q, K, V, MLP-блоків, класифікаційної голови.

7. Оновлення ваг (оптимізація): зазвичай використовується оптимізатор AdamW:

$$\theta_{t+1} = \theta_{t+1} - \eta \cdot \frac{m_t}{\sqrt{v_t} + \varepsilon} . \quad (2.14)$$

де η – learning rate,

m_t – перший момент,

v_t – другий момент.

Процес навчання моделі для задачі класифікації хвороб сільськогосподарських культур включає кілька послідовних етапів. Зупинимося на їх описі більш детально.

1. *Ініціалізація моделі*. Завантажується попередньо навчена модель ViT-Base-Patch16-224 або ViT-Tiny-Patch16, яка вже має знання про загальні візуальні структури. Вихідний шар класифікації (head) замінюється на новий шар із кількістю нейронів, що відповідає кількості класів хвороб у використовуваному датасеті (наприклад, 38 класів у PlantVillage).

2. *Заморожування шарів*. На початкових етапах частина шарів моделі заморожується, тобто їхні ваги не оновлюються під час навчання. Це дозволяє зберегти попередньо здобуті узагальнені знання моделі і зменшити ризик перенавчання при невеликій кількості нових даних.

3. *Донавчання* (англ. Fine-Tuning). Після первинної адаптації поступово розморожуються глибші шари, і модель перенавчається з невеликим коефіцієнтом навчання (learning rate). Типові параметри для fine-tuning: optimizer: AdamW; learning rate: 2e-5-5e-5; batch size: 16-32; кількість епох: 5-15; scheduler: cosine decay

або stepLR. Для стабілізації навчання використовуються Dropout (0.1–0.3) та Weight Decay (0.01).

4. *Валідація*. Частина даних (10-20%) виділяється для валідації. Після кожної епохи обчислюються метрики Accuracy, Precision, Recall, F1-score, а також будується матриця плутанини (англ. Confusion Matrix) для аналізу правильності класифікації кожного класу.

5. *Тестування*. Після завершення навчання модель тестується на нових, раніше невідомих зображеннях (test set). Це дозволяє оцінити генералізаційну здатність системи - наскільки добре вона працює в реальних умовах.

Процес fine-tuning у контексті діагностики хвороб передбачає часткове перенавчання попередньо навченої моделі ViT на спеціалізованому наборі зображень сільськогосподарських культур (наприклад, Plant Disease Classification Merged Dataset) [32]. Основна ідея полягає в тому, щоб:

– використати вже наявні знання моделі про загальні форми, текстури та кольори;

– адаптувати останні шари до особливостей аграрних зображень — плям, уражень, дефектів листя;

– оптимізувати новий класифікаційний шар під конкретні класи хвороб.

У результаті fine-tuning дозволяє досягти високих показників точності навіть при обмеженій кількості даних. Наприклад, у дослідженнях Li et al. (2023) і Ullah et al. (2024) точність класифікації хвороб яблуні та пшениці після донавчання ViT-Tiny і AppViT перевищувала 97–99%.

На відміну від CNN, ViT не має природного inductive bias щодо локальних патернів, тому потребує більшої кількості даних або попереднього навчання [33]. Найчастіше ViT моделі переднавчаються на великих загальних наборах (наприклад, ImageNet-21k), а потім проходять донавчання (fine-tuning) на спеціалізованому наборі зображень сільськогосподарських культур.

Під час донавчання оптимізуються такі гіперпараметри, як: learning rate (швидкість навчання), batch size (розмір пакета), кількість епох, dropout, кількість

шарів та attention heads [34]. Дослідження показують, що навіть коротке донавчання протягом 5–10 епох на невеликому наборі (PlantVillage, Plant Disease Merged Dataset) забезпечує значне підвищення точності, перевищуючи результати CNN-аналогів.

Таким чином, у спрощеному вигляді основні етапи алгоритму навчання та донавчання моделі ViT можна подати так:

– підготовка даних: Завантаження датасету, розподіл на train/validation/test; попередня обробка (resizing, normalization, augmentation);

– ініціалізація моделі: завантаження попередньо навченої ViT, заміна вихідного шару класифікації на новий;

– навчання: визначення оптимізатора (AdamW), функції втрат (CrossEntropyLoss), планувальника швидкості навчання, проведення епох навчання з моніторингом loss та accuracy, збереження найкращої моделі за результатами валідації;

– донавчання (fine-tuning): поступове розморожування шарів, тонке налаштування параметрів навчання для покращення результатів;

– оцінювання якості моделі: розрахунок метрик Accuracy, Precision, Recall, F1-score, Аналіз помилок класифікації за допомогою матриці плутанини.

Щоб уникнути перенавчання (англ. overfitting), під час навчання застосовуються такі стратегії:

– рання зупинка (англ. Early Stopping): припинення навчання, якщо метрика валідації не покращується протягом кількох епох;

– регуляризація Dropout: випадкове вимикання нейронів під час навчання;

– аугментація даних: збільшення різноманітності зображень;

– Weight Decay: штраф за занадто великі ваги моделі.

Після завершення донавчання та підбору гіперпараметрів найбільш оптимальна модель ViT зберігається у форматі .pth (PyTorch) або ONNX для подальшої інтеграції у вебзастосунок [35]. Під час роботи користувач завантажує

фотографію листка, зображення проходить попередню обробку та подається до моделі ViT, яка повертає ймовірності належності до певних хвороб.

Таким чином, донавчання моделі Vision Transformer для класифікації хвороб рослин включає етапи підготовки даних, ініціалізації, fine-tuning, валідації та оцінювання якості результатів. Застосування transfer learning дозволяє використовувати потужність попередньо навчених моделей для спеціалізованих аграрних задач, забезпечуючи високу точність, стабільність і можливість інтеграції у веборієнтовану інформаційну систему.

2.4 Метрики оцінки якості моделей Vision Transformers

Оцінка якості роботи моделі є одним із ключових етапів побудови системи машинного навчання. У випадку діагностики хвороб сільськогосподарських культур метрики точності дозволяють визначити, наскільки коректно модель класифікує зображення листя за класами («здорове», «іржа», «септоріоз», «фітофтороз» тощо). Для цього використовуються кількісні показники ефективності класифікації, які обчислюються на основі порівняння прогнозів моделі з істинними мітками (англ. ground truth) [36].

Базою для розрахунку більшості метрик є матриця Confusion Matrix, що відображає кількість правильних і помилкових класифікацій для кожного класу (табл. 2.1):

– TP (True Positive) – кількість зображень, які дійсно належать до класу «хворе» і були правильно визначені моделлю;

– TN (True Negative) – кількість зображень «здорових» листків, які модель класифікувала правильно;

– FP (False Positive) – випадки, коли модель помилково віднесла здорове листя до класу «хворе»;

– FN (False Negative) – випадки, коли хворе листя було класифіковано як «здорове».

Для багатокласової класифікації (декілька типів хвороб) матриця плутанини розширюється відповідно до кількості класів. На основі Confusion Matrix здійснюється розрахунок описаних нижче метрик оцінки точності класифікації [37].

Таблиця 2.1 – Confusion Matrix для випадку двох класів

	Передбачено позитивне	Передбачено негативне
Фактично позитивне	True Positive (TP)	False Negative (FN)
Фактично негативне	False Positive (FP)	True Negative (TN)

Accuracy – найпростіша та найпоширеніша метрика, що показує загальну частку правильних передбачень серед усіх зразків:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (2.15)$$

У контексті діагностики хвороб рослин високе значення Accuracy означає, що модель у цілому правильно класифікує більшість зображень. Однак при дисбалансі класів (коли, наприклад, більшість зображень - здорове листя) Accuracy може вводити в оману, тому необхідно аналізувати й інші метрики [38].

Precision характеризує, наскільки достовірними є позитивні передбачення моделі, тобто яку частку з передбачених як «хворі» зображень справді є хворими:

$$Precision = \frac{TP}{TP+FP} \quad (2.16)$$

Високе значення Precision означає, що модель рідко помиляється, коли вказує на наявність хвороби. У сільському господарстві це важливо, щоб уникнути помилкових сигналів і зайвих витрат на обробку здорових культур.

Recall (повнота або чутливість, Sensitivity) визначає, яку частку всіх дійсно хворих рослин модель змогла правильно виявити:

$$Recall = \frac{TP}{TP+FN} \quad (2.17)$$

Високе значення Recall свідчить про здатність моделі виявляти всі випадки захворювань, навіть якщо частина прогнозів є помилковими [39]. Ця метрика особливо важлива для систем раннього виявлення, де пропуск хвороби може призвести до значних втрат урожаю.

F1-міра (F1-score). Оскільки між Precision і Recall існує компроміс (підвищення одного часто знижує інше), використовується F1-міра – гармонійне середнє між ними:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision+Recall} \quad (2.18)$$

F1-score є збалансованою метрикою, яка враховує як точність, так і повноту класифікації. У задачі розпізнавання хвороб рослин високий F1-score свідчить про здатність моделі надійно відрізняти всі типи захворювань і при цьому не створювати багато хибних спрацьовувань.

Окрім базових, у дослідженнях також застосовуються допоміжні метрики:

– Macro-average Precision/Recall/F1 – середнє значення показників для всіх класів, незалежно від кількості прикладів у кожному;

– Weighted-average Precision/Recall/F1 – середнє з урахуванням частки кожного класу;

– ROC-AUC (Area Under Curve) – площа під кривою помилкових спрацьовувань проти справжніх позитивів, що відображає якість класифікації при різних порогах прийняття рішень.

Ці метрики особливо корисні при аналізі моделей, що працюють з великим дисбалансом класів або мають різну важливість помилок (наприклад, пропуск хвороби важливіший, ніж помилкова тривога).

У процесі оцінювання якості моделі використання макро-орієнтованих класифікаційних метрик: macro Precision, macro Recall, macro F1-score забезпечує

збалансовану оцінку продуктивності моделі в умовах багатокласової та потенційно дисбалансної вибірки [40]. На відміну від традиційних агрегованих метрик, які обчислюють показники на рівні всіх зразків і є чутливими до домінування великих класів, макро-метрики усереднюють результати, отримані для кожного класу окремо, без зважування на їх частку у вибірці:

$$macroPrecision = \frac{1}{m}\sum_i^m Precision_i, \quad (2.19)$$

$$macroRecall = \frac{1}{m}\sum_i^m Recall_i, \quad (2.20)$$

$$macroF1 = \frac{1}{m}\sum_i^m F1_i, \quad (2.21)$$

де $Precision_i$, $Recall_i$, $F1_i$ – початкові метрики, розраховані для кожного класу i окремо за формулами 2.16-2.18,

m – кількість класів.

Під час навчання моделей ViT у контексті даної роботи метрики дозволяють:

– оцінити точність і стабільність ViT-моделі після донавчання;

– порівняти результати різних архітектур (ViT-Tiny, ViT-Base, DeiT, MobileViT);

– проаналізувати вплив гіперпараметрів (learning rate, кількість епох, batch size) на якість класифікації;

– виявити класи хвороб, які модель розпізнає найгірше, і вдосконалити набір даних шляхом розширення відповідних категорій.

Отже, для комплексної оцінки якості класифікації хвороб рослин доцільно використовувати комбінацію метрик – Accuracy, Precision, Recall та F1-score, які разом відображають як загальну правильність, так і здатність моделі виявляти всі випадки захворювань без надлишкових помилкових спрацьовувань.

Високі значення цих показників свідчать про ефективність розробленої моделі Vision Transformer та правильність побудови алгоритму її навчання. На основі отриманих метрик у наступному розділі буде проведено аналіз результатів роботи моделі та інтеграцію її у вебзастосунок для практичного використання в аграрній сфері.

Висновки до розділу 2

У другому розділі розглянуто моделі та методи, що лежать в основі побудови інформаційної системи автоматизованої діагностики хвороб сільськогосподарських культур із використанням архітектури Vision Transformer.

Проаналізовано принципи побудови трансформерів зору, їх структуру та ключові компоненти – шар розбиття на патчі, позиційне кодування, механізм багатоголової самоуваги (Multi-Head Self-Attention), резидуальні з’єднання та класифікаційний шар (MLP Head). Показано, що на відміну від традиційних згорткових нейронних мереж CNN, ViT здійснює глобальний аналіз зображення, враховуючи просторові зв’язки між усіма його частинами. Це дає змогу точніше визначати хвороби, ознаки яких можуть бути розсіяні по різних ділянках листка.

Описано етапи попередньої обробки та аугментації зображень, які забезпечують підвищення якості навчальних даних і стійкість моделі до змін освітлення, масштабу та орієнтації. Зокрема, застосування методів нормалізації, видалення фону, балансування яскравості, обертання та віддзеркалення дозволяє покращити генералізуючу здатність моделі та уникнути перенавчання.

Розроблено алгоритм навчання та донавчання моделей ViT на спеціалізованому датасеті хвороб рослин. Алгоритм передбачає поетапне розморожування шарів попередньо навченої моделі, налаштування гіперпараметрів (learning rate, batch size, кількість епох), а також оцінювання точності за допомогою валідаційної вибірки. Застосування transfer learning дозволяє адаптувати модель, навчену на великому наборі

ImageNet-21k, до специфічних ознак сільськогосподарських культур без потреби у великих обсягах даних.

Для оцінювання якості класифікації обгрунтовано побудову матриці плутанини, що відображає рівень розпізнавання кожного класу хвороб, та використання основних метрик точності - Accuracy, Precision, Recall та F1-score. Застосування цих показників забезпечує комплексну оцінку роботи моделі та дозволяє виявити класи, які потребують додаткового навчання.

Таким чином, у другому розділі сформовано методологічну основу побудови системи діагностики на базі ViT, що включає архітектурні рішення, алгоритм навчання, етапи обробки даних і критерії оцінки результатів.

3 НАВЧАННЯ ТА ОЦІНКА ЯКОСТІ МОДЕЛЕЙ VISION TRANSFORMER

3.1 Підготовка даних і моделі до навчання

Для проведення донавчання моделі Vision Transformer для виявлення хвороб сільськогосподарських культур було сформовано єдине обчислювальне та програмне середовище, яке забезпечує відтворюваність результатів та можливість подальшого дослідження оптимальної архітектури моделі шляхом налаштування її гіперпараметрів.

Обчислювальне середовище включало персональний комп’ютер із 64-розрядною операційною системою Windows 11, процесором класу Intel Core / AMD Ryzen та апаратним прискоренням за допомогою графічного процесора GPU, сумісного з технологією CUDA. Використання GPU є критично важливим для пришвидшення навчання трансформерних моделей, оскільки операції самоуваги та матричні множення мають високу обчислювальну складність при обробці великих пакетів зображень.

Програмна частина середовища була побудована на основі мови програмування Python та фреймворку PyTorch. Для роботи з трансформерами зору додатково застосовувалися бібліотеки:

– HuggingFace Transformers: для завантаження попередньо навчених моделей ViT та зручної роботи з конфігураціями;

– timm (PyTorch Image Models): як джерело реалізацій сучасних архітектур комп’ютерного зору;

– Torchvision: для базових перетворень зображень та завантаження датасетів;

– NumPy, Pandas, Matplotlib: для обробки результатів експериментів, побудови графіків кривих навчання та аналізу метрик;

– scikit-learn: для побудови Confusion Matrix та розрахунку додаткових метрик (класифікаційний звіт).

Усі залежності були згруповані у віртуальному середовищі Python (venv), що забезпечило ізоляцію проєкту від глобально встановлених пакетів та спростило повторне розгортання середовища.

Вихідні налаштування моделі та даних. На першому етапі було обрано базову попередньо навчену модель ViT-Tiny-Patch16-224, з патчами розміром 16×16 пікселів та вхідним розміром зображення 224×224 пікселі [41]. Така конфігурація відповідає стандартним реалізаціям ViT-Base/ViT-Tiny і забезпечує баланс між якістю класифікації та часом навчання. Модель ViT натренована для розпізнавання хвороб листя чотирьох культур (кукурудза, картопля, рис, пшениця), класифікованих за 14 класами, що представляють різні види рослин та їхній стан здоров’я.

Для донавчання моделі ViT-Tiny-Patch16-224 було використано Data Set, у якому представлено 14 сільськогосподарських культур, класифікованих за 88 класами (додаток А) [42], та досліджено вплив гіперпараметрів fine-tuning ViT із метою визначенні оптимальних параметрів донавчання моделі на доменних даних для підвищення точності розпізнавання хвороб та інтерпретованості результатів. Під час підготовки даних було застосовано аугментацію для збільшення зображень маочисельних класів за допомогою бібліотеки imgaug: горизонтальне віддзеркалення (50% імовірності); випадкове обрізання до 10%; зміна контрасту (0.75-1.5); додавання гаусівського шуму; зміна яскравості (0.8-1.2); невелике обертання (−5°…+5°) і зсув (shear −16°…+16°).

Під час ініціалізації базової моделі було виконано:

– завантаження ваг попередньо навченої ViT-моделі (pretrained on ImageNet);

– заміну вихідного класифікаційного шару head на новий повнозв’язний шар із кількістю виходів, що відповідає кількості класів у використовуваному датасеті (у даному випадку 88 класів хвороб і станів листя);

– налаштування початкових значень гіперпараметрів, які надалі оптимізувалися в процесі експериментів.

Для забезпечення коректної роботи моделі над зображеннями було прийнято такі базові налаштування препроцесингу: зміна розміру всіх вхідних зображень до 224×224; нормалізація пікселів до діапазону [0; 1] з подальшим приведенням до статистик, використаних під час попереднього навчання (mean, std від ImageNet); формування батчів фіксованого розміру (batch size) для ефективного використання пам'яті GPU.

Початкові гіперпараметри навчання. Для базової конфігурації навчання були встановлені такі початкові гіперпараметри (які далі змінювалися в процесі пошуку оптимальних значень):

– оптимізатор: AdamW (рекомендований для трансформерних архітектур);

– початкова швидкість навчання (англ. learning rate): у діапазоні від $2 \cdot 10^{-5}$ до $5 \cdot 10^{-4}$ (із вибором базового значення, наприклад $1 \cdot 10^{-4}$);

– розмір пакета (англ. batch size): 16 або 32 (залежно від обсягу доступної відеопам'яті);

– кількість епох: початково 5–10 для базових експериментів з подальшим уточненням;

– функція втрат: CrossEntropyLoss для багатокласової класифікації;

– регуляризація: weight decay (0.01) та dropout у внутрішніх шарах моделі;

– планувальник швидкості навчання (англ. scheduler): cosine annealing або step LR для плавної зміни learning rate протягом епох.

Для забезпечення відтворюваності результатів було зафіксовано початкове зерно генератора випадкових чисел (англ. random seed) у PyTorch, NumPy та стандартному модулі random. Це дозволило зменшити вплив випадковості в ініціалізації ваг і формуванні батчів на кінцеві метрики.

Розподіл даних та початковий протокол експериментів. На рівні вихідних налаштувань також було визначено схему розподілу даних на три частини: train data (70%): для безпосереднього навчання моделі; validation data (10%): для контролю процесу навчання, налаштування гіперпараметрів та ранньої зупинки;

test data (20%): для підсумкової оцінки якості класифікації. Тестова вибірка містила 15 850 зображень, що охоплюють 88 класів станів рослин.

3.2 Базова моделі Vision Transformer та стратегії її донавчання

Для створення ефективної системи автоматизованої діагностики хвороб рослин на основі трансформерів зору важливим є вибір вихідної архітектури моделі та визначення оптимальної стратегії її донавчання fine-tuning. Опишемо базову структуру моделі Vision Transformer, яка використовувалася в дослідженні, а також три ключові підходи до донавчання: *freeze* – повне замороження**,** *partial fine-tuning* – часткове донавчання та *full fine-tuning* – повне донавчання.

Вибір базової моделі Vision Transformer. У роботі використовувалася попередньо навчена модель ViT-Base-Patch16-224, реалізована у фреймворку HuggingFace Transformers. Основні характеристики моделі:

- patch size: 16×16;
- image size: 224×224;
- embedding dimension: 768;
- кількість шарів трансформера: 12;
- кількість голів самоуваги: 12;
- кількість параметрів: ~86 млн.

Модель попередньо навчена на великому наборі ImageNet-21k, що забезпечує її здатність розпізнавати універсальні візуальні патерни (краї, переходи кольорів, текстури). Для задачі класифікації хвороб був замінений тільки фінальний класифікаційний шар MLP Head на новий, відповідно до 88 класів, визначених у навчаючому датасеті.

Процес fine-tuning визначає, які параметри моделі можуть змінюватися під час навчання, а які залишаються замороженими. Це впливає на: швидкість навчання, вимоги до апаратного забезпечення, ризик перенавчання, кінцеву

точність. У роботі було застосовано три стратегії донавчання, опишемо їх більш детально.

Стратегія Freeze / Feature Extraction – «заморожена модель». Це найпростіший підхід, при якому усі шари ViT заморожуються їхні ваги не оновлюються), а навчається лише класифікаційна голова – вихідний класифікаційний шар. Механіка методу: параметри усіх шарів attention та feed-forward залишаються фіксованими, під час навчання змінюються тільки ваги фінального MLP Head.

Переваги цієї стратегії: найшвидше навчання (низькі витрати GPU); мінімальний ризик перенавчання; підходить для невеликих або сильно дисбалансованих наборів даних. Недоліки стратегії: модель майже не адаптується до нової доменної задачі, специфіки зображень листя; ефективність значно нижча на «польових» даних, ніж на повному fine-tuning. Цей підхід застосовувався як базова точка відліку (англ. baseline) для порівняння з іншими стратегіями.

Стратегія Partial Fine-Tuning – часткове донавчання. У цьому підході розморожуються тільки верхні (кілька останніх) блоки трансформера, які відповідають за високорівневі ознаки та абстракції. Механіка методу: заморожується приблизно 70–80 % найнижчих шарів моделі, донавчаються тільки верхні кілька шарів ViT + класифікаційна голова: останні 2–4 encoder blocks, layer norm, класифікаційний шар.

Переваги стратегії: забезпечує баланс між швидкістю і точністю; добре підходить при середніх обсягах даних; адаптація до локальних особливостей хвороб та нової задачі помірно висока; менша ймовірність перенавчання, ніж при full fine-tuning. Недоліки стратегії: складніше підібрати, які блоки заморозити; повільніше, ніж freeze-режим; потребує точного підбору learning rate (занадто високий призведе до руйнування pretrained-представлень).

Часткове донавчання є одним із найпопулярніших підходів у практичних застосуваннях ViT – особливо тоді, коли дані різнорідні (фото з різним фоном,

освітленням, сортами рослин), коли класи трихи відрізняються від базових та якщо є обмеження на час та ресурси.

Full Fine-Tuning – повне донавчання. Найпотужніший підхід, при якому розморожуються усі шари трансформера, і модель навчається цілком, неначебто з нуля, але з ініціалізованими вагами. Механіка методу: оновлюються всі параметри моделі ViT: Q/K/V матриці, MLP, attention layers, feed-forward layers, positional embeddings, patch embeddings, класифікаційна голова.

Переваги стратегії: краща якість, максимальна точність; повна адаптація моделі під домен рослин нової доменної області; дозволяє досліджува вплив гіперпараметрів у всіх блоках ViT; дає найкращі результати на великих датасетах (десятки тисяч зображень). Недоліки стратегії: найповільніший режим, довге навчання, вимагає значних GPU-ресурсів; сильний ризик перенавчання при малих датасетах; високі обчислювальні вимоги; складніше налаштовувати learning rate, weight decay і scheduler. Підхід full fine-tuning використовувався як кінцева стадія експериментів після визначення оптимальних гіперпараметрів.

У таблиці 3.1 наведено порівняння стратегій донавчання у контексті розв’язання поставленої задачі.

Таким чином, для діагностики хвороб рослин найкращий баланс забезпечує partial fine-tuning, оскільки він дозволяє досягти високої точності навіть при обмеженому наборі польових даних. Однак остаточний вибір стратегії залежить від результатів експериментів, що будуть наведені у наступних параграфах розділу.

3.3 Метод підбору гіперпараметрів моделі Random Search

Після вибору базової моделі та визначення стратегій донавчання важливим етапом є оптимізація гіперпараметрів, що істотно впливають на точність та стабільність навчання трансформерної моделі ViT. Правильно підібрані значення learning rate, batch size, scheduler, weight decay та dropout забезпечують швидку збіжність моделі та зменшують ризик перенавчання. В даній роботі було

застосовано метод Random Search – випадковий пошук, який є простим та ефективним підходом до автоматичного налаштування гіперпараметрів.

Таблиця 3.1 – Коротка порівняльна таблиця стратегій донавчання

	Стратегія		
	Freeze	Partial fine-tuning	Full fine-tuning
Розморожені шари	Тільки голова	Верхні N блоків	Всі шари
Розмір навчального data set	малий	середній	великий
Швидкість	дуже висока	середня	низька
Ризик overfitting	мінімальний	помірний	високий (на малих даних)
Точність	низька	висока	максимальна
Потреба в GPU	мінімальна	середня	висока

Random Search випадково генерує декілька наборів гіперпараметрів у заданих діапазонах, після чого для кожного набору виконується повне навчання моделі. На відміну від систематичного перебору (у методі Grid Search, наприклад), Random Search не охоплює весь простір параметрів, але дозволяє швидко знайти вдалі комбінації при значно менших витратах обчислювальних ресурсів.

Для реалізації методу було створено окремий скрипт random_search.py, який:

– завантажує базовий конфігураційний файл;

– генерує випадкові значення гіперпараметрів у заданих діапазонах (табл. 3.2, рис. 3.1);

– зберігає згенерований конфіг у тимчасовий YAML-файл;

– запускає основний процес навчання main() із новим набором параметрів;

– повторює процедуру для визначеної кількості запусків.

Під час реалізації методу після визначення кількості випадкових запусків N_TRIALS виконується цикл, який багаторазово створює випадкові конфігурації та запускає з ними навчання моделі (рис. 3.2).

Таблиця 3.2 – Використані діапазони для Random Search

Гіперпараметр	Варіанти
lr_head	log-uniform у діапазоні від $1 \cdot 10^{-6}$ до $1 \cdot 10^{-3}$
lr_backbone	log-uniform у діапазоні від $1 \cdot 10^{-3}$ до $1 \cdot 10^{-4}$
batch_size	випадковий вибір із {8, 16, 32}

```python
def sample_cfg(base_cfg):  1 usage
    #Генерує випадкову конфігурацію
    cfg = copy.deepcopy(base_cfg)

    cfg["lr_head"] = 10 ** random.uniform(-5, -3)
    cfg["lr_backbone"] = 10 ** random.uniform(-6, -4)
    cfg["batch_size"] = random.choice([8, 16, 32])

    return cfg
```

Рисунок 3.1 – Задання діапазонів генерації гіперпараметрів

```python
for i in range(N_TRIALS):
    cfg_i = sample_cfg(base_cfg)

    # Збереження тимчасового конфігу
    cfg_path = f"configs/random_trials/random_{i}.yaml"
    with open(cfg_path, "w", encoding="utf-8") as f:
        yaml.safe_dump(cfg_i, f, allow_unicode=True)

    run_name = f"random_trial_{i}"
    print(f"\n===== Запуск {i+1}/{N_TRIALS}: {run_name} =====\n")

    main(cfg_path=cfg_path, run_name=run_name)
```

Рисунок 3.2 – Генерація гіперпараметрів у циклі

Цикл виконує такі дії:

– генерує випадковий набір гіперпараметрів за допомогою sample_cfg();

– створює тимчасовий YAML-файл, у якому зберігає згенерований набір параметрів;

– формує ім’я експерименту, щоб кожен запуск мав окрему папку та окремі логи;

– запускає повне навчання моделі з цими параметрами, викликаючи основну функцію main().

Таким чином, кожна ітерація циклу – це окремий повний експеримент з унікальними гіперпараметрами.

Оскільки одне повне навчання моделі займає приблизно 20 годин, виконання великої кількості експериментів було б надто дорогим у часі. У практиці глибинного навчання для обчислювально важких моделей часто застосовують мінімальні Random Search серії з 3–5 запусків, що дозволяє: протестувати різні області простору параметрів; знайти робочу або близьку до оптимальної конфігурацію; уникнути багатоденного або тижневого перебору десятків значень.

Було виконано 5 запусків, що забезпечило достатню варіативність результатів при прийнятному часі обчислень. У результаті були знайдені наступні оптимальні діапазони: batch_size: 16; lr_head: $5 \cdot 10^{-4}$; lr_backbone: $1 \cdot 10^{-5}$. Ця конфігурація забезпечила найвищу точність серед усіх протестованих варіантів та демонструвала стабільне зменшення втрат під час навчання. Для оцінки кожної комбінації параметрів використовувалися наступні метрики: Accuracy, Precision (macro), Recall (macro), F1-score (macro).

Вибір macro-метрик зумовлений наявністю значного дисбалансу між класами у вихідному датасеті, що могло призвести до переадаптації моделі до найбільш репрезентованих класів. Тому оцінка якості моделі лише за точністю була б недостатньою та потенційно хибною, оскільки accuracy може залишатися високою навіть за слабкої продуктивності на малочисельних класах.

Таким чином, у результаті застосування методу оптимізації Random Search було визначено набір гіперпараметрів, які забезпечили найкращий баланс між точністю, швидкістю навчання та стійкістю моделі до перенавчання. Ці параметри стали основою для експериментів, описаних у наступному параграфі.

3.4 Методика дослідження впливу гіперпараметрів та стратегій донавчання на якість класифікації

Для дослідження впливу гіперпараметрів та стратегій донавчання на якість класифікації було проведено серію експериментів із трьома моделями Vision Transformer (ViT), які відрізнялися налаштуваннями навчання. Метою експериментів було:

– визначити оптимальну швидкість навчання для класифікаційної голови моделі та бекбону;

– оцінити ефективність часткового заморожування (freeze epochs);

– проаналізувати вплив вагового семплінгу (WeightedRandomSampler) на баланс класів;

– порівняти криві навчання та метрики для кожної конфігурації.

Усі моделі навчалися на одному датасеті, мали ідентичні параметри препроцесингу та однакову кількість епох (10), що забезпечує коректність порівняння. Було проведено три окремі експерименти (див. табл. 3.3).

Ці три конфігурації дають змогу ізольовано оцінити вплив:

– глибини замороження (freeze_backbone_epochs);

– швидкості навчання тіла моделі (lr_backbone);

– наявності вагового семплінгу (use_weighted_sampler).

Нижче наведено детальне пояснення кожного гіперпараметра, що використовувався у файлах експериментів.

1. *Загальні гіперпараметри* (спільні для всіх експериментів):

– *image_size: 224* – стандартний розмір входу для Vision Transformer, ViT Base та ViT Tiny навчаються на 224×224 під час попереднього pretraining, тому використання цього розміру гарантує сумісність із Patch Embedding;

– *batch_size: 16* – оптимальний баланс між якістю градієнтів (чим більший batch, тим стабільніше навчання) та доступною відеопам'яттю (GPU), є типовим вибором для ViT batch_size 16-32;

– *epochs: 10* – фіксована кількість епох дає змогу порівнювати моделі в однакових умовах, уникати перенавчання, та пришвидшити експерименти, для fine-tuning ViT часто достатньо 5-10 епох;

– *weight_decay: 0.01* – регуляризаційний параметр, який запобігає збільшенню ваг, зменшує перенавчання, є рекомендованим для AdamW за документацією Transformers;

– *use_amp: true* – включення автоматичного змішаного точного обчислення (FP16) знижує використання пам'яті, прискорює навчання не впливає на точність.

Таблиця 3.3 – Особливості трьох експериментів з донавчання моделі ViT

Експеримент	Особливість	Мета
Exp1 (default.yaml)	Стандартні гіперпараметри, LR_backbone = $5 \cdot 10^{-5}$, freeze 1 епоха, sampler = ON	Базовий контрольний експеримент
Exp2 (exp2_freeze_backbone.yaml)	Заморожування тіла на 3 епохи, дуже малий LR_backbone = $1 \cdot 10^{-5}$	Перевірити користь довшого freeze та обережного оновлення бекбону
Exp3 (exp3_no_sampler.yaml)	Повністю аналогічна Exp1, але sampler = OFF	Перевірити вплив балансування класів на навчання

2. *Гіперпараметри, що розрізняють експерименти*. Найважливіші відмінності між моделями – дві швидкості навчання (табл. 3.4) та кількість заморожених епох (табл. 3.5), а також наявність/відсутність балансувального семплера (табл. 3.6).

Швидкість навчання голови моделі – lr_head = 5e-4. Це значення однакове у всіх експериментах, що обумовлено наступним: нова класифікаційна голова (MLP Head) ініціалізується випадково. Їй потрібно навчитися швидше, ніж решті моделі, тому LR для голови зазвичай у 5-10 разів більший, ніж для бекбону.

Таблиця 3.4 – Швидкість навчання бекбону – lr_backbone

Експеримент	Значення	Обґрунтування
Exp1	$5 \cdot 10^{-5}$	Стандартний LR для часткового fine-tuning ViT
Exp2	$1 \cdot 10^{-5}$	Обережне навчання → перевірка стабільності
Exp3	$5 \cdot 10^{-5}$	Як у Exp1, але без самплера

LR_backbone має бути малим, оскільки бекбон вже навчився на ImageNet. Занадто великий LR може зруйнувати корисні вагові представлення. Тому значення 1e-5 - 5e-5 – загальноприйнята практика для ViT.

Заморожування бекбону (табл. 3.5) потрібне, оскільки перших епохах модель має: адаптуватись до нового датасету, навчити класифікатор розрізняти класи, уникнути деструктивних градієнтів у бекбоні.

Таблиця 3.5 – Замороження бекбону – freeze_backbone_epochs

Експеримент	Значення	Обґрунтування
Exp1	1	Легка стабілізація навчання
Exp2	3	Довгий freeze → перші епохи вчимо лише голову
Exp3	1	Як у базовому експерименті

Датасети хвороб рослин майже завжди сильно незбалансовані по класах. Із метою врахування проблеми дисбалансу класів було використано метод Class-balanced sampling, який передбачає під час кожної епохи створення батчів із однаковим представленням прикладів кожного класу. Для цього підраховують частоту n_i кожного класу у навчаючих даних та обчислюють вагу ω_i, обернену до частоти класу:

$$\omega_i = \frac{1}{n_i}. \tag{3.1}$$

Такми чином семплер забезпечує збалансоване представлення класів під час навчання незалежно від реального представлення класів у датасет. Метод не дублює дані, а модель бачить приблизно однакову кількість прикладів кожного класу в процесі навчання.

Для реалізації методу було створено семплер WeightedRandomSampler (рис. 3.3), який змінює спосіб вибірки даних під час навчання, вирівнюючи ймовірність появи класів. Семплер WeightedRandomSampler було застосовано у експериментах Exp1 та Exp2. Мета експерименту Exp3 – перевірити, наскільки модель погіршить якість виявлення хвороб без вирівнювання класів. Це дасть пряме розуміння, чи модель переобтяжує домінуючі класи.

```
def build_sampler_from_dataset(dataset, class2id):
    labels = []
    for p in dataset.paths:
        cls = Path(p).parent.name
        labels.append(class2id[cls])

    cnt = Counter(labels)
    weights = [1.0 / cnt[l] for l in labels]
    return WeightedRandomSampler(weights, num_samples=len(weights), replacement=True)
```

Рисунок 3.3 – Реалізація семплеру WeightedRandomSampler

Протокол проведення експериментів. Усі моделі навчалися за єдиним протоколом:

– розподіл даних: 70% – train data, 10% – validation data, 20% test data;

– підготовка даних: ресайз 224×224, нормалізація під ImageNet, аугментації (Random Flip, Random Rotate: по 20% для кожного класу);

– навчання: оптимізатор AdamW, раннє заморожування бекбону, логування loss та accuracy;

– збереження: найкраща модель, збереження кривих навчання для порівняння;

– валідація: здійснювалася побудова Confusion Matrix та оцінювалися метрики Accuracy, Precision, Recall, F1-score (macro, основна метрика);

– порівняння моделей: основні точки аналізу були наступними – вплив freeze на стабільність та темп навчання, вплив швидкості навчання бекбону, вплив вирівнювання класів, ефективність моделі у малочисельних класах.

Для вирішення проблеми дисбалансу класів під час навчання моделі було враховано ваги класів у функції втрат. Тому формула 2.13 для обчислення функції втрат була модифікована:

$$L = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}\omega_j y_{i,j}\log(\hat{y}_{i,j}) \quad (3.2)$$

де n – кількість зразків у батчі,

$\omega_j = \frac{N}{n_j}$ – вага класу j (N – загальна кількість зразків, n_j – кількість зразків у класі j),

m – кількість класів,

$y_{i,j}$ – істинна мітка (0 або 1),

$\hat{y}_{i,j}$ – передбачена ймовірність належності до класу j.

Комбінація семплеру WeightedRandomSampler та використання ваг у функції втрат значно підвищує стійкість моделі до дисбалансу. Оскільки семплер вирівнює

розподіл даних у у батчах під час навчання, забезпечуючи рівномірність вхідних прикладів, тоді як зважена функція втрат loss робить модель чутливою до важливості малочисельних класів. У результаті модель не переорієнтовується на домінуючі класи, покращує якість передбачень для всіх категорій хвороб і зменшує ризик деградації таких метрик, як recall та F1-score.

Таким чином, узагальнене призначення трьох експериментів є наступним. Експеримент Exp1 є базовим, дає еталонну точку порівняння. Експеримент Exp2 здійснює перевірку, чи покращить результат стабільність і узагальнення: довге freeze + дуже малий LR. Експеримент Exp3 перевіряє реальний вплив дисбалансу класів, проводиться без weighted sampler.

3.3 Аналіз отриманих результатів

У дослідженні виконано серію експериментів із донавчання попередньо натренованої моделі ViT-Tiny-Patch16-224 для класифікації зображень рослин. Архітектура моделі залишалася незмінною, а варіювалися лише гіперпараметри навчання, зокрема: глибина заморозки шарів (freeze depth), стратегія fine-tuning (partial та full), параметри оптимізації, а також методи урахування дисбалансу класів (класові ваги та ваговий семплінг). Метою експериментів був пошук оптимальної комбінації гіперпараметрів, яка забезпечує найвищу якість класифікації без модифікації базової архітектури ViT.

У дослідженні було сформовано три fine-tuned варіантів моделі ViT-Tiny-Patch16-224, які відрізнялися гіперпараметрами навчання, глибиною заморозки шарів: 1) exp_1 – default: базове донавчання моделі без заморожування шарів та без модифікацій семплера; 2) exp_2 – freeze_backbone: заморожено backbone (основні трансформер-шари), донавчається лише «голова» класифікатора; 3) exp_3 – no_sampler: донавчання без семплера класів, тобто модель бачить дані з природним дисбалансом. Хоча архітектура моделі залишалася незмінною, різні

конфігурації навчання привели до формування різних наборів ваг, що дозволило порівняти якість отриманих екземплярів моделі.

На рисунках 3.3-3.6 представлено графіки втрат Loss та точності Accuracy для навчаючих і валідаційних даних, отримані за епохами під час дослідження трьох варіантів конфігурацій гіперпараметрів моделі ViT.

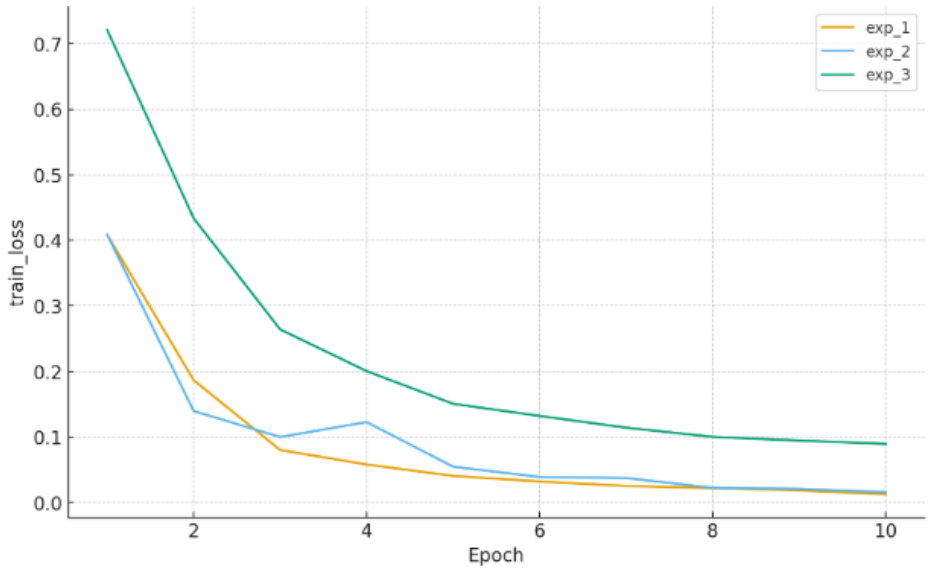

Рисунок 3.3 – Значення втрат на навчаючих даних

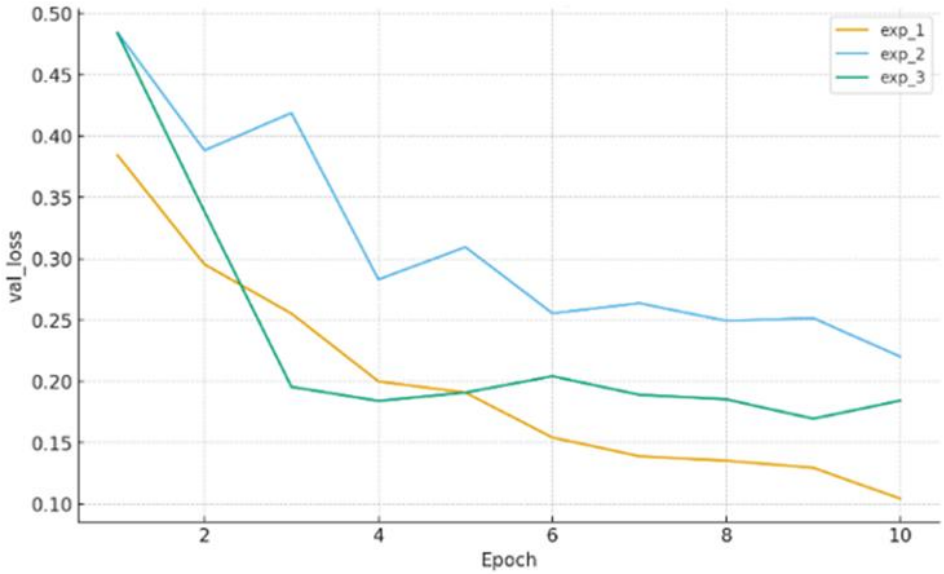

Рисунок 3.4 – Значення втрат на валідаційних даних

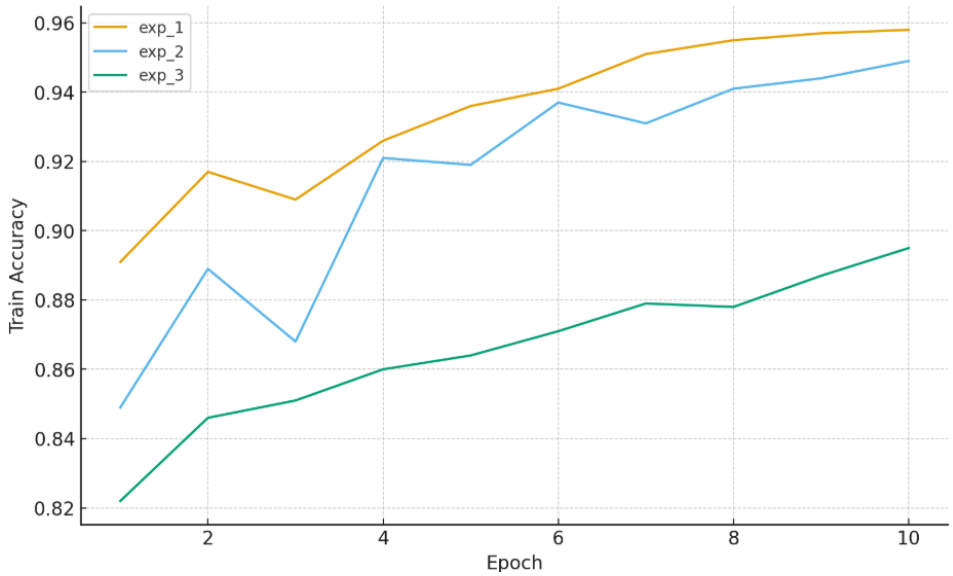

Рисунок 3.5 – Значення Accuracy на навчаючих даних

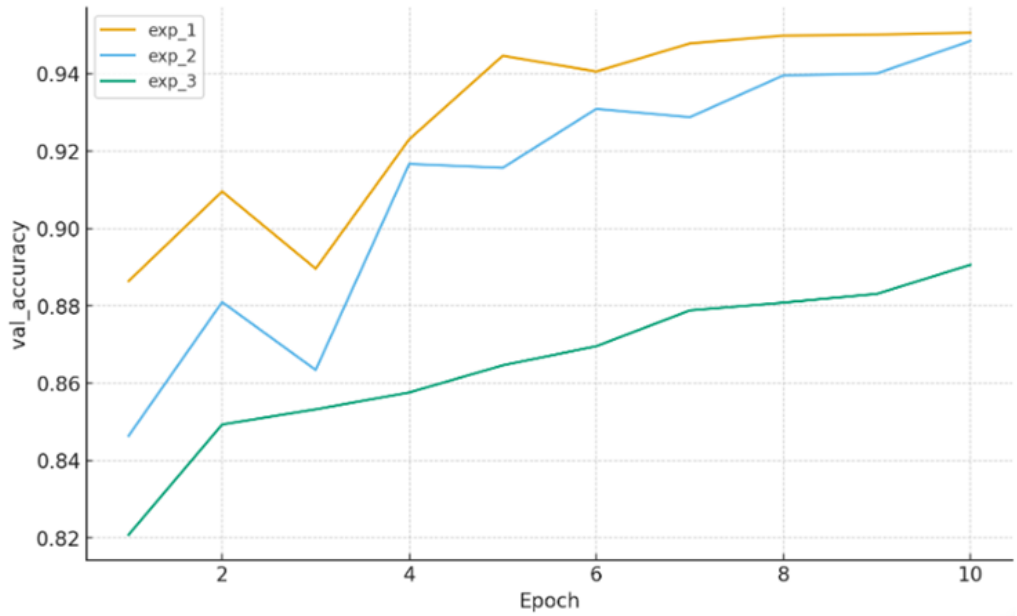

Рисунок 3.6 – Значення Accuracy на валідаційних даних

Оцінювання якості отриманих конфігурацій моделі ViT здійснювалося на основі сукупності показників, що характеризують ефективність навчання та здатність моделі до узагальнення: loss, accuracy, а також метрик Precision, Recall та F1-score у macro-варіанті для Train/Validation/Test data. Вибір macro-метрик зумовлений наявністю значного дисбалансу між класами у вихідному датасеті.

У таблиці 3.6 показано результати оцінки моделі ViT за значеннями функції втрат Loss та точності Accuracy, отриманими на навчаючій, валідаційній та тестовій вибірках для трьох експериментів підбору гіперпараметрів. Значення макро метрик Precision, Recall, F1-score наведено у таблиці 3.7.

Таблиця 3.6 – Значення Loss та Accuracy для трьох експериментів

Експеримент	Train		Validation		Test	
	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
Exp1	0,012	0,958	0,104	0,950	0,041	0,954
Exp2	0,015	0,949	0,220	0,948	0,074	0,935
Exp3	0,089	0,895	0,184	0,890	0,127	0,873

Таблиця 3.7 – Значення Precision, Recall, F1-score (macro) для трьох експериментів

Експеримент	Вибірка	Метрики		
		Precision (macro)	Recall (macro)	F1-score (macro)
Exp1	Train	0,967	0,960	0,963
	Validation	0,938	0,933	0,935
	Test	0,940	0,931	0,933
Exp2	Train	0,940	0,935	0,937
	Validation	0,924	0,920	0,921
	Test	0,905	0,897	0,900
Exp3	Train	0,912	0,903	0,907
	Validation	0,886	0,879	0,882
	Test	0,868	0,860	0,864

Проведений аналіз трьох варіантів конфігурації гіперпараметрів моделі ViT (exp_1, exp_2 та exp_3) показав суттєві відмінності у якості класифікації.

У експерименті *exp_1 (default)* варіант конфігурації гіперпараметрів демонструє найкращу збалансованість між швидкістю збіжності та якістю узагальнення: train_loss швидко зменшується до ≈0,01; val_loss стабільно знижується, без різких стрибків. Цей варіант продемонстрував найменші значення Train Loss та Test Loss: 0,012 та 0,041 відповідно, що супроводжувалося найвищими показниками точності Accuracy на тестових даних 0,954. Висока узгодженість між Train Accuracy =0,958 та Test Accuracy =0,954 свідчить про відсутність критичного перенавчання моделі та високу якість репрезентацій, адаптованих до нового домену. Додатково macro-метрики на рівні Train/Validation/Test підтвердили ефективність даної конфігурації (F1-score: 0,963, 0,935, 0,933 відповідно), що вказує на збалансовану здатність моделі правильно розпізнавати як багаточисельні, так і малочисельні класи. Саме базовий режим навчання виявився оптимальним: модель вільно адаптує всі шари, що дозволяє максимально ефективно підлаштуватися під специфіку датасету.

У експерименті *exp_2 (freeze_backbone)*, у якому було заморожено основні шари ViT і донавчалися лише параметри класифікаційної голови, результати очікувано гірші за default: train_loss падає швидко, але це обмежений ефект, бо backbone не оновлюється; val_loss має значно більшу варіативність (містить «гойдалки»), що означає недостатню адаптивність моделі; val_accuracy росте, але зупиняється близько 0,948; train_accuracy сягає значень 0,949, що нижче ніж у default. Причина – модель намагається компенсувати фіксовану трансформер-частину, але цього недостатньо для повної адаптації до нового датасету.

Спостерігалося зростання Train Loss та Test Loss та зниження точності на тестових даних до 0,935. Хоча узгодженість Train-Test залишалася відносно прийнятною, значне збільшення loss на тестовій вибірці вказує на знижену здатність моделі до узагальнення, що є наслідком недостатньої адаптації трансформерних ваг до цільової області даних. Macro-метрики також підтвердили

спад якості (F1-score: 0,937 для Train, 0,900 для Validation, 0,907 для Test), що свідчить про погіршення класифікації класів із меншою представленістю.

Експеримент *exp_3 (no_sampler)*, у якому модель навчалася без корекції дисбалансу класів, показав найгірші результати серед усіх конфігурацій. У експерименті навчання без збалансованого семплера призводить до того, що модель бачить класи з різною частотою. Через дисбаланс: train_loss стартує зі значно вищих значень; val_loss зменшується, але повільніше в порівнянні з двома іншими моделями; val_accuracy підвищується плавно, але досягає лише 0,890, а train_accuracy – 0,895, що є найнижчим результатом серед усіх експериментів. Модель підлаштовується переважно під багаточисельні класи, тому якість прогнозування на менш представлених класах погіршується.

Значення train accuracy та test accuracy становили відповідно 0,895 та 0,873, що, разом із високими значеннями train loss та test loss, вказує на як загальну нижчу якість навчання, так і зменшення можливостей моделі до узагальнення. Macro-метрики істотно знизилися (F1-score: 0,907 для Train, 0,882 для Validation, 0,864 для Test), що підтверджує систематичну упередженість. Це підкреслює критичну роль методів балансування при роботі з ViT у задачах класифікації, особливо при обмеженій кількості даних.

Отримані результати дозволяють зробити декілька важливих висновків. По-перше, повне донавчання без заморожування шарів забезпечує найбільшу гнучкість та адаптивність моделі до нового домену, що призводить до найкращих показників якості та мінімальної різниці між train та test продуктивністю. По-друге, заморожування трансформерних шарів може призводити до недоадаптації, що погіршує якість класифікації, навіть за високої точності на тренуваній вибірці. По-третє, відсутність механізмів корекції дисбалансу даних істотно погіршує якість розпізнавання класів та призводить до систематичного зниження macro-метрик. Таким чином, оптимальним підходом до донавчання ViT у дослідженому сценарії є використання повного fine-tuning у поєднанні з методами балансування вибірки. Ця конфігурація забезпечує найкращу узагальнюючу здатність моделі та

максимальну рівномірність класифікації класів різної частотності, що є критично важливим у реальних застосуваннях у сфері розпізнавання рослин.

За результатами порівняння трьох експериментів найкращою виявилась конфігурація гіперпараметрів exp_1 (default), яка продемонструвала найвищу стабільність навчання та максимальну точність під час донавчання моделі. Отже, для подальшого використання найкращим вибором є default-конфігурація, а freeze_backbone може застосовуватися лише за умов обмежених ресурсів або коли потрібне швидке донавчання без повного тренування.

Нормалізована Confusion matrix для тестових даних показує майже ідеальну діагональ, що свідчить про мінімум помилок під час діагностування хвороб (рис. 3.7). Нормалізація була проведена для більш інформативної оцінки якості класифікації набору даних, що містить велику кількість класів (88), які є незбалансованими. Вона демонструє відсоткові співвідношення правильних і помилкових прогнозів без прив’язки до кількості зразків у кожному класі.

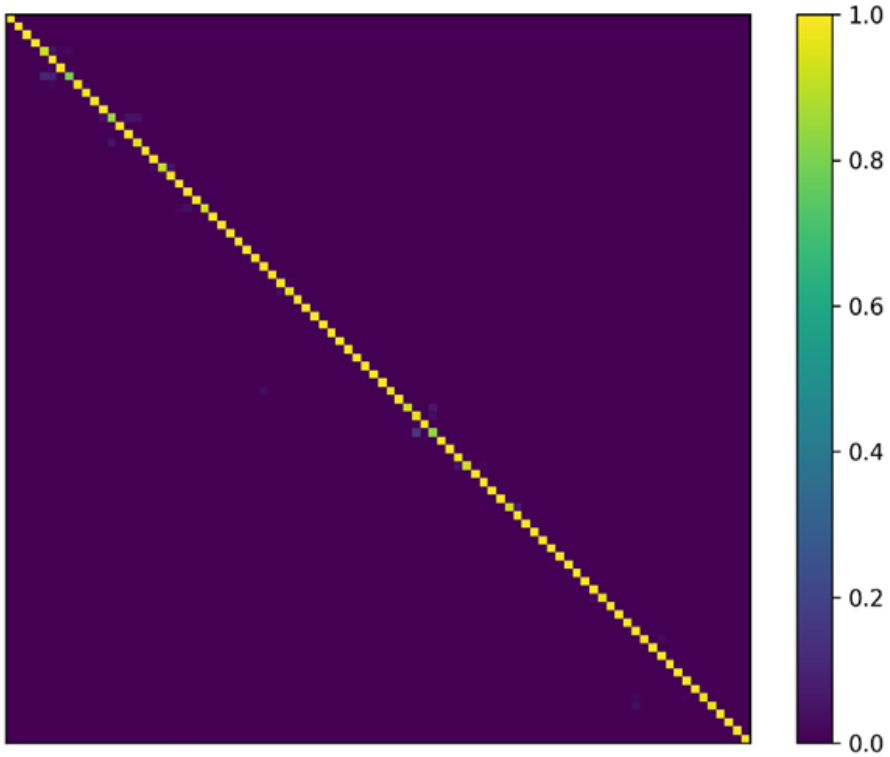

Рисунок 3.7 – Нормалізована Confusion matrix на тестових даних

Хвороби рослин, які показали найнижчу точність, представлено у таблиці 3.8. Попри це, навіть найнижчі метрики залишаються в межах 0,83-0,90, що є високим показником для 88 класів. Багаточисельні класи модель обробляє відмінно (табл. 3.9). Один із прикладів високої точності класифікації: Grape_black_rot (2278 тестових зразків): Precision: 0,98, Recall: 0,98, F1: 0,99.

Таблиця 3.8 – Проблемні класи зі зниженої точністю на тестових даних

Клас	Precision	Recall	Примітки
Cassava__bacterial_blight	0,84	0,92	Схожість з іншими хворобами цієї культури
Cassava__healthy	0,97	0,79	Модель часто плутає зі слабкими ознаками хвороб
Rice__leaf_blast	0,90	0,83	Ускладнення через неоднорідні патерни плям
Rice__healthy	0,90	0,96	Деякі хвороби мають легкі прояви
Apple__rust	0,87	0,92	Помилки переважно у precision через схожість забарвлення

Таблиця 3.9 – Багаточисельні класи з високою точністю на тестових даних

Клас	Кількість зразків	Accuracy exp_1	Accuracy exp_2	Accuracy exp_3
Grape__black_rot	2278	0,99	0,99	0,98
Soybean__healthy	1200	0,99	0,98	0,98
Tomato__yellow_leaf_curl_virus	643	0,98	0,97	0,97
Corn__healthy	233	0,99	0,98	0,98
Soybean__caterpillar	662	0,98	0,97	0,96

Таким чином, отримані результати свідчать, що модель exp_1 (default): 1) має високий рівень узагальнення та працює стабільно на даних, яких не бачила під час навчання; 2) демонструє високу точність (≈95%) при великій кількості класів – це типовий рівень state-of-the-art моделей ViT для задач сільськогосподарської діагностики; 3) виявляє складні хвороби навіть при наявності візуальних перешкод (плями, шум, освітлення). Дає найбільшу стабільність серед усіх експериментів: мінімальні коливання функції втрат, відсутність перенавчання, збалансовані метрики для малочисельних класів та класів з великою кількістю зразків. Саме цей варіант fine-tuned моделі ViT-Tiny-Patch16-224 обрано для інтеграції в інформаційну систему для діагностики хвороб рослин.

Висновки до розділу 3

У третьому розділі описано донавчання та оцінку якості різних варіантів конфігурації моделі ViT для задачі виявлення хвороб сільськогосподарських культур за зображеннями листя. На етапі підготовки даних та обчислювального середовища сформовано відтворювану інфраструктуру на основі Python, PyTorch та бібліотек HuggingFace Transformers, timm, Torchvision, scikit-learn. Забезпечено препроцесинг зображень (ресайз до 224×224, нормалізація до статистик ImageNet, формування батчів). Вибір попередньо навченої моделі ViT-tiny-patch16-224 з патчами 16×16 і заміна класифікаційної голови на 88 виходів дозволили ефективно перенести знання, отримані на великому наборі ImageNet, у предметну область діагностики хвороб рослин.

Здійснено аналіз та обгрунтовано вибір стратегій донавчання трансформерів зору. Розглянуто три підходи до fine-tuning: повне заморожування моделі (freeze), часткове донавчання верхніх блоків (partial fine-tuning) та повне донавчання всіх шарів (full fine-tuning). Показано, що для задачі класифікації хвороб рослин найкращий баланс між точністю, стійкістю до перенавчання та вимогами до ресурсів забезпечує часткове або обережне повне донавчання з малим значенням

швидкості навчання для бекбону. Для налаштування гіперпараметрів застосовано метод Random Search, що дозволило звузити простір пошуку та підібрати значення learning rate, weight decay, dropout, кількості розморожених шарів і warmup, які забезпечують стабільне зменшення функції втрат і зростання точності.

Методика експериментального дослідження впливу стратегій донавчання та гіперпараметрів на якість класифікації включала три конфігурації, що відрізнялися глибиною заморожування бекбону, величиною швидкості навчання для нього та використанням або відсутністю вагового семплера для балансування класів. У процесі оцінювання якості моделей використовувалися макро-орієнтовані класифікаційні метрики (macro Precision, macro Recall, macro F1-score), що забезпечують збалансовану оцінку продуктивності моделі в умовах багатокласової та потенційно дисбалансної вибірки.

Аналіз отриманих результатів показав, що конфігурація гіперпараметрів із коротким заморожуванням бекбону, малим значенням learning rate для нього та використанням вагового семплера продемонструвала найкращий компроміс між точністю, стійкістю й обчислювальними витратами: тренувальна втрата монотонно прямує до малих значень, валідаційна втрата стабільно зменшується без різких стрибків, а загальна точність на тестових даних досягає 0,95 і перевищує результати інших експериментів. Донавчена модель саме з цією конфігурацією гаперпараметрів обрана як базова для подальшої інтеграції в інформаційну систему виявлення хвороб сільськогосподарських рослин.

Таким чином запропонована методика донавчання моделі ViT на доменному датасеті з 88 класами хвороб і станів рослин забезпечує рівень якості, співставний із сучасними state-of-the-art рішеннями для аграрного сектору. Отримані результати підтверджують, що використання архітектури Vision Transformer у поєднанні з коректною підготовкою даних та оптимізацією гіперпараметрів забезпечує високу точність автоматизованої діагностики хвороб рослин (понад 95-98%) та має значний потенціал для практичного впровадження у системи агромоніторингу.

4 РОЗРОБКА ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДІАГНОСТИКИ ХВОРОБ СІЛЬСЬКОГОСПОДАРСЬКИХ КУЛЬТУР

4.1 Моделювання та проєктування інформаційної системи

На етапі моделювання інформаційної системи виявлення хвороб сільськогосподатських рослин було сформовано основні сценарії використання та описано логіку взаємодії користувача з системою. Це дозволило чітко визначити функціональні вимоги до неї та структурувати її внутрішні процеси.

Інформаційна система орієнтована на фермерів та агрономів, які хочуть оперативно перевірити стан здоров’я рослини за фотографією. Користувач взаємодіє із системою через графічний інтерфейс вебзастосунку: завантажує зображення рослини, запускає аналіз, а у відповідь отримує результат діагностики, інформацію про рослину та опис виявленої хвороби з рекомендаціями щодо її лікування та профілактики. Для формалізації взаємодії користувача з системою розроблено діаграму прецедентів (рис. 4.1).

Усі прецеденти можна поділити на дві групи: ті, що ініціюються користувачем (інтерактивні), та ті, що виконуються системою автоматично (обчислювальні системні підпроцеси). Це підкреслює атвономність бекенду та мінімальне навантаження на користувача. Взаємодія користувача з системою передбачає описані нижче основні сценарії використання.

Use case 1: завантаження зображення листя рослини. Користувач (фермер чи агроном) відкриває інтерфейс вебзастосунку та завантажує зображення рослини (наприклад, листка кукурудзи, картоплі, рису чи пшениці). Система перевіряє коректність файлу (формат, розмір, наявність зображення), після чого передає файл на сервер. У випадку недопустимого формату система видає відповідне повідомлення.

Use case 2: попередня обробка зображення. Система виконує попередню обробку (масштабування, нормалізацію тощо) та передає зображення у модель Vision Transformer. У випадку помилки обробки користувач отримує повідомлення про помилку з рекомендацією повторити спробу.

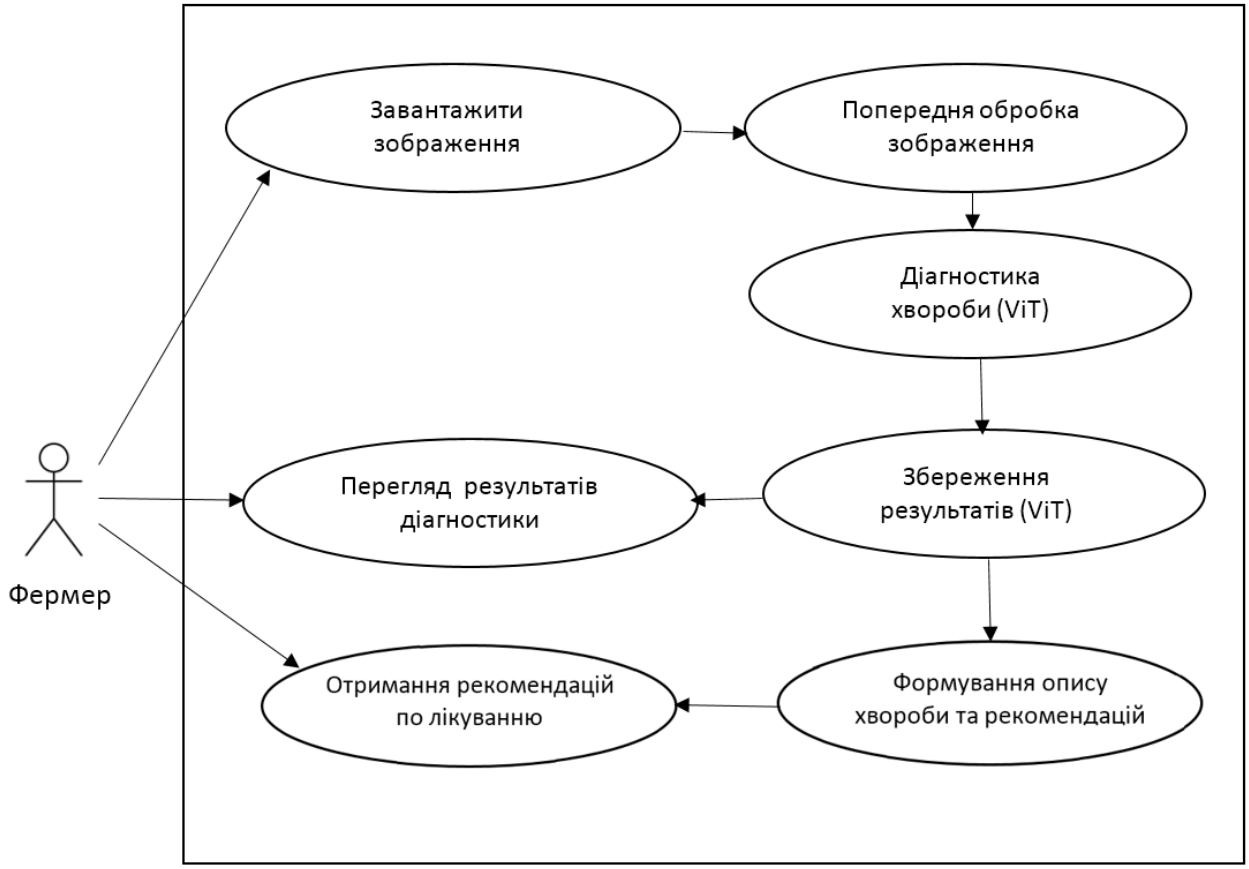

Рисунок 4.1 – Діаграма прецедентів інформаційної системи виявлення хвороб сільськогосподарських рослин

Use case 3: діагностика хвороби. Модель ViT виконує інференс – класифікує стан рослини та повертає в систему прогноз: «рослина здорова» або «виявлено хворобу» з зазначенням типу (наприклад, іржа, фітофтороз, плямистість листя). Отриманий результат система передає на збереження. У випадку помилки класифікації користувач отримує повідомлення про помилку з рекомендацією повторити спробу.

Use case 4: збереження результатів діагностики. Система формує JSON об’єкт з результатами, додає метадані (час обробки, ідентифікатор, рівень довіри) та зберігає результати діагностики про стан рослини.

Use case 5: формування опису хвороби та рекомендацій. Система отримує клас хвороби з JSON, після чого виконує запит до бази знань PostgreSQL. На основі отриманого опису симптомів та рекомендацій з профілактики і лікування виявленої хвороби система формує структурований текст та передає його у вебзастосунок.

Use case 6: перегляд результатів діагностики. На основі отриманого результату система формує відповідь, яка відображається користувачеві: інформація про виявлену хворобу (відсоток ймовірності) чи її відсутність, вихідне зображення.

Use case 7: отримання рекомендацій по лікуванню та профілактиці. У разі діагностування захворювання користувач має можливість детальніше переглянути інформацію про рослину та виявлене захворювання. Система відображає короткий опис культури, основні ознаки хвороби, можливі наслідки для врожаю, а також перелік рекомендованих заходів лікування та профілактики. У разі, якщо модель визначає, що рослина перебуває у здоровому стані, система повідомляє про відсутність ознак хвороб і може надати загальні рекомендації з профілактики (правильний полив, сівозміна, моніторинг симптомів тощо).

На етапі проєктування розроблено діаграму станів та переходів системи діагностики хвороб сількогосподарських культур (рис. 4.2). Діаграма відображає життєвий цикл обробки зображення у системі розпізнавання хвороб рослин: від моменту завантаження файлу до формування результатів діагностики. Вона показує послідовні стани, через які проходить система (завантаження, перевірка, попередня обробка, класифікація, формування опису), та переходи між ними, що залежать від результатів обробки та умов (коректність файлу, наявність помилки інференсу). Діаграма також демонструє можливість повернення до початкового стану у разі помилки та завершення процесу після відображення результатів і рекомендацій.

Рисунок 4.2 – Діаграма станів та переходів інформаційної системи

Під час моделювання інформаційної системи також було розроблено високорівневу архітектуру системи, яка була деталізована на етапі проєктування (рис. 4.3). Для роботи системи діагностики хвороб сільськогосподарських культур передбачено використання бази даних та серверного сховища зображень. Дані та метадані після обробки зберігаються в СУБД PostgreSQL.

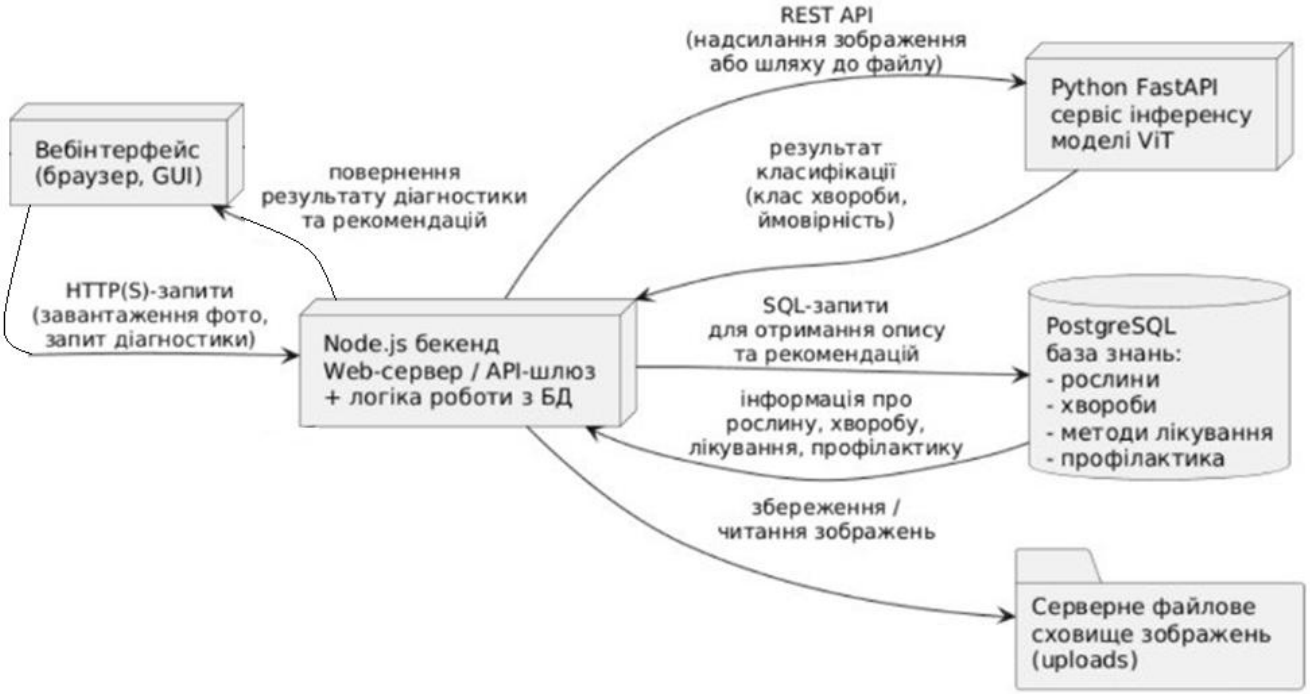

Рисунок 4.3 – Архітектура інформаційної системи діагностики хвороб

Таким чином, визначено основні сценарії використання, стани та переходи системи, а також ключові етапи обробки зображення та формування результату. Побудовані діаграма прецедентів, діаграма станів та переходів і архітектурна схема дозволяють наочно відобразити структуру і логіку роботи системи, що спрощує її подальшу реалізацію та тестування.

4.2 Інструментальні засоби розробки

Для реалізації інформаційної системи діагностики хвороб сільськогосподарських культур було розроблено вебзастосунок, що складається з фронтенд-частини (React), серверного API-шлюзу (Node.js + Express) та сервісу інференсу моделі Vision Transformer, реалізованого на Python (FastAPI). Вибір технологій був продиктований вимогами до продуктивності, зручності розробки, кросплатформенності та простоти інтеграції моделі ViT у вебзастосунок.

Використано наступні основні мови програмування та фреймворки – JavaScript / Node.js, Python, React та Vite. JavaScript / Node.js застосовується для: створення backend-шлюзу (backend/node/), обробки HTTP-запитів від фронтенду, завантаження файлів користувачем (через Multer), передачі файлів для класифікації у Python-сервіс. Переваги його використання є наступними: швидке реактивне API; велика кількість бібліотек; легка інтеграція з фронтендом без конвертації форматів.

Python 3.10 використовується для: завантаження навченої ViT-моделі; обробки зображень (Pillow); виконання інференсу (PyTorch); реалізації REST-ендпоінту в FastAPI. Переваги: природна сумісність із моделями машинного навчання; асинхронність FastAPI → висока швидкодія; простота реалізації логіки підготовки зображення.

Фронтенд розроблено за допомогою React та збірника Vite. Переваги: миттєве гаряче оновлення (HMR), оптимізована збірка, зручність створення компонентів та UI-логіки. Додатково використано TailwindCSS для швидкої стилізації інтерфейсу та створення адаптивних компонентів.

На backend (Node.js) використано наступні бібліотеки та програмні засоби:

– express – створення HTTP-роутів;
– multer – обробка завантаження файлів з фронтенду;
– axios – взаємодія з Python-сервісом;
– dotenv – конфігурація середовища;
– cors – дозвіл міждоменних запитів;
– form-data – формування multipart-запитів до FastAPI.

Ці бібліотеки дозволили забезпечити стабільне приймання файлів, безпечне збереження зображень, обмін даними між Node-шлюзом і Python-інференсом.

На Python-сервісі використано:

– fastapi – створення REST-сервісу інференсу;
– uvicorn – вебсервер ASGI-типу;
– transformers – завантаження Vision Transformer та пов’язаної конфігурації;

– torch – виконання інференсу на CPU/GPU;

– pillow – обробка та попередня підготовка зображень;

– python-multipart – підтримка завантаження файлів.

Цей стек технологій дозволяє завантажити модель один раз при старті серверу, обробляти запити надзвичайно швидко (порівняно з Flask), забезпечувати стабільну інференс-платформу.

На фронтенді (React) використано:

– vite – збірка проєкту;

– tailwindcss – стилізація;

– fetch API – відправлення файлів на Node API;

– React Hooks (useState) – управління станом форми.

Фронтенд відповідає за завантаження зображення користувачем, відображення прев'ю, надсилання файлу, візуалізацію TOP-K прогнозів.

Архітектурні принципи вибору стеку були наступними:

– розділення логіки на мікросервіси: Node.js – шлюз для роботи з файлами, Python – спеціалізований сервіс інференсу;

– переваги такого підходу: легке горизонтальне масштабування, незалежний деплой кожної частини, можливість заміни моделі без змін frontend-коду;

– зручність для користувача: React забезпечує швидкий та адаптивний інтерфейс, Vite мінімізує час завантаження та швидкість оновлення.

Структура проєкту у WebStorm – система має три рівні:

– Frontend: React + Vite застосунок з компонентом UploadForm;

– Backend API: Express-сервер зі шляхом /api/upload, який приймає файл, зберігає його у локальну папку, пересилає у Python-сервіс;

– Python Inference Service: FastAPI застосунок, який завантажує модель ViT один раз, виконує препроцесинг, повертає TOP-K прогнозів.

4.3 Реалізація серверної частини: Node.js API-шлюз та Python FastAPI сервіс інференсу

Серверна частина інформаційної системи складається з двох логічно відокремлених, але тісно пов’язаних компонентів: API-шлюзу, реалізованого на платформі Node.js, та сервісу інференсу Vision Transformer, побудованого на основі Python і фреймворку FastAPI. Такий поділ дозволяє одночасно забезпечити зручність роботи з HTTP-запитами, стабільну обробку файлів і ефективне виконання моделі ViT, яка потребує окремого високопродуктивного середовища.

Node.js-сервер виконує роль проміжної ланки між фронтендом і Python-сервісом, ізолюючи модель від прямого доступу користувачів та захищаючи інференс від потенційно некоректних запитів. Сервер ініціалізується у файлі server.js, де налаштовуються CORS-політики, JSON-парсер та публікується статична директорія для перегляду завантажених файлів.

Найважливішим компонентом Node-частини є маршрут /api/upload, реалізований у файлі upload.js. Саме він приймає multipart-форми з React-клієнта, виконує валідацію типу файлу та зберігає зображення у локальну директорію uploads. Робота із завантаженнями організована через бібліотеку Multer, яка дозволяє контролювати формат файлу, його ім’я та обмеження розміру. Після успішного завантаження Node-сервер формує новий multipart-запит через бібліотеку FormData, додає до нього шлях до зображення на файловій системі та перенаправляє цей запит до Python-сервісу за допомогою HTTP-клієнта axios.

Отримавши відповідь від FastAPI, Node.js-шлюз повертає клієнту JSON-структуру, яка містить як метадані збереженого файлу, так і перелік top-5 передбачень з їхніми ймовірностями. У разі помилки (некоректний файл, збій інференсу, проблеми з мережею) шлюз повертає відповідь із кодом 500 та текстом помилки, тим самим виконуючи роль контролера стабільності системи.

Python-сервіс розташований у директорії backend/python і працює як незалежний вебсервіс, що взаємодіє з Node тільки через HTTP. Основний файл app.py під час старту застосунку завантажує модель Vision Transformer, яка зберігається у папці model. Це передбачає одноразове зчитування ваг, конфігурації та препроцесора, що значно зменшує затримки під час обробки запитів, оскільки модель не завантажується щоразу повторно.

FastAPI надає ендпоінт /predict, який приймає файл зображення у форматі UploadFile. Після завантаження сервіс виконує послідовність обов'язкових перетворень, необхідних для стабільної роботи Vision Transformer: усуває помилки орієнтації EXIF, конвертує зображення у формат RGB, виконує центрований квадратний кроп і масштабує файл до розміру 224×224. Препроцесинг реалізовано вручну через PIL, оскільки це дає змогу гарантувати ідентичні параметри обробки як у вебсервері, так і під час навчання моделі.

Після препроцесингу зображення передається у процесор моделі HuggingFace, де відбувається нормалізація та перетворення на тензор. Модель ViT обчислює логіти, які далі проходять через softmax, після чого вибираються top-K результатів разом із відповідними мітками класів та ймовірностями. Додатково реалізовано механізм порогової фільтрації, який видаляє надто малоймовірні передбачення, що підвищує інформативність відповіді для користувача. Якщо всі значення виявляються нижчими порогу, сервіс повертає хоча б найбільш імовірний клас, щоб забезпечити стійкість роботи.

Завдяки використанню Uvicorn як ASGI-сервера FastAPI-сервіс здатний обробляти багатопотокові паралельні запити з низькою затримкою, що є критично важливим для вебзастосунку, який працює з файлами та моделями машинного навчання. У поєднанні з Node.js API-шлюзом це створює гнучку, ефективну та безпечну серверну архітектуру, де Node.js відповідає за логіку маршрутизації та взаємодії з клієнтом, а Python – за високоточний інференс Vision Transformer.

4.4 Реалізація клієнтської частини на React/Vite та взаємодія з REST API

Клієнтська частина вебзастосунку реалізована за допомогою фреймворку React та збірника Vite, що забезпечило можливість створення інтерактивного, швидкого та легкого у розгортанні інтерфейсу для кінцевого користувача. Основна логіка інтерфейсу розміщена у директорії frontend/src, де компоненти та стилі організовані таким чином, щоб забезпечити простоту підтримки та розширення функціональності застосунку.

React застосовується як основний інструмент побудови компонентів, що динамічно реагують на зміни стану. Завдяки архітектурі компонентів користувацький інтерфейс залишається стабільним та передбачуваним під час обробки файлів, комунікації з сервером і відображення результатів діагностики. Головним елементом інтерфейсу є компонент UploadForm.jsx, який відповідає за весь процес взаємодії користувача з системою: вибір зображення, попередній перегляд, надсилання файлу на сервер і візуалізацію результатів, отриманих від моделі Vision Transformer.

Після того як користувач обирає зображення, компонент виконує локальне формування попереднього перегляду за допомогою URL-об’єкта, що дозволяє користувачу миттєво побачити своє зображення до надсилання. Управління станом реалізовано через React Hooks (useState), що дозволяє зберігати інформацію про файл, стан завантаження, прев’ю та отримані відсотки ймовірностей кожного класу.

Процес надсилання зображення відбувається через стандартний API браузера fetch, який формує FormData і відправляє POST-запит на Node.js API-шлюз за маршрутом /api/upload. Адреса сервера задається через змінну середовища VITE_API_URL, що дозволяє легко перемикати середовище між локальним та серверним розгортанням без змін у вихідному коді. Після успішної відправки Node-сервер пересилає файл до Python-сервісу, а отримані результати повертає React-клієнту у форматі JSON. Компонент UploadForm інтерпретує цю відповідь та

зберігає її у вигляді локального стану, після чого інтерфейс автоматично оновлюється та відображає користувачу передбачені класи.

Особливу увагу приділено відображенню результатів класифікації. Кожен елемент передбачення містить назву класу та відсоток ймовірності. У компоненті реалізована функція pretty, що перетворює технічні мітки класів із моделі у читабельні назви. Візуалізація виконується через адаптивні елементи інтерфейсу TailwindCSS, де ймовірність кожного діагнозу доповнюється графічним індикатором у вигляді горизонтальної смуги-прогресу. Це не лише підвищує зручність сприйняття результатів, але й дозволяє користувачу швидко оцінити найбільш імовірні варіанти.

Головний компонент App.jsx організовує загальне оформлення сторінки, поміщаючи форму завантаження до стилізованої картки з використанням тіней, округлення та адаптивних відступів. Увесь фронтенд-застосунок стилізовано за допомогою TailwindCSS, підключеного через конфігурацію Vite. Це забезпечує швидку розробку інтерфейсу без необхідності ручного написання великої кількості CSS-правил.

Файл index.html містить мінімальну структуру HTML-документа з єдиним контейнером div#root, у який React вмонтовує свій віртуальний DOM. Початкове підключення компонентів здійснюється через main.jsx, де викликається ReactDOM для рендерингу застосунку. Така модульність дозволяє легко модифікувати або розширювати функціональність інтерфейсу без зміни логіки бекенду чи сервісу інференсу.

Ще однією важливою властивістю клієнтської частини є її повна незалежність від сервера. Завдяки використанню REST API фронтенд може працювати з будь-яким бекендом, який дотримується структури запиту /api/upload та повертає відповідний JSON-формат. Це дає можливість легко переносити застосунок на інші інфраструктури або підключати нові версії моделі Vision Transformer без необхідності змінювати клієнтський код.

Загалом клієнтська частина забезпечує інтуїтивно зрозумілу взаємодію з системою, простий доступ до функції діагностики, адаптивне відображення результатів та повну сумісність із серверною архітектурою. Використання React та Vite дозволило створити швидкий, легкий і сучасний вебінтерфейс, який ефективно поєднується з високоточною моделлю глибинного навчання на стороні сервера.

4.5 Програмна реалізація вебзастосунку

Програмна реалізація веб-застосунку для виявлення хвороб рослин побудована за клієнт-серверною архітектурою. Серверна частина виконує обробку зображень та взаємодію з моделлю машинного навчання, тоді як клієнтська частина відповідає за відображення інтерфейсу, завантаження зображень та виведення результатів діагностики.

Розробка виконувалася у середовищі WebStorm, яке забезпечує зручну роботу з JavaScript/Node.js-проєктами. Засоби реалізації серверної частини включають:

– кросплатформенне середовище виконання Node.js та фреймворк Express.js: для створення API-шлюзу та REST-маршрутів;

– бібліотеку Multer: для прийому файлів із форми (multipart/form-data) та збереження їх у локальну директорію uploads;

– зв’язка бібліотеки для HTTP-запитів Axios, вбудованого в браузер об’єкта FormData та класу httpClient: для передачі зображення до сервісу інференсу моделі;

– Python + FastAPI: ендпоінт інференсу;

– бібліотеки HuggingFace transformers і Pillow, фреймворк PyTorch: для доступу до моделей ViT, їх донавчання і препроцесингу зображень.

Код для запуску серверної частини міститься у файлі app.js (рис. 4.4). У файлі httpClient.js міститься модуль для HTTP-запитів до моделі.

Node.js API-шлюз: серверний маршрут для завантаження зображення та запиту до моделі – файл upload.js (додаток Б). Цей модуль обробляє POST-запит із форми, приймає зображення та пересилає його до сервісу моделі, здійснюючи

налаштування сховища файлів і логіки прийому зображення (рис. 4.5, рис. 4.6). Тут Node.js: приймає файл, проксує його до Python-сервісу, повертає клієнту інференс та текстову інформацію, яку сформував бекенд Python.

```
import express from "express";
import uploadRouter from "./routes/upload.js";

const app = express();
app.use(express.static("public"));
app.use("/upload", uploadRouter);

app.listen(3000, () => {
    console.log("Server is running on http://localhost:3000");
});
```

Рисунок 4.4 – Запуск серверної частини

```
const uploadDir = path.resolve(process.env.UPLOAD_DIR || "uploads");
fs.mkdirSync(uploadDir, { recursive: true });

const storage = multer.diskStorage({
  destination: (req, file, cb) => cb(null, uploadDir),
  filename: (req, file, cb) => cb(null, Date.now() + path.extname(file.originalname))
});

const upload = multer({ storage, });
```

Рисунок 4.5 – Налаштування сховища файлів і логіки прийому зображення

Клас FormData() використовується для формування запиту, а функція fs.createReadStream() — для передачі бінарного файлу. Програмний код для запуску API-шлюзу наведено на рисунку 4.7.

```
router.post( path: "/", upload.single("image"), async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> ,
                                                        res : Response<any, Record<...>> ) : Promise<any>  => {
  if (!req.file) return res.status( code: 400).json( body: { error: "No file uploaded" });

  const form : FormData  = new FormData();
  form.append( name: "file", fs.createReadStream(req.file.path), {
    filename: req.file.filename,
    contentType: req.file.mimetype,
  });

  const { data } = await http.post(process.env.PY_SERVICE_URL, form,  config: {
    headers: form.getHeaders(),
  });

  return res.json( body: {
    file: { name: req.file.filename, path: `/uploads/${req.file.filename}` },
    prediction: data.topk,
    plantInfo: data.plantInfo,
    diseaseInfo: data.diseaseInfo
  });
});
```

Рисунок 4.6 – Основний обробник POST /api/upload

```
const app = express();

app.use(cors());
app.use(express.json());
app.use("/uploads", express.static(staticDir));
app.use("/api/upload", uploadRoute);

app.listen(port, () : any | void  =>
  console.log(`Node API listening on http://localhost:${port}`)
);
```

Рисунок 4.7 – Запуск API-шлюзу

Python FastAPI-сервіс: інференс та звернення до бази знань. FastAPI-сервіс відповідає як за інференс, так і за формування розширеної відповіді для користувача. На старті вебзастосунку завантажуються модель і препроцесор (рис. 4.8).

```
MODEL_DIR = BASE_DIR / "model"
processor = AutoImageProcessor.from_pretrained(MODEL_DIR, use_fast=False)
model = ViTForImageClassification.from_pretrained(MODEL_DIR)
model.eval()
MODEL_IMG_SIZE = int(getattr(model.config, "image_size", 224))
```

Рисунок 4.8 – Завантаження моделі ViT і препроцесора

Попередня обробка зображення (EXIF-орієнтація, RGB, центрований кроп, масштабування) реалізовано із використанням функції prepare_pil_for_model() (рис. 4.9).

```
def prepare_pil_for_model(file_bytes: bytes) -> Image.Image:
    img = Image.open(io.BytesIO(file_bytes))
    img = ImageOps.exif_transpose(img)
    img = _rgba_to_rgb(img)
    img = _center_square_crop(img)
    if img.size != (MODEL_IMG_SIZE, MODEL_IMG_SIZE):
        img = img.resize((MODEL_IMG_SIZE, MODEL_IMG_SIZE), Image.BILINEAR)
    return img
```

Рисунок 4.9 – Функція для попередньої обробки зображення

Фрагмент бази знань для прив’язки класів моделі ViT до описів хвороб рослин та рекомендацій по лікуванню і профілактиці наведено на рисунку 4.10.

```
DISEASE_KB = {
    "Grape___black_rot": {
        "plant": { "name": "Виноград (Vitis vinifera)", ...
        "disease": {
            "name": "Black Rot (чорна гниль винограду)",
            "symptoms": [...],
            "treatment": [...]
        }
    },
    # інші культури та захворювання
}
```

Рисунок 4.10 – База знань для прив’язки класів моделі ViT до описів хвороб

Основний ендпоінт /predict виконує інференс, вибирає top-K класів, застосовує порогову фільтрацію та звертається до бази знань (рис. 4.11).

```python
@app.post("/predict")
async def predict(file: UploadFile = File(...)):
    image_bytes = await file.read()
    img = prepare_pil_for_model(image_bytes)

    inputs = processor(images=img, return_tensors="pt", do_resize=False)
    with torch.no_grad():
        probs = torch.softmax(model(**inputs).logits, dim=-1)[0]
        topk = torch.topk(probs, k=min(TOP_K, probs.shape[-1]))

    labels = [model.config.id2label[i.item()] for i in topk.indices]
    scores = [float(v.item()) for v in topk.values]
    filtered = [
        {"label": l, "score": s}
        for l, s in zip(labels, scores)
        if s >= THRESHOLD
    ] or [{"label": labels[0], "score": scores[0]}]

    main_label = filtered[0]["label"]
    kb_entry = DISEASE_KB.get(main_label, {})
    plant_info = kb_entry.get("plant")
    disease_info = kb_entry.get("disease")

    return {
        "topk": filtered,
        "plantInfo": plant_info,
        "diseaseInfo": disease_info,
    }
```

Рисунок 4.11 – Функція predict()

У файлі script.js реалізована логіка обробки введення та виведення результату на сторінку вебзастосунку.

У підсумку саме Python-сервіс перетворює зображення, обчислює ймовірності класів, підтягує з бази знань опис рослини та хвороби, повертає готові структури plantInfo та diseaseInfo до Node.js, а далі – до клієнта.

Клієнтська частина реалізована із використанням HTML/CSS/JS для інтерфейсу взаємодії з користувачем на основі JavaScript бібліотеки React та засобу збірки проєкту Vite. Це забезпечило швидке оновлення інтерфейсу, модульність та

зручну роботу зі станом вебзастосунку. Стилізація здійснюється через CSS-фреймворк TailwindCSS, що дало змогу інтерактивно формувати інтерфейс без надлишкових CSS-описів. Застосування механізму DOM-рендерінгу після інференсу забезпечує динамічне оновлення вебсторінки.

Головна роль фронтенду – відправити зображення на бекенд та відобразити інформацію, сформовану сервером, не виконуючи жодної додаткової логіки щодо хвороб чи рослин. Вся доменна інформація (опис культури, симптоми, лікування, рекомендації) надходить у JSON-відповіді.

Структура клієнтської частини проєкту містить такі ключові файли: index.html: базовий HTML-документ з контейнером для React; main.jsx: точка входу, що монтує React-застосунок; App.jsx (додаток В): компонент, який організовує макет сторінки та розміщення блоків; UploadForm.jsx (додаток Г): форма завантаження файлу та відображення результатів класифікації.

Компонент UploadForm реалізує вибір файлу з зображенням, попередній перегляд, запит на сервер та отримання даних з сервера (рис. 4.12).

```
const onSubmit = async (e) : Promise<void> => {
  e.preventDefault();

  const form : FormData = new FormData();
  form.append( name: "image", file);

  const resp : Response = await fetch( input: `${API}/api/upload`, init: {
    method: "POST",
    body: form
  });

  const data = await resp.json();
  setResult(data);
  onResult(data);
};
```

Рисунок 4.12 – Фрагмент логіки завантаження

Користувач отримує top-K передбачень, а також структуровані блоки plantInfo і diseaseInfo, які сервер відібрав із основі внутрішньої бази знань (рис. 4.13).

```
{result?.prediction?.map((p, i) => (
  <li key={i}>
    {pretty(p.label)} — {(p.score * 100).toFixed(2)}%
  </li>
))}
```

Рисунок 4.13 – Відображення передбачень

Компонент App.jsx є основним, отримує повну відповідь сервера та розміщує три блоки: інформацію про рослину (рис. 4.14); форму завантаження; інформацію про хворобу (рис. 4.15). Усі дані беруться зі структури backendResult:

```
const plantInfo = backendResult?.plantInfo;
const diseaseInfo = backendResult?.diseaseInfo;
```

```
{plantInfo && (
  <div>
    <h2>Рослина: {plantInfo.name}</h2>
    <p>{plantInfo.description}</p>

    {plantInfo.care?.map((item, i) => (
      <p key={i}>• {item}</p>
    ))}
  </div>
)}
```

Рисунок 4.14 – Відображення інформації про рослину

Функція renderResult() відповідає за оновлення HTML-контейнерів зліва (про рослину) і справа (про хворобу та лікування) (рис. 4.16). Наприклад, якщо модель

виявила хворобу Grape black rot, клієнтський код додає інформацію в два окремих блоки інтерфейсу. Це забезпечує динамічне відображення інформації без перезавантаження сторінки.

```
{diseaseInfo && (
  <div>
    <h2>Хвороба: {diseaseInfo.name}</h2>
    <p>{diseaseInfo.description}</p>

    {diseaseInfo.symptoms?.map((s, i) => (
      <p key={i}>• {s}</p>
    ))}

    {diseaseInfo.treatment?.map((t, i) => (
      <p key={i}>• {t}</p>
    ))}
  </div>
)}
```

Рисунок 4.15 – Відображення даних про хворобу

```
function renderResult(result) {
    document.getElementById("plantInfo").innerHTML = `
        <h3>Рослина: Виноград</h3>
        <p>Виноград — багаторічна ліана...</p>
    `;

    if (result.class === "Grape___black_rot") {
        document.getElementById("diseaseInfo").innerHTML = `
            <h3>Хвороба: Black Rot</h3>
            <p>Black rot — грибкове захворювання...</p>
            <h4>Методи лікування:</h4>
            <p>— Видалення уражених листків<br>— Обробка фунгіцидами...</p>
        `;
    }
}
```

Рисунок 4.16 – Функція renderResult()

Списки симптомів, опис рослини та методи лікування подаються однорівнево, без вкладених списків, що робить розмітку чистою та простою.

Принцип логіки на фронтенд: не визначає, яка рослина або хвороба на зображенні; не містить жодних описів хвороб у коді; не виконує обробку семантики результатів; не перевіряє назви класів. Увесь зміст формується бекендом. React лише: надсилає файл, отримує JSON, відображає текстові блоки. Завдяки цьому будь-яке оновлення моделі або бази знань не вимагає зміни клієнтської частини.

У параграфі описано структуру вебзастосунку, основні модулі серверної та клієнтської частин, принцип роботи маршруту завантаження та взаємодію з моделлю ViT. Наведені фрагменти коду демонструють ключові елементи реалізації: прийом файлів, відправку запиту до моделі, обробку результату та динамічне оновлення інтерфейсу користувача.

4.6 Інтерфейс користувача та сценарій роботи з системою

Користувацький інтерфейс розробленого вебзастосунку побудований із урахуванням простоти використання, інтуїтивності та мінімальної кількості дій, необхідних для отримання діагнозу за зображенням листка. Завдяки використанню React та TailwindCSS інтерфейс є адаптивним, чистим і сучасним, а всі взаємодії з системою виконуються у режимі реального часу без перезавантаження сторінки.

Після відкриття головної сторінки вебзастосунку користувач бачить екран із формою для завантаження зображення листя рослини. Інтерфейс не потребує додаткових налаштувань: користувач натискає кнопку *Завантажити файл* (рис. 4.17), обирає файл із зображенням хворої рослини з папки локального пристрою, після чого на сторінці з’являється зображення (рис. 4.18). Завдяки цьому у користувача є можливість переконатися, що обране зображення є коректним та добре кадрованим.

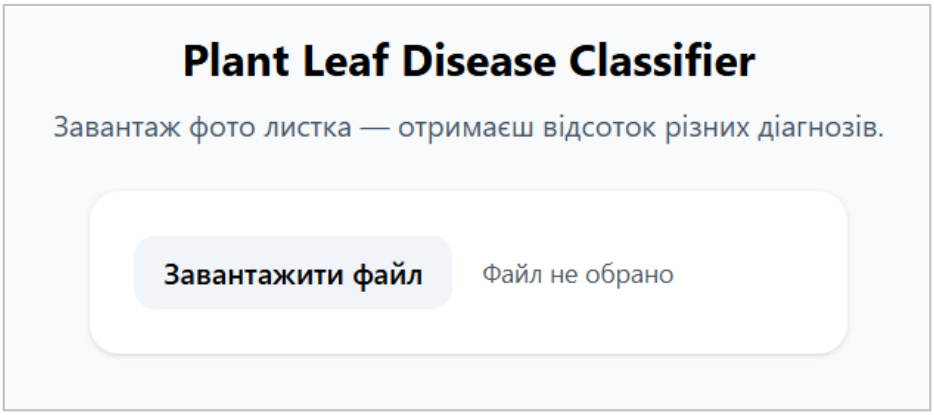

Рисунок 4.17 – Головна сторінка вебзастосунку перед початком роботи

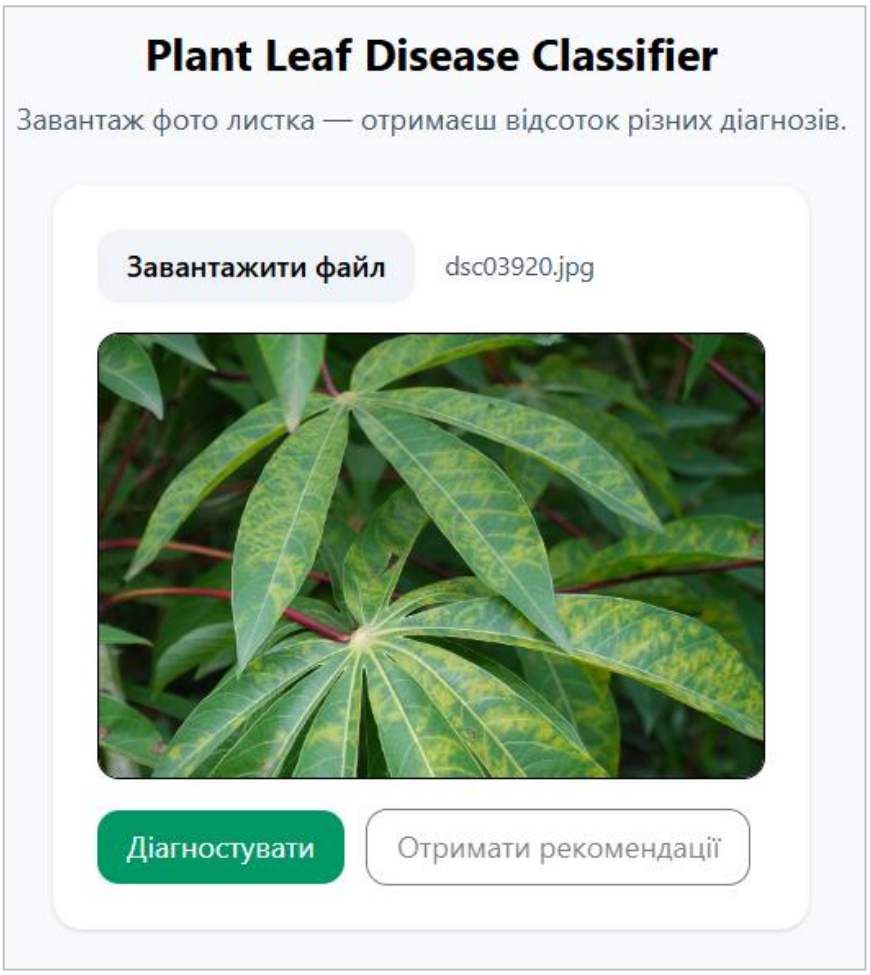

Рисунок 4.18 – Сторінка вебзастосунку після вибору файлу із зображенням листя рослини

Після натискання кнопки *Діагностувати* інтерфейс переходить у стан очікування. Кнопка стає неактивною, що запобігає повторним натисканням та

дублюванню запитів. Після отримання відповіді застосунок автоматично оновлює інтерфейс, додаючи до нього блок результатів. Кожен діагноз відображається у вигляді назви хвороби, числової оцінки ймовірності та горизонтального індикатора, що дає змогу швидко оцінити значення (рис. 4.19).

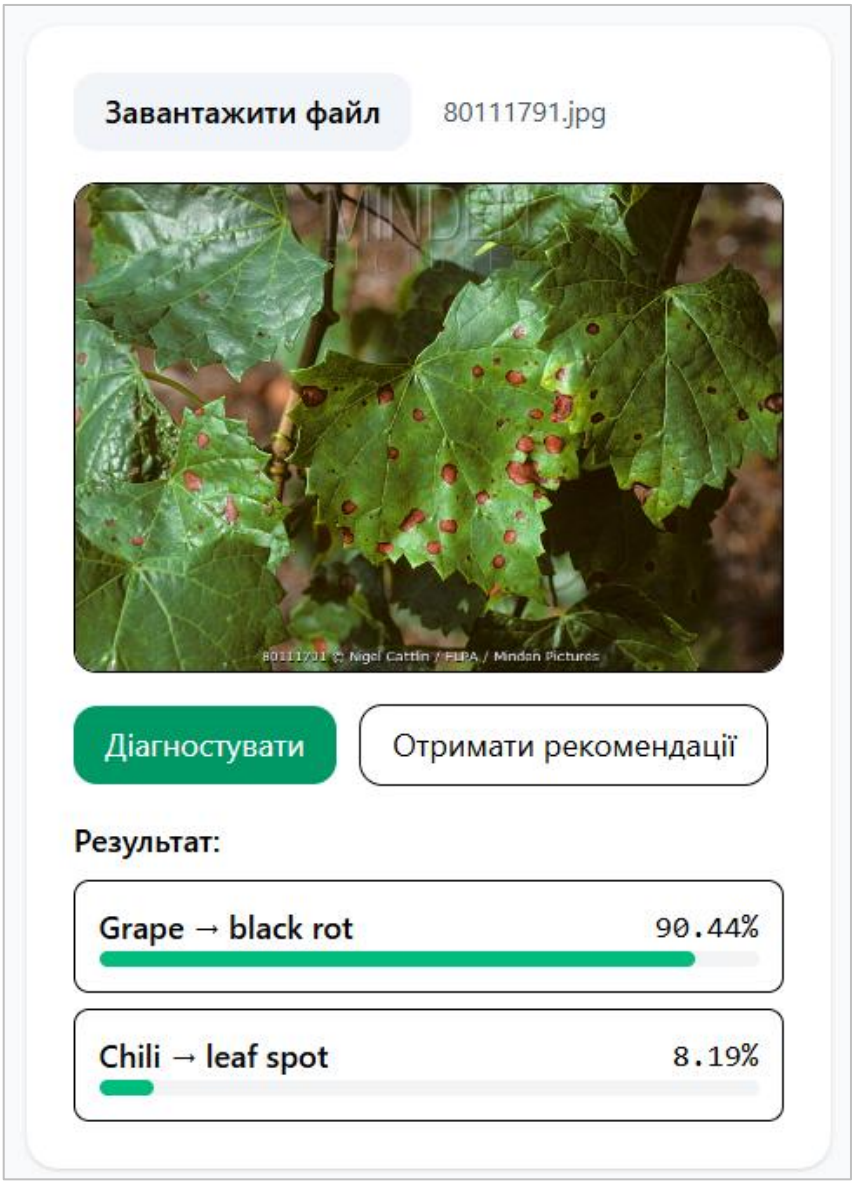

Рисунок 4.19 – Відображення прогнозів моделі ViT із ймовірностями

На рисунках 4.20 та 4.21 показано діагностування хвороб із використанням вебзастосунку таких як іржа у яблука та рання фітофторозна хвороба у картоплі.

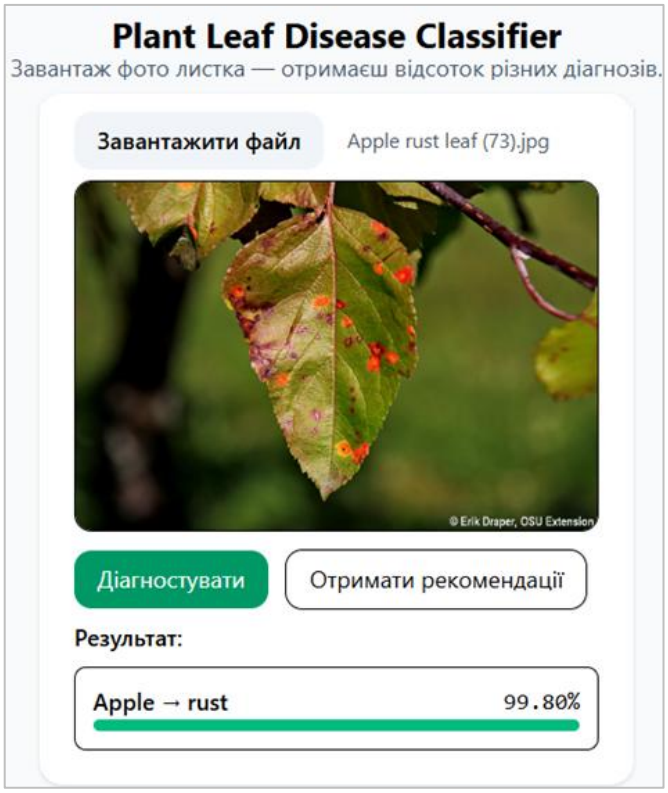

Рисунок 4.20 – Виявлення хвороби іржа у яблука

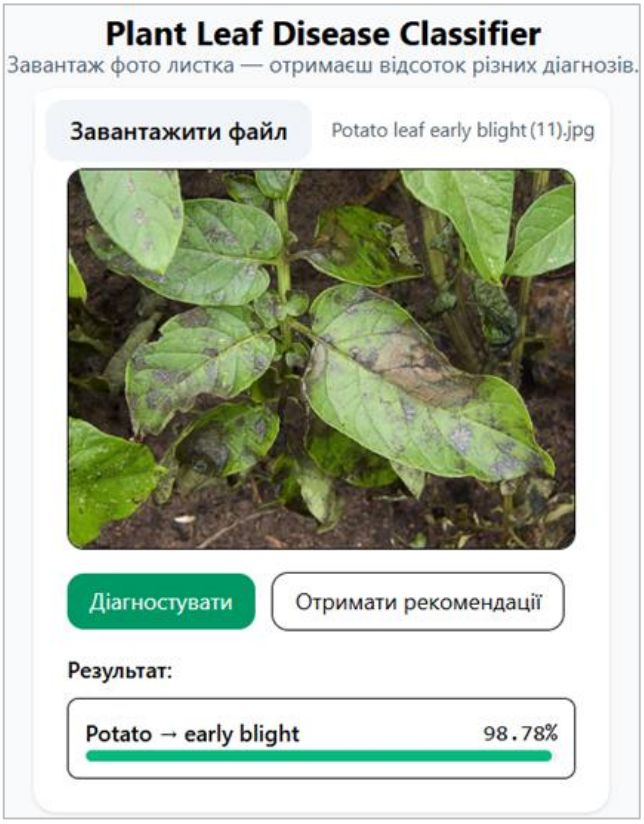

Рисунок 4.21 – Виявлення ранньої фітофторозної хвороби у картоплі

Коли модель виявляє захворювання, користувач може натиснути кнопку *Отримати рекомендації* і інтерфейс автоматично розширюється двома додатковими панелями: з інформацією про рослину та інформацією про діагностовану хворобу (рис. 4.22).

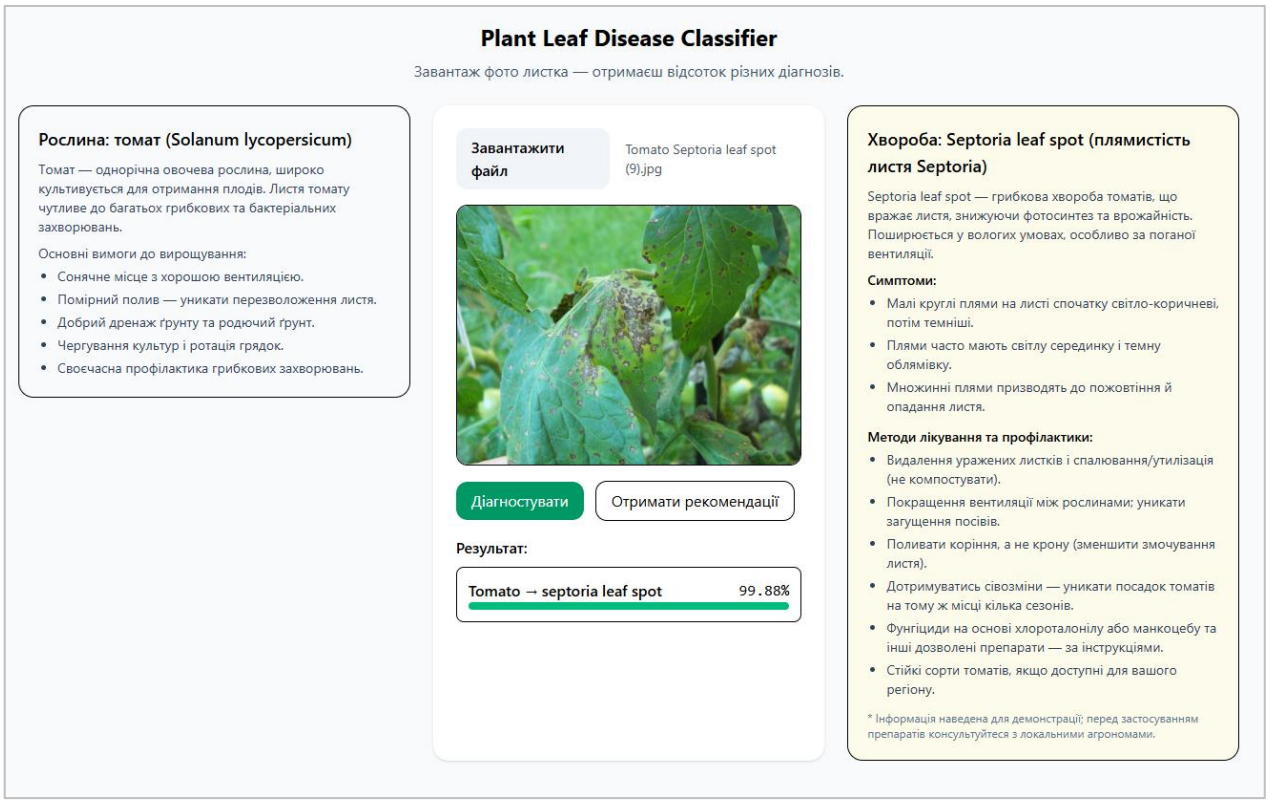

Рисунок 4.22 – Сторінка вебзастосунку з рекомендаціями по лікуванню та профілактиці виявленої хвороби

Панель з інформацією про рослину (ліворуч) містить загальну характеристику культури: назву рослини, короткий ботанічний опис, ключові особливості вирощування (рис. 4.23). Цей блок дозволяє користувачеві одразу співвіднести результат діагностики з конкретною рослиною, навіть якщо фото було зроблене в польових умовах.

Панель з інформацією про виявлену хворобу та методи її лікування (праворуч) містить: назву хвороби, короткий опис її природи, ознаки та симптоми, детальні рекомендації щодо лікуваннята поради з профілактики (рис. 4.24).

Наприклад, для Grape Black Rot: Black Rot – грибкове захворювання, що уражає листя, пагони та ягоди. Характеризується появою бурих плям із темною облямівкою. За відсутності контролю може призвести до повної втрати врожаю.

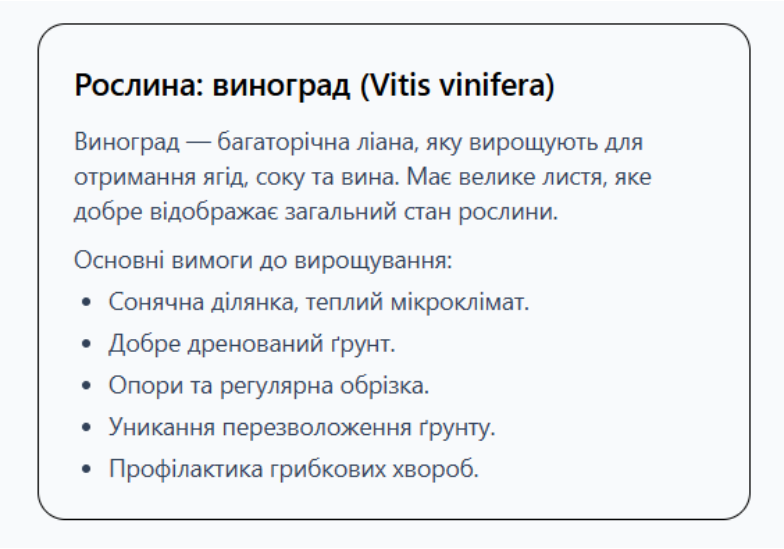

Рисунок 4.23 – Блок з інформацією про рослину у лівій панелі вебзастосунку

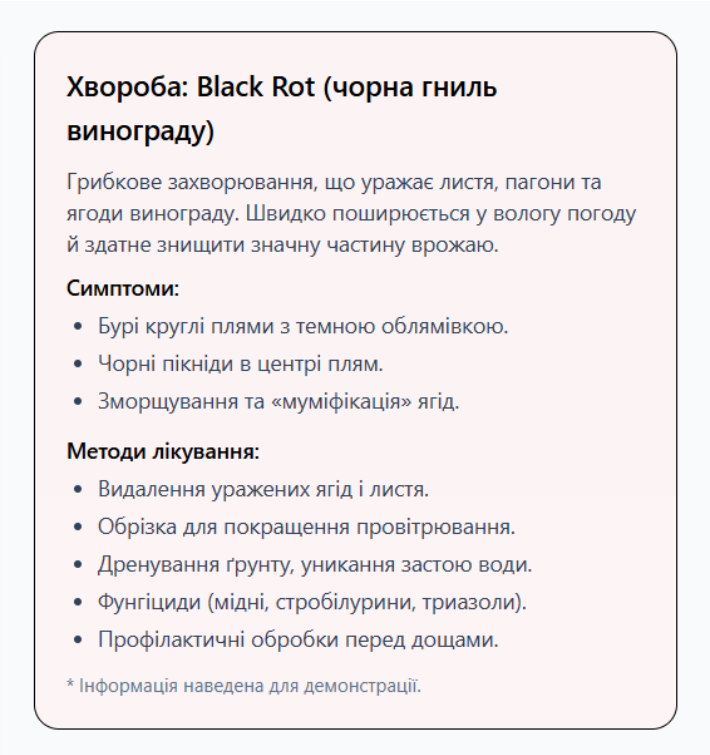

Рисунок 4.24 – Блок з інформацією про хворобу рослини у правій панелі вебзастосунку

Такий підхід створює зрозумілий і лаконічний сценарій роботи: обрати зображення → переглянути прев'ю → надіслати → отримати діагностику. Користувач отримує результат швидко, без зайвих переходів, і має можливість одразу повторити запит або завантажити файл із зображеннм іншої рослини.

4.7 Тестування та оцінка якості інформаційної системи

Тестування інформаційної системи виявлення хвороб рослин було спрямоване на перевірку коректності роботи моделі Vision Transformer, стабільності вебзастосунку, а також відповідності інтерфейсу функціональним вимогам. Оцінювання виконувалося за кількома напрямами: функціональне тестування, тестування продуктивності та зручності використання.

Функціональне тестування підтвердило, що всі ключові сценарії взаємодії користувача із системою працюють правильно та послідовно. Було перевірено такі основні функції:

– завантаження зображення листя рослини у форматах .jpg, .jpeg, .png;

– коректне відображення попереднього перегляду обраного зображення;

– передача файлу на сервер та обробка запиту маршрутом /upload;

– виконання інференсу моделлю та повернення результатів у форматі JSON;

– відображення отриманих прогнозів: назва хвороби, ймовірність, індикатор;

– активація та деактивація кнопки *Класифікувати* під час запиту;

– виведення інформаційних блоків про рослину та хворобу у разі виявлення хвороби;

– стабільність роботи системи при повторних класифікаціях та заміні зображення.

Тестування показало, що система коректно обробляє помилки, зокрема спробу відправити форму без зображення або завантаження файлів некоректного формату.

Тестування продуктивності включало перевірку часу обробки запиту та тестування навантаженням.

Перевірка час обробки запиту показала, що у середньому повний цикл «завантаження → передача → інференс → відповідь → відображення» становив:

– 0,3–0,5 с – обробка інтерфейсом;

– 0,8–1,2 с – інференс моделі (на сервері з GPU або оптимізованим CPU);

– 100–200 мс – мережевий обмін.

Сумарний час відповіді становив 1.3–1.8 секунди, що відповідає вимогам реального часу.

Тестування навантаженням. Система була протестована за середнього навантаження: 10 одночасних запитів – без зниження продуктивності; 50 одночасних запитів – демонструвалося часткове збільшення затримки інференсу, але помилки серверу були відсутні. Це підтверджує можливість масштабування застосунку до реальних фермерських або виробничих умов.

Зручність використання. Оцінка інтерфейсу виконувалася відповідно до критеріїв:

– простота виконання основної задачі (класифікації фотографії);

– інтуїтивність;

– кількість дій, необхідних для отримання результату;

– зрозумілість виведених діагностичних і довідкових даних.

Тестування показало, що користувач може отримати результат, виконавши чотири дії: 1) обрати фото; 2) переглянути прев’ю; 3) діагностувати; 4) переглянути сформовані рекомендації. Додаткові блоки з інформацією про рослину та рекомендації щодо лікування та профілактики роблять систему не лише діагностичною, а й консультаційною, що підвищує її практичну цінність.

Оцінка якості роботи інформаційної системи. На основі виконаних тестувань можна зробити такі узагальнення:

– система стабільно працює у всіх стандартних сценаріях;

– модель демонструє високу точність класифікації;

– функціонал додаткових інформаційних блоків реалізований коректно та повністю відповідає очікуваній логіці;

– інтерфейс забезпечує швидку взаємодію та не перевантажує користувача інформацією;

– час відповіді системи знаходиться в межах прийнятних значень;

– вебзастосунок придатний для практичного використання в аграрній сфері для експрес-діагностики хвороб рослин.

Висновки до розділу 4

У четвертому розділі було розглянуто моделювання, проєктування та програмну реалізацію вебзастосунку для автоматизованої діагностики хвороб сільськогосподарських культур на основі моделі Vision Transformer. Система демонструє комплексний підхід до інтеграції методів глибинного навчання у вебсередовище та забезпечує зручний інтерфейс для кінцевого користувача, поєднуючи високу точність моделі з доступністю у використанні.

Було обгрунтовано вибір технологічного стеку, який включає React та Vite для побудови сучасного та продуктивного фронтенду, Node.js та Express для створення API-шлюзу, а також Python, FastAPI та PyTorch для реалізації сервісу інференсу моделі. Така багаторівнева структура дозволила відокремити користувацький інтерфейс, бізнес-логіку та обчислювальний модуль, що виконує класифікацію зображень. Окреме розташування сервісів забезпечує масштабованість, гнучкість і можливість подальшого розширення системи, зокрема шляхом заміни або оновлення моделі без змін у клієнтському коді.

Архітектури системи складається з трьох логічних рівнів: клієнтського інтерфейсу, серверного API-шлюзу та сервісу інференсу. Node.js виконує роль проміжної ланки між React-клієнтом і Python-сервісом, відповідаючи за приймання файлів, валідацію, збереження зображень і формування запитів до моделі. FastAPI

забезпечує швидке формування прогнозів, використовуючи інтегровану ViT-модель та повний цикл попередньої обробки зображень.

Клієнтська частина вебзастосунок представлена як одноекранний інтерфейс, у якому користувач може завантажити зображення листя рослини, переглянути його прев'ю, отримати діагностику у вигляді списку найбільш імовірних хвороб та рекомендації по лікуванню. Завдяки TailwindCSS інтерфейс є адаптивним, мінімалістичним і зрозумілим. Представлений сценарій роботи дозволяє використовувати систему без спеціальних технічних знань, а візуалізація результатів – швидко оцінити ситуацію та приймати агротехнічні рішення.

Реалізований програмний комплекс повністю відповідає вимогам сучасних інформаційних систем у арній сфері, демонструє можливість ефективної інтеграції моделей ШІ у вебзастосунки, забезпечує швидку обробку даних та надає користувачеві зручний функціональний сервіс для діагностики хвороб рослин. Отримані результати створюють основу для подальшого розвитку системи, зокрема удосконалення архітектури, розширення класів хвороб, підключення мобільного застосунку або інтеграції з апаратними пристроями для автоматизованого моніторингу стану посівів.

ВИСНОВКИ

Проведене комплексне дослідження проблеми автоматизованої діагностики хвороб сільськогосподарських культур дозволило реалізувати повноцінну інформаційну систему, яка поєднує сучасні моделі глибинного навчання, інструменти комп'ютерного зору та технології ШІ. Отримані результати надають можливість сформувати цілісне уявлення про всі етапи – від теоретичного аналізу до програмної реалізації та впровадження моделі Vision Transformer у практичний вебзастосунок.

Досліджено предметну сферу та фундаментальні аспекти діагностики хвороб рослин. Проаналізовано традиційні підходи, сучасні методи комп'ютерного зору та глибинного навчання, а також тенденції останніх років, зокрема перехід до трансформерних архітектур у задачах класифікації зображень. Огляд наукових публікацій показав, що моделі Vision Transformer забезпечують високу точність, стійкість до шуму та здатність виявляти складні просторові патерни. Водночас було відзначено низку проблем, зокрема нестачу польових даних, дисбаланс наборів і обмеження продуктивності великих моделей. На основі аналізу було сформульовано мету, об'єкт, предмет та завдання дослідження.

Проаналізовано архітектури Vision Transformer, методи підготовки зображень, аугментації, алгоритмів навчання та обґрунтовано використання основних метрик оцінки якості моделі ViT. Виявленовнутрішню структуру ViT: механізм самоуваги, позиційне кодування, шар класифікації та також наведено обґрунтування використання цієї моделі для задач агровізуальної діагностики. Розроблено методику донавчання моделі на спеціалізованому наборі Plant Disease Classification Dataset та визначено оптимальні інструменти оцінювання результатів: Accuracy, Precision, Recall і F1-score.

Виконано серію експериментів із донавчання Vision Transformer за трьома різними конфігураціями гіперпараметрів. Досліджено вплив швидкості навчання голови моделі та бекбону, глибини заморожування шарів і способів боротьби з

дисбалансом даних. Зокрема, порівняно заморожене навчання, часткове розморожування та різні стратегії використання WeightedRandomSampler. Результати експериментів показали, що часткове донавчання з помірною швидкістю навчання та увімкненим семплером забезпечує найкращий баланс між точністю та стабільністю.

Здійснено моделювання, проєктування та програмну реалізацію вебзастосунку інфрмаційної системи, який дозволяє користувачеві завантажити фото листка та отримати діагноз у вигляді ймовірнісного розподілу по класах. Описано архітектуру, що складається з трьох незалежних компонентів: клієнтської частини (React + Vite), Node.js API-шлюзу та Python FastAPI сервісу інференсу. Кожен із компонентів виконує окрему роль: React забезпечує зручний інтерфейс, Node.js приймає та передає файли, а FastAPI відповідає за виконання моделі Vision Transformer. Така об’єктно-модульна структура дає можливість масштабувати систему, оновлювати модель без змін на клієнті та адаптувати сервіс під реальні потреби користувачів. Окремо проаналізовано сценарій роботи користувача та реалізацію інтерфейсу, який є максимально інтуїтивним і не потребує спеціальних знань.

Узагальнюючи результати роботи, можна стверджувати, що поставленої мети досягнуто повністю: створено інформаційну систему, здатну виконувати діагностику хвороб рослин із використанням Vision Transformer, проведено порівняння різних стратегій навчання, сформовано оптимальний набір гіперпараметрів та реалізовано вебзастосунок, який може застосовуватися в агросекторі. Система є масштабованою, розширюваною та придатною до практичного використання у фермерських господарствах, дослідницьких установах або як освітній інструмент.

Отримані результати відкривають перспективи подальшого вдосконалення: розширення набору хвороб та культур, інтеграція моделей локалізації (обведення уражених ділянок), перехід до мобільних застосунків, використання lightweight-моделей для роботи на пристроях IoT або дронах, а також збільшення кількості

польових даних для підвищення генералізації системи. Таким чином, робота не лише демонструє можливість ефективного застосування Vision Transformer у аграрній сфері, але й закладає фундамент для подальших досліджень у напрямі розумних аграрних систем.

Поставлені завдання виконано повністю, однак швижкий розвиток технологій комп’ютерного зору та ШІ обумовлює необхідність регулярного вдосконалення функціоналу системи у відповідності з новими підходами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Dosovitskiy A. Beyer L., Kolesnikov A., Weissenborn D., Zhai X., Unterthiner T. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020. ICLR. URL: https://arxiv.org/abs/2010.11929 (дата звернення 7.09.2025).

2. Li, Y., Zhang, J., Liu, D., Wang, C. PMVT: A Lightweight Vision Transformer for Plant Disease Detection. *Frontiers in Plant Science*. 2023. Vol. 14. URL: https://doi.org/10.3389/fpls.2023.1256773 (дата звернення 10.09.2025).

3. Thakur A., Singh P., Jain R. Vision Transformer Meets Convolutional Neural Network for Plant Disease Classification. *Ecological Informatics*. 2023. Vol 77. DOI: 10.1016/j.ecoinf.2023.102245 (дата звернення 11.09.2025).

4. Ullah A., Ahmad M., Zafar A., Habib S. AppViT: Hybrid Conv + ViT for Apple Leaf Disease Classification. *Springer Nature Computer Science*, 5(3). 2024. Vol. 5(3). DOI: https://doi.org/10.1007/s42979-024-02548-3 (дата звернення 7.10.2025).

5. Zhao H., Xu R., Tang Y., Wang X. MAE-fViT: Fine-tuned Masked Autoencoder Vision Transformer for Potato Leaf Disease Classification. *ResearchGate*. 2024. URL: https://www.researchgate.net/publication/380050145_MAE-fViT (дата звернення 13.10.2025).

6. Keetawan P., Luan Venancio L. (2024). CLIP-ViT Models for Zero-Shot Plant Disease Recognition in Real-World Environments. Cornell University, arXiv preprint. 2024. URL: https://doi.org/10.48550/arXiv.2403.01125 (дата звернення 3.10.2025).

7. Zhao Z., et al. Benchmarking In-the-Wild Multimodal Plant Disease Recognition Using Vision Transformers. *arXiv preprint*. 2024. URL: https://arxiv.org/abs/2404.06421 (дата звернення 1.10.2025).

8. PlantVillage Dataset. Kaggle Open Dataset for Plant Disease Classification. 2022. URL: https://www.kaggle.com/datasets/emmarex/plantdisease (дата звернення 7.09.2025).

9. Plant Disease Classification Merged Dataset. *Kaggle Datasets*. 2024. URL: https://www.kaggle.com/datasets/vipoooool/plant-disease-dataset (дата звернення 3.11.2025).

10. Vision Transformer (ViT) Documentation and Pretrained Models. Hugging Face: *вебсайт*. 2025. URL: https://huggingface.co/docs/transformers/model_doc/vit (дата звернення 4.10.2025).

11. Touvron H., Cord M., Douze M., Massa F., Sablayrolles A., Jégou H. Training Data-Efficient Image Transformers & Distillation Through Attention (DeiT). *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2021. Vol. 45(3). DOI: https://doi.org/10.1109/TPAMI.2021.3106468 (дата звернення 27.10.2025).

12. Liu Z., Lin Y., Cao Y., Hu H. Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows. *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021. URL: https://doi.org/10.1109/ICCV48922.2021.01216 (дата звернення 1.11.2025).

13. Mehta S., Rastegari M. MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022. DOI: https://doi.org/10.1109/CVPR52688.2022.00452 (дата звернення 7.10.2025).

14. Howard A., Sandler M., et al. Searching for MobileNetV3. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019. URL: https://doi.org/10.1109/ICCV.2019.00020 (дата звернення 20.11.2025).

15. Pereira L., Souza F., Gonçalves W., et al. Deep Learning in Agriculture: A Comprehensive Review of Plant Disease Detection Techniques. *Computers and Electronics in Agriculture*. 2024. Vol. 220, 108936. DOI: https://doi.org/10.1016/j.compag.2024.108936 (дата звернення 21.11.2025).

16. Sharma R., Kaur H., Singh G. Application of Deep Learning Models in Smart Agriculture: Challenges and Future Trends. *Artificial Intelligence in Agriculture*. 2024. Vol. 10(2), P. 142–158. DOI: https://doi.org/10.1016/j.aiia.2024.02.002 (дата звернення 14.11.2025).

17. Ferentinos K.P. Deep Learning Models for Plant Disease Detection and Diagnosis. *Computers and Electronics in Agriculture*. 2018. Vol. 145, P. 311-318. DOI: https://doi.org/10.1016/j.compag.2018.01.009 (дата звернення 17.10.2025).

18. Muhammad S., Zafar A., Ahmad M. Comparative Evaluation of CNN, EfficientNet and Vision Transformer for Crop Disease Classification. *IEEE Access*. 2023. Vol. 11, P. 19256–19267. DOI: https://doi.org/10.1109/ACCESS.2023.3246205 (дата звернення 3.11.2025).

19. Plantix – Intelligent Plant Disease Detection App. *PEAT GmbH* (Berlin, Germany). 2025. URL: https://plantix.net/en/ (дата звернення 23.10.2025).

20. BASF Digital Farming – xarvio FIELD MANAGER. *BASF SE*. 2025. URL: https://www.xarvio.com/global/en/Field-Manager.html (дата звернення 7.11.2025).

21. Taranis SmartScout System. *Taranis Digital Agriculture Platform*. 2025. URL: https://www.taranis.ag/ (дата звернення 3.10.2025).

22. DroneDeploy. Agricultural Drone Mapping and Crop Health Analysis Platform. 2024. URL: https://www.dronedeploy.com/ (дата звернення 14.11.2025).

23. PyTorch Documentation: Vision Transformer Implementation. *PyTorch*: вебсайт. 2025. URL: https://pytorch.org/vision/stable/models/vision_transformer.html.

24. Transfer Learning with Vision Transformers (ViT). *Keras.io*: вебсайт. 2025. URL: https://keras.io/examples/vision/vision_transformer/ (дата звернення 3.10.2025).

25. Goodfellow I., Bengio Y., Courville A. Deep Learning. *MIT Press*. 2016. URL: https://www.deeplearningbook.org/ (дата звернення 2.11.2025).

26. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. URL: https://arxiv.org/abs/1512.03385 (дата звернення 3.11.2025).

27. Tan M., Le Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *Proceedings of the 36th International Conference on Machine Learning (ICML)*. 2019. URL: https://arxiv.org/abs/1905.11946 (дата звернення 19.10.2025).

28. Krizhevsky, A., Sutskever, I., Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks (AlexNet). *Advances in Neural Information Processing Systems (NeurIPS)*. 2012. URL: https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (дата звернення 17.10.2025).

29. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition (VGG). 2015. URL: https://arxiv.org/abs/1409.1556 (дата звернення 3.09.2025).

30. Ronneberger O., Fischer P., Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *MICCAI*. 2015. URL: https://arxiv.org/abs/1505.04597 (дата звернення 23.09.2025).

31. Selvaraju R. R., Cogswell M., Das A., et al. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *Proceedings of ICCV*. 2017. URL: https://arxiv.org/abs/1610.02391 (дата звернення 8.11.2025).

32. Carion N., Massa F., Synnaeve G., et al. End-to-End Object Detection with Transformers (DETR). *ECCV*. 2020. URL: https://arxiv.org/abs/2005.12872 (дата звернення 3.11.2025).

33. He K., Chen X., Xie S., Li Y., Dollár P., Girshick R. Masked Autoencoders Are Scalable Vision Learners (MAE). *Proceedings of CVPR*. 2022. URL: https://arxiv.org/abs/2111.06377 (дата звернення 9.10.2025).

34. Caron M., Touvron H., Misra I., et al. Emerging Properties in Self-Supervised Vision Transformers (DINO). 2021. URL: https://arxiv.org/abs/2104.14294 (дата звернення 4.11.2025).

35. Radford A., Kim J. W., Hallacy C., et al. (Learning Transferable Visual Models From Natural Language Supervision *CLIP*. 2021). URL: https://arxiv.org/abs/2103.00020 (дата звернення 19.10.2025).

36. Jain S., Wallace B. C. Attention is not Explanation. Proceedings of NAACL-HLT. 2019. URL: https://arxiv.org/abs/1902.10186 (дата звернення 3.11.2025).

37. Болюбаш Н. М. Інтелектуальний аналіз даних: навчальний посібник. Миколаїв: Вид-во ЧНУ ім. П. Могили, 2023. – 320 с. URL: https://dspace.chmnu.edu.ua/jspui/handle/123456789/1461 (дата звернення 17.10.2025).

38. Lin T.-Y., Goyal P., Girshick R., He K., Dollar P. Focal Loss for Dense Object Detection (RetinaNet). *ICCV*. 2017. URL: https://arxiv.org/abs/1708.02002 (дата звернення 3.11.2025).

39. Bochkovskiy A., Wang C.-Y., Liao H.-Y. M. YOLOv4: Optimal Speed and Accuracy of Object Detectors. 2020. URL: https://arxiv.org/abs/2004.10934 (дата звернення 27.11.2025).

40. Sandler M., Howard A., Zhu M., Zhmoginov A., Chen L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *CVPR*. 2018. URL: https://arxiv.org/abs/1801.04381 (дата звернення 3.11.2025).

41. Model Card for Smart Farming Disease Detection Transformer. *Hugging Face*: вебсайт. URL: https://huggingface.co/wambugu71/crop_leaf_diseases_vit?utm_source=chatgpt.com (дата звернення 29.09.2025).

42. Plant Disease Classification Merged Dataset. *Keggle*: вебсайт. URL: https://www.kaggle.com/datasets/alinedobrovsky/plant-disease-classification-merged-dataset/data (дата звернення 30.09.2025).

ДОДАТОК А

Характеристика датасет для донавчання базової моделі ViT

Джерело: Kaggle (https://www.kaggle.com/datasets/alinedobrovsky/plant-disease-classification-merged-dataset/data)
Обсяг: ≈ 83 603 зображення листків рослин
Формат: кольорові зображення (RGB)
Розмір вибірки: зменшено й збалансовано під потреби дослідження
Кількість класів: 88,
Кількість видів рослин: 14
Таблиця А.1 – Рослини, що містяться у наборі даних

Рослина	Класи/захворювання
Apple (яблуня)	Black rot, Rust, Scab, Healthy
Cassava (маніока)	Bacterial blight, Brown streak disease, Green mottle, Healthy, Mosaic disease
Cherry (вишня)	Healthy, Powdery mildew
Chili (перець чилі)	Leaf curl, Leaf spot, Whitefly, Yellowish, Healthy
Corn (кукурудза)	Common rust, Gray leaf spot, Northern leaf blight, Healthy
Cucumber (огірок)	Diseased, Healthy
Grape (виноград)	Black measles, Black rot, Leaf blight, Healthy
Pomegranate (гранат)	Diseased, Healthy
Potato (картопля)	Early blight, Late blight, Healthy
Soybean (соєві)	Caterpillar, Diabrotica speciosa, Healthy
Strawberry (полуниця)	Leaf scorch, Healthy
Sugarcane (цукрова тростина)	Red rot, Rust, Bacterial blight, Red stripe, Healthy
Tomato (помідор)	10 класів (Bacterial spot, Early/Late blight, Mosaic virus, Leaf mold, Septoria leaf spot, Target spot, Yellow leaf curl virus, Spider mites, Healthy)
Wheat (пшениця)	Brown rust, Yellow rust, Septoria, Healthy
Apple (яблуня)	Black rot, Rust, Scab, Healthy

ДОДАТОК Б

JavaScript-скрипт upload.js для визначення серверного маршруту зображення

```
dotenv.config();
const router = Router();

const uploadDir = path.resolve(process.env.UPLOAD_DIR || "uploads");
fs.mkdirSync(uploadDir, { recursive: true });

const storage = multer.diskStorage({
  destination: (req, file, cb) => cb(null, uploadDir),
  filename: (req, file, cb) => {
    const unique = Date.now() + "-" + Math.round(Math.random() * 1e9);
    const ext = path.extname(file.originalname || "");
    cb(null, `${unique}${ext}`);
  },
});
const fileFilter = (req, file, cb) => {
  const ok = /image\/(jpeg|png|jpg|webp)/.test(file.mimetype);
  cb(null, ok);
};
const upload = multer({
  storage,
  fileFilter,
  limits: { fileSize: 10 * 1024 * 1024 } // 10MB
});

router.post("/", upload.single("image"), async (req, res) => {
  try {
    if (!req.file) return res.status(400).json({ error: "No file uploaded" });

    const form = new FormData();
    form.append("file", fs.createReadStream(req.file.path), {
      filename: req.file.filename,
      contentType: req.file.mimetype,
    });

    const pyUrl = process.env.PY_SERVICE_URL;
    const { data } = await http.post(pyUrl, form, { headers: form.getHeaders() });

    return res.json({
      file: { name: req.file.filename, path: `/uploads/${req.file.filename}` },
      prediction: data.topk
    });
  } catch (e) {
    console.error(e);
    return res.status(500).json({ error: "Inference failed" });
  }
});

export default router;
```

ДОДАТОК В

JavaScript-скрипт App.jsx для формування макету сторінки та розміщення блоків

```
import UploadForm from "./components/UploadForm";
import { useState } from "react";

export default function App() {
  const [detectedDisease, setDetectedDisease] = useState(null);
  const [showRecommendations, setShowRecommendations] = useState(false);
  const [plantInfo, setPlantInfo] = useState(null);
  const [diseaseInfo, setDiseaseInfo] = useState(null);

  const handleRequestRecommendations = async () => {
    if (!detectedDisease) return;

    const res = await fetch(`/api/recommendations?disease=${detectedDisease}`);
    const data = await res.json();

    setPlantInfo(data.plant);
    setDiseaseInfo(data.disease);
    setShowRecommendations(true);
  };

  return (
    <div className="min-h-screen bg-slate-50 p-6">
      <h1 className="text-2xl font-bold mb-2 text-center">Plant Leaf Disease Classifier</h1>
      <p className="text-slate-600 mb-6 text-center">
        Завантаж фото листка — отримаєш відсоток різних діагнозів.
      </p>

      <div className="grid grid-cols-1 lg:grid-cols-3 gap-6 max-w-7xl mx-auto">

        {showRecommendations && plantInfo ? (
          <div className="border rounded-2xl bg-slate-50 p-5 h-fit">
            <h2 className="font-semibold text-lg mb-2">{plantInfo.title}</h2>
            <p className="text-sm text-slate-700 mb-2">{plantInfo.description}</p>

            {plantInfo.requirements && (
              <>
                <p className="text-sm text-slate-700 mb-1">Основні вимоги:</p>
                <ul className="list-disc pl-5 text-sm text-slate-700 space-y-1">
                  {plantInfo.requirements.map((r, i) => (
                    <li key={i}>{r}</li>
                  ))}
                </ul>
              </>
            )}
          </div>
        ) : (
          <div />
        )}

        <div className="bg-white rounded-2xl shadow p-6">
          <UploadForm
            onDiseaseDetected={(d) => {
```

```
            setDetectedDisease(d);
            setShowRecommendations(false);
            setPlantInfo(null);
            setDiseaseInfo(null);
          }}
          onRequestRecommendations={handleRequestRecommendations}
        />
      </div>

      {showRecommendations && diseaseInfo ? (
        <div className="border rounded-2xl bg-rose-50/70 p-5 h-fit">
          <h2 className="font-semibold text-lg mb-2">{diseaseInfo.title}</h2>
          <p className="text-sm text-slate-700 mb-2">{diseaseInfo.description}</p>

          {diseaseInfo.symptoms && (
            <>
              <h3 className="font-semibold mt-2 text-sm mb-1">Симптоми:</h3>
              <ul className="list-disc pl-5 text-sm text-slate-700 space-y-1">
                {diseaseInfo.symptoms.map((s, i) => (
                  <li key={i}>{s}</li>
                ))}
              </ul>
            </>
          )}

          {diseaseInfo.treatment && (
            <>
              <h3 className="font-semibold mt-3 text-sm mb-1">Методи лікування:</h3>
              <ul className="list-disc pl-5 text-sm text-slate-700 space-y-1">
                {diseaseInfo.treatment.map((t, i) => (
                  <li key={i}>{t}</li>
                ))}
              </ul>
            </>
          )}
        </div>
      ) : (
        <div />
      )}
    </div>
  </div>
 );
}
```

ДОДАТОК Г

JavaScript-скрипт UploadForm.jsx для завантаження файлу та відображення результатів класифікації

```
import { useState, useEffect } from "react";

export default function UploadForm({ onDiseaseDetected, onRequestRecommendations }) {
  const [file, setFile] = useState(null);
  const [preview, setPreview] = useState("");
  const [result, setResult] = useState(null);
  const [loading, setLoading] = useState(false);

  const API = import.meta.env.VITE_API_URL || "http://localhost:8000";

  const onFile = (e) => {
    const f = e.target.files?.[0] ?? null;
    setFile(f);
    setResult(null);
    onDiseaseDetected(null);

    if (f) {
      const url = URL.createObjectURL(f);
      setPreview(url);
    } else {
      setPreview("");
    }
  };

  useEffect(() => {
    return () => {
      if (preview) URL.revokeObjectURL(preview);
    };
  }, [preview]);

  const onSubmit = async (e) => {
    e.preventDefault();
    if (!file) return;

    setLoading(true);
    setResult(null);
    onDiseaseDetected(null);

    try {
      const form = new FormData();
      form.append("image", file);

      const resp = await fetch(`${API}/api/upload`, { method: "POST", body: form });
      if (!resp.ok) throw new Error(`Upload failed: ${resp.status}`);

      const data = await resp.json();
      setResult(data);

      const predicted = data?.prediction ?? [];

      const labelToLower = (label) => String(label || "").toLowerCase();
```

```
    const foundGrapeBlackRot = predicted.some((p) => {
      const l = labelToLower(p.label);
      return l.includes("grape") && l.includes("black") && l.includes("rot");
    });

    const foundTomatoSeptoria = predicted.some((p) => {
      const l = labelToLower(p.label);
      const hasTomato = l.includes("tomato");
      const hasSeptoria = l.includes("septoria");
      const hasSeptoriaPhrase = l.includes("septoria leaf") || (l.includes("septoria") && l.includes("spot"));
      return (hasTomato && hasSeptoria) || hasSeptoriaPhrase;
    });

    if (foundGrapeBlackRot) {
      onDiseaseDetected("grape_black_rot");
    } else if (foundTomatoSeptoria) {
      onDiseaseDetected("tomato_septoria");
    } else {
      onDiseaseDetected(null);
    }

  } catch (err) {
    console.error(err);
    alert("Помилка завантаження або обробки файлу.");
  } finally {
    setLoading(false);
  }
};

const pretty = (s) => String(s).replaceAll("__", " → ").replaceAll("_", " ");

return (
  <form onSubmit={onSubmit} className="space-y-4">
    <div className="flex items-center gap-4">
      <label
        htmlFor="file-input"
        className="cursor-pointer py-2 px-4 rounded-xl bg-slate-100 hover:bg-slate-200 font-medium inline-flex items-
center gap-2"
      >
        Завантажити файл
        <input
          id="file-input"
          type="file"
          accept="image/*"
          onChange={onFile}
          className="hidden"
        />
      </label>

      <span className="text-sm text-gray-600">
        {file ? file.name : "Файл не обрано"}
      </span>
    </div>

    {preview && (
      <img
        src={preview}
        alt="preview"
        className="w-full rounded-xl border object-contain max-h-80"
      />
```

```
    )}

    <div className="flex items-center gap-3">
     {file && (
      <button
       type="submit"
       disabled={loading}
       className="px-4 py-2 rounded-xl bg-emerald-600 text-white disabled:opacity-50"
      >
       {loading ? "Діагностую..." : "Діагностувати"}
      </button>
     )}

     <button
      type="button"
      onClick={onRequestRecommendations}
      disabled={!result || !result.prediction || result.prediction.length === 0}
      className="px-4 py-2 rounded-xl border bg-white hover:bg-slate-50 disabled:opacity-50"
      title={(!result || !result.prediction || result.prediction.length === 0)
       ? "Спочатку виконайте діагностику"
       : "Показати рекомендації"}
     >
      Отримати рекомендації
     </button>
    </div>

    {result?.prediction && (
     <div className="mt-4 space-y-2 text-left">
      <h3 className="font-semibold">Результат:</h3>
      <ul className="space-y-2">
       {result.prediction.map((p, i) => (
        <li key={i} className="rounded-lg border p-3">
         <div className="flex justify-between">
          <span className="font-medium">{pretty(p.label)}</span>
          <span className="font-mono">{(p.score * 100).toFixed(2)}%</span>
         </div>
         <div className="w-full bg-gray-100 h-2 rounded">
          <div
           className="h-2 rounded bg-emerald-500"
           style={{ width: `${p.score * 100}%` }}
          />
         </div>
        </li>
       ))}
      </ul>
     </div>
    )}
   </form>
  );
}
```