МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

СЛОТЮК Антон Іванович

Програмний модуль класифікації пошкоджень бетонних конструкцій на основі методів глибинного навчання / Software Module for Classification of Damage in Concrete Structures Based on Deep Learning Methods

спеціальність: 122 - Комп'ютерні науки
освітньо-професійна програма - Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-43
Слотюк А.І.

_____

Науковий керівник
к.т.н., доцент Д.І. Загородня

_____

Кваліфікаційну роботу допущено до захисту

«___»_____ 20___ р.

Завідувач кафедри

_____ Н. В. Дзюбановська

Тернопіль – 2026

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.В. Дзюбановська
«____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
СЛОТЮК Антон Іванович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Програмний модуль класифікації пошкоджень бетонних конструкцій на основі методів глибинного навчання / Software Module for Classification of Damage in Concrete Structures Based on Deep Learning Methods
керівник роботи Д. І. Загородня к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 01 грудня 2025 р. № 894.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:

- здійснити огляд існуючих систем виявлення та класифікації пошкоджень бетонних конструкцій;
- проаналізувати існуючі методи виявлення та класифікації пошкоджень бетонних конструкцій;
- розробити архітектурне, алгоритмічне та інформаційне забезпечення системи та створити загальну структуру модуля класифікації пошкоджень бетонних конструкцій;
- реалізувати програмне забезпечення, провести тестування програми.

5. Перелік графічного матеріалу в роботі:

- концептуальна схема аналізу зображення бетонної конструкції;
- структурна системи класифікації пошкоджень бетонних конструкцій;
- алгоритм динамічного блокового поділу (тайлінгу) зображень;
- функціональна схема перетворення даних у програмному модулі класифікації пошкоджень
- структурна схема перетворення даних у програмному модулі класифікації пошкоджень;
- результати тестування програми.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 01 грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи	до 01.01. 2026 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02. 2026 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 15.03.2026 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05. 2026 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2026 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2026 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2026 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2026 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06. 2026 р.	

Студент_____ А.І. Слотюк_____
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи_____ Д. І. Загородня
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмний модуль класифікації пошкоджень бетонних конструкцій на основі методів глибинного навчання» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом 74 сторінки і містить 14 ілюстрацій, 3 таблиці, 5 додатків та 30 використаних джерел.

Метою роботи є розроблення програмного модуля для автоматичної класифікації пошкоджень бетонних конструкцій на основі методів глибинного навчання.

Методи дослідження: використано методи системного аналізу для дослідження предметної області комп'ютерного зору, методи об'єктно-орієнтованого проєктування для побудови архітектури програмної системи, методи глибинного та мультимодального машинного навчання, а також методи експериментального оцінювання точності класифікації.

Розроблений програмний продукт може бути використаний для задач автоматичного моніторингу, виявлення та класифікації пошкоджень бетонних конструкцій.

Ключові слова: БЕТОННІ КОНСТРУКЦІЇ, КОМП'ЮТЕРНИЙ ЗІР, ГЛИБИННЕ НАВЧАННЯ, МУЛЬТИМОДАЛЬНІ МОДЕЛІ, КЛАСИФІКАЦІЯ ПОШКОДЖЕНЬ, АНАЛІЗ ЗОБРАЖЕНЬ.

ANNOTATION

The qualification work entitled “Software Module for Classification of Damage in Concrete Structures Based on Deep Learning Methods”, submitted for the Bachelor's degree in Specialty 122 Computer Science under the Educational Program Computer Science, comprises 74 pages and contains 14 figures, 3 tables, 5 appendices, and 30 references.

The aim of the qualification work is to develop a software module for the automatic classification of damage in concrete structures based on deep learning methods.

Research methods: system analysis methods were used to investigate the domain of computer vision; object-oriented design methods were applied to develop the architecture of the software system; deep learning and multimodal machine learning methods were employed, along with experimental methods for evaluating classification accuracy.

The developed software product can be used for automatic monitoring, detection, and classification of damage in concrete structures.

Keywords: CONCRETE STRUCTURES, COMPUTER VISION, DEEP LEARNING, MULTIMODAL MODELS, DAMAGE CLASSIFICATION, IMAGE ANALYSIS.

ЗМІСТ

ВСТУП

У сучасному будівництві бетонні та залізобетонні конструкції є одними з найбільш поширених елементів інфраструктури, що використовуються при зведенні житлових будівель, промислових об’єктів, мостів, транспортних споруд та інших інженерних систем. Надійність і довговічність таких конструкцій безпосередньо впливають на безпеку експлуатації будівель і споруд. Проте в процесі тривалої експлуатації, під впливом механічних навантажень, агресивного середовища, температурних змін та інших факторів у бетонних конструкціях можуть виникати різноманітні пошкодження, зокрема тріщини, відшарування, корозія арматури та інші дефекти.

Своєчасне виявлення та класифікація пошкоджень бетонних конструкцій є важливим етапом технічного обстеження будівель і споруд. Традиційні методи діагностики часто передбачають візуальний огляд фахівцями або використання спеціалізованого обладнання, що може потребувати значних часових та фінансових витрат. Крім того, результати такого аналізу можуть залежати від досвіду експерта, що іноді призводить до суб’єктивності оцінювання стану конструкцій.

У зв’язку з розвитком інформаційних технологій і комп’ютерного зору з’являється можливість автоматизувати процес виявлення та аналізу дефектів будівельних матеріалів. Особливого значення у цьому напрямі набувають методи машинного навчання та глибинного навчання, які дозволяють аналізувати великі масиви зображень і автоматично визначати характерні ознаки пошкоджень. Сучасні нейронні мережі, зокрема згорткові нейронні мережі та мультимодальні великі мовні моделі (Vision-Language Models, VLM), демонструють високу ефективність у задачах аналізу та класифікації зображень, що відкриває нові можливості для застосування таких технологій у галузі будівельної діагностики.

Використання програмних модулів на основі глибинного навчання дозволяє значно підвищити швидкість і точність аналізу стану бетонних конструкцій, зменшити вплив людського фактора та оптимізувати процес технічного обстеження об’єктів. Автоматизовані системи класифікації пошкоджень можуть застосовуватися під час моніторингу стану будівель, оцінювання їх технічного

стану, планування ремонтних робіт та забезпечення безпеки експлуатації інженерних споруд.

Метою роботи є розробка програмного модуля для автоматичної класифікації пошкоджень бетонних конструкцій на основі методів глибинного навчання.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати існуючі методи виявлення та класифікації пошкоджень бетонних конструкцій;
- дослідити сучасні методи глибинного та мультимодального машинного навчання, що використовуються для аналізу зображень;
- обрати та обґрунтувати архітектуру мультимодальної моделі для аналізу зображень бетонних конструкцій;
- розробити програмний модуль класифікації пошкоджень бетонних конструкцій;
- провести експериментальні дослідження ефективності запропонованого підходу та оцінити якість роботи моделі.

Об’єктом дослідження є процес виявлення та класифікації пошкоджень бетонних конструкцій на основі аналізу зображень.

Предметом дослідження є методи глибинного та мультимодального машинного навчання для автоматичної класифікації пошкоджень бетонних конструкцій.

Практичне значення роботи полягає у можливості використання розробленого програмного модуля для автоматизованого аналізу стану бетонних конструкцій під час технічного обстеження будівель, споруд та інфраструктурних об’єктів. Використання програмного модуля дозволяє скоротити час обробки та аналізу зображень і підвищити ефективність виявлення пошкоджень.

Отже, розробка програмного модуля класифікації пошкоджень бетонних конструкцій на основі методів глибинного навчання є актуальним завданням, спрямованим на підвищення ефективності та точності діагностики стану будівельних конструкцій.

Кваліцікаційна робота складається з трьох логічно послідовних розділів, у яких висвітлено аналіз предметної області та сучасних підходів до виявлення

дефектів бетонних конструкцій, проєктування та реалізацію програмного модуля, а також результати експериментальних досліджень і оцінювання ефективності розробленого рішення.

Апробацію результатів дослідження проведено на Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Інтелектуальні комп'ютерні системи та мережі», яка відбулася у 2026 році (м. Тернопіль, Україна). Копія публікації представлена в додатку А.

1 СТАН ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Бетонні та залізобетонні конструкції є одними з найпоширеніших елементів сучасного будівництва. Вони широко використовуються під час спорудження житлових, промислових, транспортних та інфраструктурних об’єктів. Популярність бетонних конструкцій зумовлена їх високою міцністю, довговічністю, відносною економічністю та здатністю витримувати значні статичні й динамічні навантаження.

Бетон є штучним кам’яним матеріалом, який утворюється внаслідок твердіння суміші цементу, води, заповнювачів (піску та щебеню) та, за потреби, спеціальних добавок. У залізобетонних конструкціях бетон поєднується зі сталевою арматурою, що забезпечує високу міцність на розтяг та згин. Завдяки такому поєднанню матеріал набуває комплексних механічних властивостей, які дозволяють використовувати його у складних інженерних спорудах.

Однак, незважаючи на значні переваги, бетонні конструкції можуть зазнавати різноманітних пошкоджень у процесі експлуатації. Поява дефектів зумовлена впливом зовнішніх факторів, помилками проєктування, порушенням технології будівництва або тривалою експлуатацією споруд. Виявлення та своєчасна діагностика таких пошкоджень є важливою умовою забезпечення безпечності та довговічності будівель і споруд [1].

Пошкодження бетонних конструкцій можуть виникати під впливом різноманітних факторів. До основних причин належать [2-5]:

- технологічні фактори. Помилки, допущені під час виготовлення або укладання бетонної суміші, можуть призвести до утворення дефектів у структурі матеріалу. Наприклад, недостатнє ущільнення бетонної суміші, неправильне співвідношення компонентів або використання неякісних матеріалів можуть спричинити появу порожнин, тріщин або зниження міцності бетону.
- конструктивні фактори. Неправильні розрахунки навантажень або недосконале конструктивне рішення можуть призвести до перевищення

допустимих напружень у конструкції. У результаті виникають тріщини або деформації, що поступово призводять до руйнування елементів споруди.

- експлуатаційні фактори. У процесі експлуатації будівлі піддаються різним механічним, температурним та вологісним впливам. Зміни температури можуть спричиняти розширення або стискання матеріалу, що призводить до появи тріщин. Крім того, тривале навантаження на конструкцію може викликати її поступову деформацію.
- кліматичні фактори. Вплив атмосферних опадів, перепадів температури, циклів замерзання та відтавання, а також агресивних хімічних середовищ може негативно впливати на структуру бетону. У таких умовах відбувається поступове руйнування поверхневого шару матеріалу.
- корозія арматури. Однією з найбільш поширених причин руйнування залізобетонних конструкцій є корозія сталевої арматури. Під впливом вологи та кисню метал поступово руйнується, що призводить до збільшення об’єму корозійних продуктів і виникнення внутрішніх напружень у бетоні.

У процесі експлуатації бетонних конструкцій можуть виникати різні типи дефектів. Найбільш поширеними є: тріщини, сколи та відшарування бетону, корозійні пошкодження, пористість, деформації конструкцій [6-8].

Тріщини є одним із найпоширеніших видів пошкоджень бетонних конструкцій. Вони можуть виникати внаслідок перевищення допустимих напружень, температурних деформацій, усадки бетону або корозії арматури. Тріщини можуть бути поверхневими або наскрізними, а також мати різну довжину, ширину та напрямок.

Сколи та відшарування бетону виникають внаслідок механічних ударів, впливу атмосферних факторів або корозії арматури. У результаті від поверхні конструкції відокремлюються окремі фрагменти бетону, що призводить до оголення арматури та зниження захисного шару.

Корозійні пошкодження арматури призводять до утворення тріщин і відшарування бетону. Цей процес супроводжується поступовим руйнуванням металевих елементів конструкції.

Пористість виникає під час виготовлення бетонної суміші або її укладання. Недостатнє ущільнення бетону призводить до утворення порожнин у матеріалі, що знижує його міцність.

Під впливом навантажень або нерівномірного осідання фундаменту можуть виникати деформації елементів будівлі. Такі пошкодження можуть супроводжуватися появою тріщин і зміною геометричних параметрів конструкції.

Для забезпечення безпечної експлуатації будівель і споруд необхідно регулярно проводити технічне обстеження бетонних конструкцій. Основними методами виявлення пошкоджень є: візуальний огляд, інструментальні методи контролю, неруйнівні методи контролю, цифрові методи аналізу.

Візуальний огляд – це найпростіший і найбільш поширений метод, який передбачає безпосередній огляд поверхні конструкції з метою виявлення тріщин, сколів та інших дефектів.

До інструментальних методів контролю належать ультразвукові дослідження, використання склерометрів, вимірювання ширини тріщин та інші способи визначення міцності бетону.

Неруйнівні методи контролю дозволяють оцінювати стан конструкцій без їх пошкодження. До них належать ультразвукові, магнітні, радіографічні та інші методи.

У сучасних умовах дедалі ширше застосовуються технології комп’ютерного зору (цифрові методи аналізу), які дозволяють автоматично аналізувати зображення бетонних поверхонь та виявляти дефекти.

Розвиток технологій штучного інтелекту та комп’ютерного зору відкриває нові можливості для автоматизованого аналізу стану будівельних конструкцій. Одним із перспективних напрямів є застосування методів глибинного навчання для класифікації пошкоджень бетонних поверхонь.

Глибинне навчання базується на використанні штучних нейронних мереж, які здатні самостійно виявляти закономірності у великих масивах даних. Зокрема, згорткові нейронні мережі та сучасні мультимодальні моделі типу VLM широко використовуються для аналізу зображень та розпізнавання об’єктів [9].

У задачі аналізу бетонних конструкцій такі алгоритми можуть використовуватися для автоматичного виявлення тріщин, сколів та інших дефектів на основі фотографій або відеозаписів. Для навчання моделі використовується набір зображень із позначеними дефектами, що дозволяє алгоритму визначати характерні ознаки пошкоджень.

Використання таких систем дозволяє значно підвищити ефективність технічних обстежень, оскільки автоматизований аналіз може обробляти великі обсяги даних за короткий час.

Розроблення програмного модуля класифікації пошкоджень бетонних конструкцій є важливим завданням сучасної інженерної діагностики. Такий модуль може використовуватися для аналізу фотографій, отриманих під час технічних обстежень або моніторингу будівель [10].

Автоматизована система аналізу дозволяє (рисунок 1.1):

- швидко виявляти дефекти бетонних поверхонь;
- класифікувати тип пошкодження;
- оцінювати ступінь небезпеки дефекту;
- формувати рекомендації щодо подальшої експлуатації конструкції.

На рисунку 1.1 наведено концептуальну схему автоматизованого моніторингу бетонних конструкцій. На першому етапі здійснюється отримання зображень бетонної поверхні за допомогою камери або безпілотного літального апарата та їх збереження у базі даних. На другому етапі виконується підготовка зображень до аналізу та їх обробка за допомогою мультимодальної моделі штучного інтелекту (VLM), яка здійснює класифікацію виявлених дефектів. На завершальному етапі формується структурований JSON-звіт із результатами аналізу, який може бути використаний для візуалізації інформації та підтримки прийняття рішень під час технічного обстеження будівельних конструкцій.

Використання подібних систем особливо актуальне для великих інфраструктурних об’єктів, таких як мости, тунелі, промислові споруди та багатоповерхові будівлі.

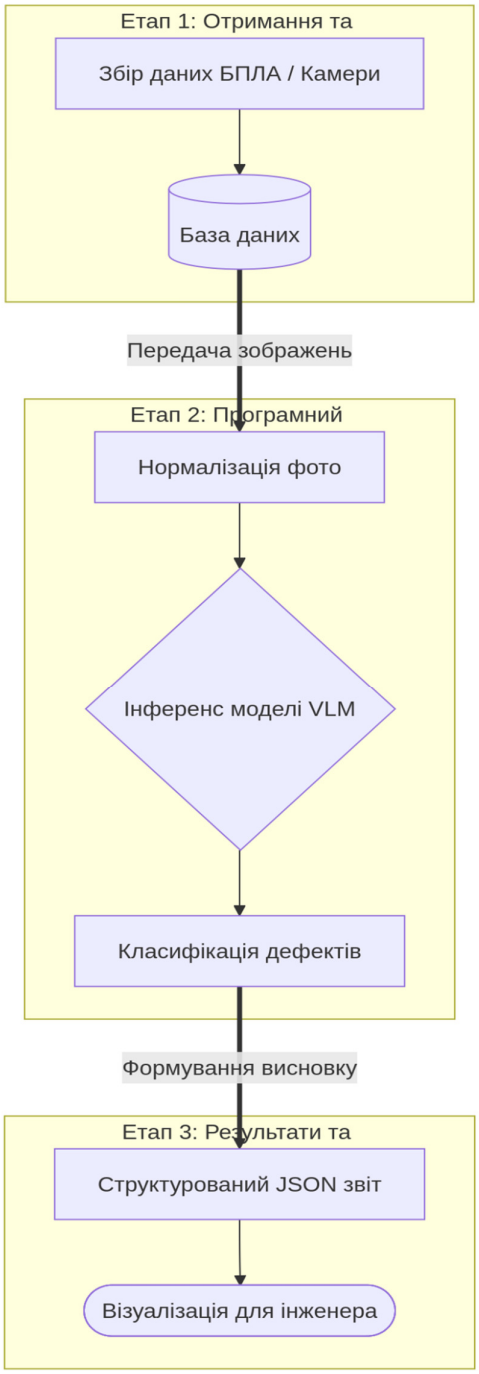

Рисунок 1.1 – Концептуальна схема автоматизованого моніторингу

Для забезпечення надійності будівель і споруд необхідно проводити регулярний контроль технічного стану конструкцій. Традиційні методи обстеження поступово доповнюються сучасними цифровими технологіями, зокрема методами комп’ютерного зору та глибинного навчання.

Застосування алгоритмів штучного інтелекту дозволяє автоматизувати процес виявлення та класифікації пошкоджень бетонних конструкцій, що значно підвищує ефективність інженерної діагностики. Саме тому розроблення програмного модуля класифікації пошкоджень на основі методів глибинного навчання є актуальним та перспективним напрямом досліджень у галузі будівельної інженерії та інформаційних технологій.

1.2 Огляд і аналіз існуючих рішень

Контроль технічного стану бетонних конструкцій є важливим етапом забезпечення надійності та безпеки будівель і інженерних споруд. У процесі експлуатації бетон піддається різноманітним механічним, фізичним та хімічним впливам, що можуть призводити до появи дефектів, зокрема тріщин, відшарувань, корозії арматури та руйнування поверхневого шару. Своєчасне виявлення таких пошкоджень дозволяє попередити подальше руйнування конструкцій та зменшити витрати на їх ремонт і обслуговування.

Традиційні методи діагностики стану бетонних конструкцій базуються на візуальному огляді або використанні спеціалізованих приладів неруйнівного контролю. Однак ці методи можуть бути трудомісткими, вимагати значних часових та фінансових витрат, а також залежати від кваліфікації фахівців. У зв’язку з цим все більшої актуальності набувають автоматизовані системи аналізу стану конструкцій, що базуються на методах комп’ютерного зору та глибинного навчання [1, 5, 11-13].

Сучасні алгоритми глибинного навчання дозволяють ефективно аналізувати зображення поверхні бетонних конструкцій та автоматично визначати наявність і тип дефектів. На сьогодні розроблено ряд програмних систем та алгоритмів, які використовуються для автоматичного виявлення та класифікації пошкоджень бетону. У даному розділі розглянуто та проаналізовано три найбільш поширені підходи до вирішення цієї задачі.

Одним із найбільш поширених підходів до автоматичного аналізу пошкоджень бетонних конструкцій є використання згорткових нейронних мереж

(Convolutional Neural Networks, CNN). Такі мережі здатні автоматично виділяти інформативні ознаки зображень і виконувати класифікацію без необхідності ручного формування ознак [14].

Система на основі CNN використовує навчальний набір зображень бетонних поверхонь, що містить приклади як пошкоджених, так і непошкоджених ділянок. У процесі навчання нейронна мережа аналізує зображення та формує внутрішні представлення ознак, які дозволяють відрізняти тріщини від інших елементів поверхні.

Зазвичай структура такої системи включає такі етапи:

1. Отримання зображення бетонної поверхні.
2. Попередня обробка зображення.
3. Подання зображення на вхід нейронної мережі.
4. Класифікація зображення.
5. Формування результатів аналізу.

У більшості досліджень використовуються набори даних, що містять тисячі зображень бетонних поверхонь з різними типами дефектів. Після навчання модель може визначати наявність тріщин з високою точністю.

Основними перевагами даного підходу є:

- висока точність класифікації;
- можливість автоматичного виділення ознак;
- ефективність при обробці великих обсягів даних.

Водночас існують і певні недоліки:

- необхідність великого обсягу навчальних даних;
- значні обчислювальні ресурси для навчання;
- зниження точності при зміні умов освітлення або якості зображення.

Іншим ефективним підходом є використання методів перенавчання нейронних мереж (transfer learning). У цьому випадку використовуються попередньо навчені моделі глибинного навчання, які адаптуються для задачі класифікації пошкоджень бетонних конструкцій.

До найбільш поширених архітектур належать:

- VGG16;

- ResNet;
- MobileNet;
- AlexNet.

Ці моделі були попередньо навчені на великих наборах зображень, наприклад ImageNet, і можуть бути використані для вирішення інших задач класифікації після додаткового навчання.

У системах аналізу пошкоджень бетону використання transfer learning дозволяє значно скоротити час навчання моделі та підвищити її точність. Після адаптації моделі до конкретного набору даних вона може визначати різні типи дефектів, зокрема:

- тріщини;
- відколи;
- відшарування;
- поверхневі руйнування.

Основними перевагами цього підходу є:

- зменшення потреби у великому наборі навчальних даних;
- висока точність класифікації;
- швидкість розробки системи.

Недоліками є:

- висока складність моделей;
- значне споживання пам’яті;
- потреба у графічних прискорювачах для ефективної роботи.

Ще одним сучасним підходом до аналізу стану бетонних конструкцій є використання алгоритмів детекції об’єктів у реальному часі. Одним із найпопулярніших алгоритмів такого типу є YOLO (You Only Look Once).

Алгоритм YOLO дозволяє одночасно визначати місцезнаходження дефектів на зображенні та класифікувати їх тип. На відміну від класичних методів класифікації, які аналізують усе зображення, YOLO виконує детекцію конкретних областей, у яких можуть знаходитися пошкодження [16].

Основні етапи роботи системи:

1. Отримання зображення бетонної поверхні.

2. Обробка зображення нейронною мережею YOLO.
3. Визначення координат дефектів.
4. Класифікація типу пошкодження.

Перевагами цього підходу є:

- можливість роботи в режимі реального часу;
- одночасна детекція і класифікація дефектів;
- висока швидкість обробки зображень.

Недоліки:

- потреба у значному обсязі навчальних даних;
- складність навчання моделі;
- можливі помилки при виявленні дуже дрібних дефектів.

Для більш наочного аналізу в таблиці 1.1 представлені результати порівняння за основними характеристиками.

Таблиця 1.1 – Порівняльна характеристика розглянутих систем

Характеристика	CNN-система	Transfer Learning система	YOLO-система
Основний підхід	Згорткові нейронні мережі	Перенавчання готових моделей	Детекція об’єктів
Тип задачі	Класифікація	Класифікація	Детекція та класифікація
Точність	Висока	Дуже висока	Висока
Швидкість роботи	Середня	Середня	Висока
Робота в реальному часі	Обмежена	Обмежена	Так
Вимоги до ресурсів	Середні	Високі	Високі
Необхідний обсяг даних	Великий	Середній	Великий
Основні переваги	Простота реалізації	Висока точність	Швидкість і локалізація дефектів
Основні недоліки	Потребує багато даних	Складність моделей	Складність навчання

Проведений аналіз показав, що сучасні системи автоматичного виявлення пошкоджень бетонних конструкцій переважно базуються на методах глибинного

навчання та комп’ютерного зору. Найбільш ефективними є підходи, що використовують згорткові нейронні мережі та їх модифікації.

Зокрема, системи на основі CNN забезпечують високу точність класифікації дефектів і можуть використовуватися для аналізу великих наборів зображень. Використання методів transfer learning дозволяє значно скоротити час навчання моделей і підвищити їх ефективність навіть при обмеженій кількості навчальних даних. Алгоритми детекції об’єктів, зокрема YOLO, дозволяють виконувати не лише класифікацію, але й локалізацію дефектів на зображенні та забезпечують можливість роботи системи в режимі реального часу [17].

Водночас існуючі рішення мають певні обмеження, зокрема залежність від якості навчальних даних, потребу у значних обчислювальних ресурсах та складність інтеграції деяких алгоритмів у прикладні системи. Тому актуальним є розроблення програмного модуля класифікації пошкоджень бетонних конструкцій, який забезпечуватиме високу точність розпізнавання дефектів, ефективну обробку зображень та можливість практичного застосування у системах моніторингу технічного стану будівель і споруд.

Перехід від традиційних моделей комп'ютерного зору, таких як ResNet, EfficientNet та архітектури детекції об'єктів YOLO або EfficientNet, до великих мультимодальних мовних моделей (VLM) став парадигмальним зрушенням у галузі комп'ютерного зору. Традиційні архітектури чудово справляються із задачами локалізації (Object Detection) та семантичної сегментації, проте їхній вихід обмежений жорстко заданим набором міток класів або обмежувальними рамками (bounding boxes). Для задач інженерного моніторингу залізобетонних конструкцій цього часто недостатньо, оскільки інспектору потрібна не лише констатація факту наявності тріщини, але й розуміння контексту: характеру її розповсюдження, наявності супутніх дефектів (наприклад, слідів іржі, що свідчить про корозію арматури) та загального стану поверхні.

Мультимодальні великі мовні моделі здатні долати так званий «семантичний розрив» (Semantic Gap) між піксельним представленням зображення та його високорівневим інженерним розумінням. Основою більшості сучасних відкритих

VLM є використання попередньо натренованих моделей типу CLIP (Contrastive Language-Image Pre-training) як візуальних екстракторів ознак.

Для обґрунтування вибору архітектури програмного модуля проведено порівняльний аналіз сучасних мультимодальних VLM-рішень (Таблиця 1.2). Оскільки модуль призначений для класифікації пошкоджень інфраструктури, критичними вимогами є конфіденційність даних (неможливість відправлення фото стратегічних об'єктів у сторонні API), точність та здатність до локального розгортання на Python.

Таблиця 1.2 — Порівняльний аналіз сучасних мультимодальних LLM для задач технічної інспекції

Назва моделі	Розробник	Відкритий код (Open-source)	Кіль-кість парамет-рів	Можливість локального розгортання	Потенційно придатні після спеціалізованого тонкого налаштування
GPT-4V / GPT-4o	OpenAI	Ні	Невідомо	Ні (тільки API)	Висока, але має ризики конфіденційності
Gemini 1.5 Pro	Google	Ні	Невідомо	Ні (тільки API)	Висока (гарне розуміння контексту)
LLaVA-1.5	Дослідницька спільнота	Так	7B / 13B	Так	Середня (обмеження роздільної здатності ViT)
Qwen-VL-Chat	Alibaba Cloud	Так	~10B	Так	Висока (підтримка високої роздільної здатності)
Moondream2	Спільнота	Так	1.8B	Так (навіть на мобільних пристроях)	Низька (недостатня глибина інженерних знань)

Аналіз таблиці 1.2 показує, що для нашого програмного модуля найбільш доцільним є використання відкритих архітектур класу LLaVA або Qwen-VL. Вони забезпечують оптимальний баланс між точністю аналізу та можливістю розгортання на власному обладнанні з використанням Python.

Проте існують специфічні виклики при застосуванні мультимодальних моделей VLM для аналізу зображень бетонних поверхонь. Основною проблемою є обмежена роздільна здатність вхідного вікна візуальних енкодерів. Більшість сучасних моделей працюють із зображеннями розміром 224×224 або 336×336 пікселів, що призводить до втрати дрібних деталей під час масштабування. У результаті мікротріщини, локальні відколи або початкові ознаки корозії можуть бути недостатньо помітними для моделі.

Для подолання цього обмеження у програмному модулі використовується підхід динамічного блокового поділу (Image Tiling) у поєднанні з мультимодальною моделлю VLM. Як показано на рисунку 1.2, початкове зображення високої роздільної здатності (4K) надходить до модуля препроцесингу, реалізованого засобами Python та OpenCV. На цьому етапі виконується розбиття зображення на набір окремих фрагментів (тайлів), які можуть частково перекриватися для збереження безперервності візуальних ознак на межах фрагментів.

Кожен сформований тайл незалежно передається до мультимодальної моделі VLM (Qwen-VL або LLaVA), яка здійснює локальний аналіз відповідної ділянки бетонної поверхні. Результатом обробки є набір локальних висновків, що містять інформацію про наявність або відсутність дефектів, а також їх тип. Наприклад, для одного фрагмента модель може визначити відсутність пошкоджень, для іншого — виявити ознаки мікротріщини, а для третього — зафіксувати сліди корозії арматури.

Отримані локальні результати передаються до модуля агрегації, який виконує їх узагальнення та формує єдиний підсумковий висновок щодо стану бетонної конструкції. На основі аналізу всіх фрагментів система генерує фінальний звіт, що містить інформацію про загальний технічний стан об'єкта та виявлені дефекти.

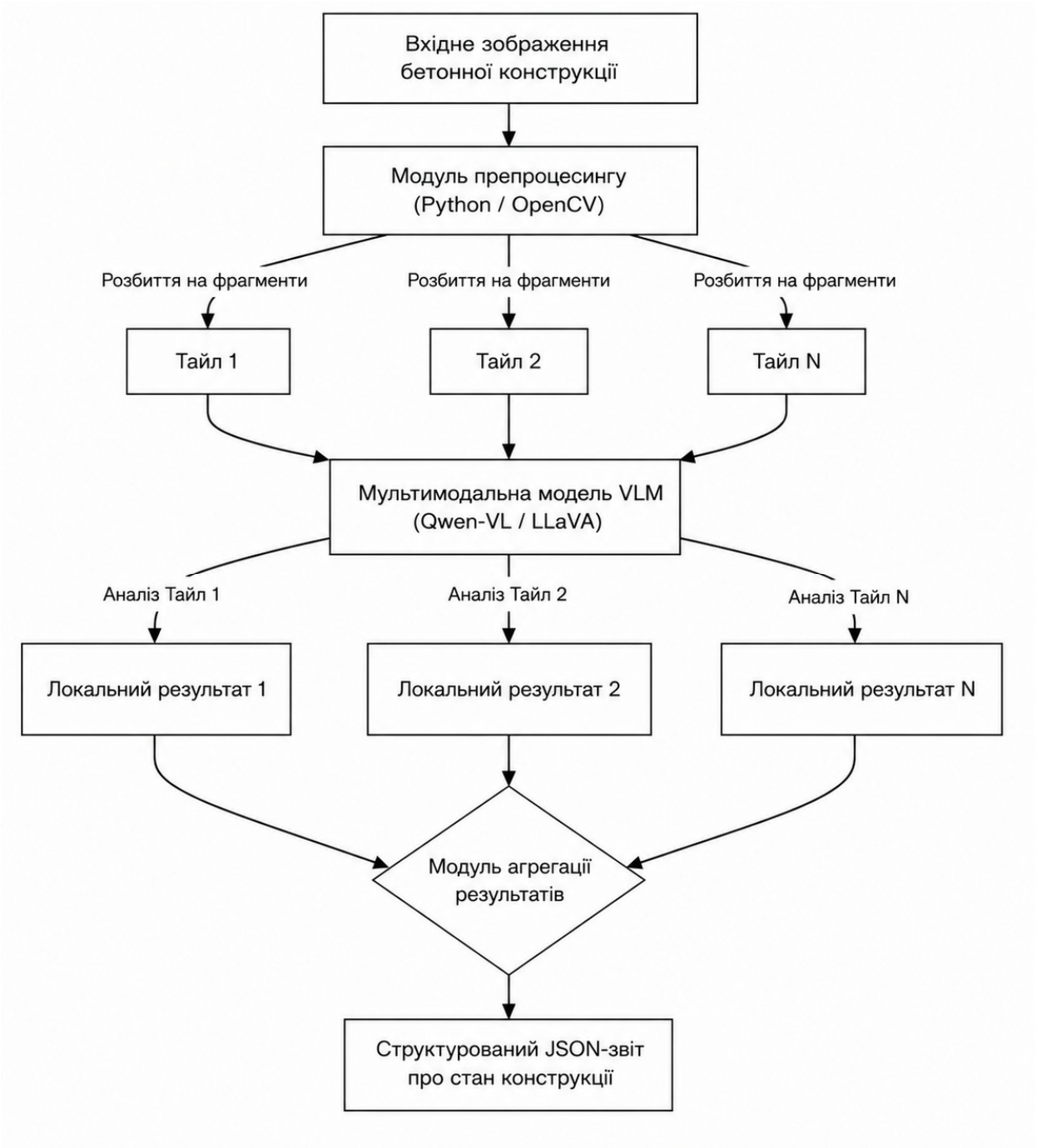

Рисунок 1.2 – Концептуальна схема аналізу зображення бетонної конструкції на основі VLM

Такий підхід дозволяє ефективно аналізувати великі зображення високої роздільної здатності без втрати дрібних деталей, підвищує ймовірність виявлення локальних пошкоджень і забезпечує формування обґрунтованого підсумкового висновку про стан конструкції.

1.3 Постановка задачі дослідження

Традиційні методи діагностики стану бетонних конструкцій здебільшого базуються на візуальному огляді фахівцями або застосуванні спеціалізованих приладів неруйнівного контролю. Проте такі методи можуть бути трудомісткими, потребувати значних витрат часу та ресурсів, а також залежати від досвіду та кваліфікації експертів. У зв'язку з цим актуальним є впровадження автоматизованих систем аналізу стану конструкцій, що дозволяють підвищити точність і швидкість виявлення дефектів.

Сучасні досягнення у галузі комп'ютерного зору та глибинного навчання створюють передумови для автоматизації процесу виявлення і класифікації пошкоджень на основі аналізу цифрових зображень поверхні бетонних конструкцій. Використання сучасних моделей глибинного навчання, зокрема згорткових нейронних мереж та мультимодальних моделей штучного інтелекту, дозволяє ефективно виділяти характерні ознаки дефектів та виконувати їх автоматичну класифікацію. Для практичного застосування таких підходів необхідно розробити програмне забезпечення, яке забезпечуватиме обробку зображень бетонних поверхонь та формування структурованих результатів аналізу.

Таким чином, задачею дослідження є розроблення програмного модуля автоматизованої класифікації пошкоджень бетонних конструкцій на основі методів глибинного навчання, який забезпечує аналіз цифрових зображень поверхні конструкцій та визначення типу виявлених дефектів. Для реалізації поставленої задачі передбачається використання мультимодальної моделі штучного інтелекту, здатної аналізувати візуальну інформацію та формувати структурований текстовий опис результатів класифікації.

Вхідними даними програмного модуля є цифрові зображення бетонних поверхонь, отримані за допомогою камер, мобільних пристроїв або безпілотних літальних апаратів. Вихідними даними є структуровані результати аналізу, що містять інформацію про наявність, тип та рівень впевненості виявлених дефектів. Для забезпечення інтеграції із зовнішніми інформаційними системами результати роботи модуля формуються у форматі JSON.

Для успішної програмної реалізації системи автоматизованої класифікації пошкоджень бетону необхідно чітко формалізувати функціональні можливості програмного модуля та визначити його експлуатаційні характеристики. В таблиці 1.3 наведено функціональні та нефункціональні вимоги до розроблюваної системи.

Таблиця 1.3 — Вимоги до програмного модуля

Тип вимоги	Опис вимоги	Критерій виконання
Функціональна	Прийом зображень через REST API	Підтримка форматів JPEG, PNG через HTTP POST
Функціональна	Мультимодальний аналіз дефектів	Виявлення тріщин, відколів, корозії за допомогою LLM
Функціональна	Форматування результатів	Повернення даних у валідному JSON форматі
Нефункціональна	Кросплатформність	Пакування модуля у Docker-контейнер
Нефункціональна	Апаратна сумісність	Підтримка CPU-інференсу на персональних комп'ютерах середнього рівня продуктивності.
Нефункціональна	Масштабованість	Можливість запуску у віртуалізованому середовищі

Як видно з таблиці 1.2, до програмного модуля висуваються як функціональні, так і нефункціональні вимоги. До функціональних вимог належать прийом зображень через REST API, автоматизований мультимодальний аналіз пошкоджень бетонних конструкцій та формування структурованих результатів у форматі JSON. Реалізація зазначених вимог забезпечує можливість інтеграції програмного модуля з іншими програмними системами та автоматизацію процесу технічного обстеження будівельних конструкцій.

Нефункціональні вимоги визначають експлуатаційні характеристики системи. Зокрема, програмний модуль повинен бути кросплатформним, підтримувати контейнеризацію за допомогою Docker, забезпечувати роботу на обчислювальних системах без обов’язкового використання графічних

прискорювачів та підтримувати розгортання у віртуалізованому середовищі. Виконання зазначених вимог забезпечує гнучкість використання програмного модуля, спрощує його розгортання та підвищує можливості практичного застосування.

Об’єктом дослідження є процес виявлення та класифікації пошкоджень бетонних конструкцій на основі аналізу цифрових зображень.

Предметом дослідження є методи глибинного та мультимодального машинного навчання для автоматичної класифікації пошкоджень бетонних конструкцій.

Практичне значення роботи полягає у можливості використання розробленого програмного модуля для автоматизованого аналізу стану бетонних конструкцій під час технічного обстеження будівель, споруд та інфраструктурних об’єктів. Використання програмного модуля дозволяє скоротити час аналізу зображень, підвищити ефективність виявлення дефектів та зменшити вплив людського фактора на результати оцінювання.

Таким чином, у результаті аналізу предметної області та формування вимог до програмного модуля було визначено основні функціональні можливості системи, архітектурні обмеження та критерії ефективності. Це створює основу для подальшого проєктування алгоритмічного та інформаційного забезпечення програмного модуля класифікації пошкоджень бетонних конструкцій.

2 АЛГОРИТМІЧНЕ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ ПРОГРАМНОГО МОДУЛЯ

2.1 Загальна структура проєктованої системи

Основу системи класифікації пошкоджень бетонних конструкцій становлять методи комп'ютерного зору та глибинного навчання, які дозволяють здійснювати автоматизовану обробку зображень та класифікацію пошкоджень з високою точністю. Запропонована система складається з декількох функціональних модулів, кожен з яких виконує окрему задачу в процесі аналізу зображень (рисунок 2.1). До основних компонентів системи належать:

- модуль отримання зображень;
- модуль попередньої обробки;
- модуль розбиття зображення на фрагменти (тайли);
- модуль мультимодального аналізу;
- модуль агрегації результатів;
- модуль формування та візуалізації результатів.

Взаємодія цих компонентів забезпечує повний цикл аналізу зображень — від отримання вихідних даних до формування результатів класифікації.

Першим етапом роботи системи є отримання зображень бетонних поверхонь. Джерелами даних є:

- цифрові камери;
- мобільні пристрої;
- безпілотні літальні апарати;
- існуючі бази даних зображень.

У практичних системах моніторингу технічного стану будівель зображення можуть отримуватися під час періодичних інспекцій або за допомогою автоматизованих систем спостереження.

На цьому етапі важливими факторами є:

- роздільна здатність зображення;
- умови освітлення;
- кут зйомки;

- відстань до об'єкта.

Якість отриманих зображень безпосередньо впливає на точність подальшого аналізу, тому на практиці використовуються рекомендації щодо стандартизації процесу фотографування бетонних поверхонь.

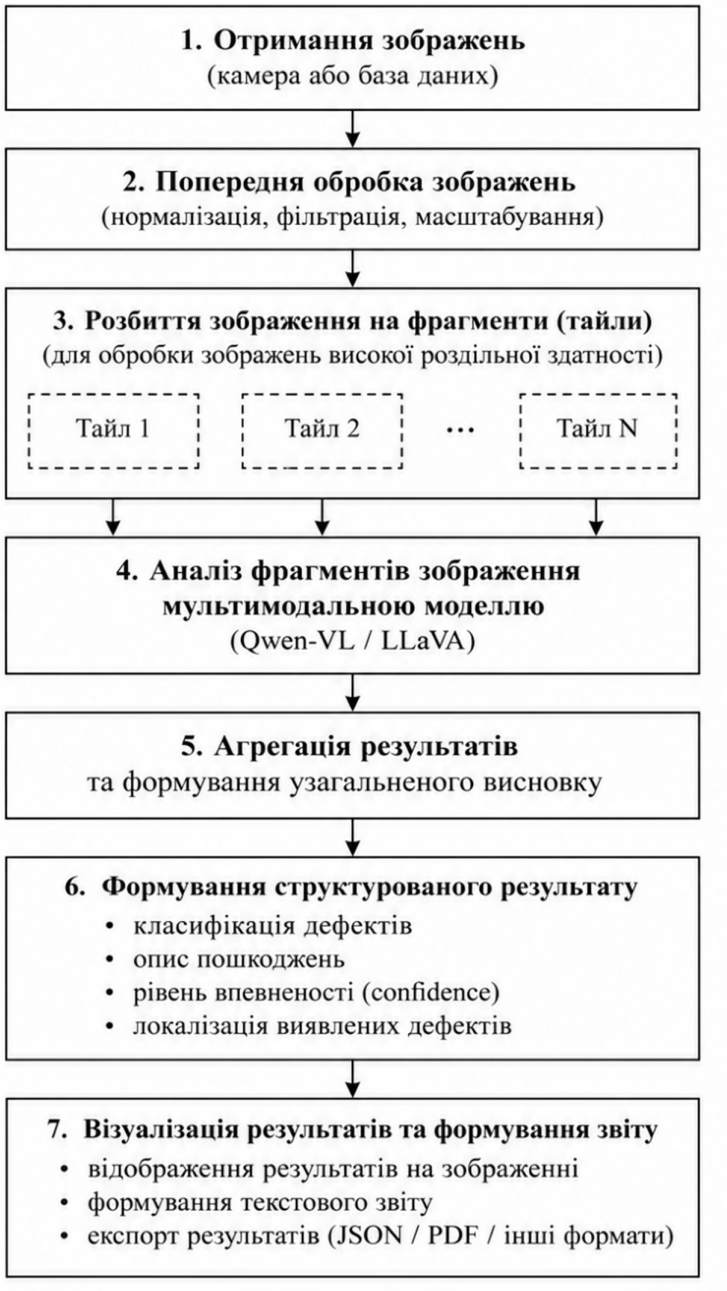

Рисунок 2.1 – Структура системи класифікації пошкоджень бетонних конструкцій

Після отримання зображення воно надходить до модуля попередньої обробки. Основною метою цього етапу є підготовка зображення до подальшого

аналізу. Попередня обробка включає масштабування зображення, нормалізацію піксельних значень, фільтрацію шумів та покращення контрасту.

Для забезпечення ефективної обробки фотографій високої роздільної здатності використовується підхід розбиття зображення на окремі фрагменти (Image Tiling). Кожний фрагмент аналізується незалежно, що дозволяє зберегти дрібні деталі поверхні та підвищити ймовірність виявлення локальних пошкоджень.

На наступному етапі кожний фрагмент передається до мультимодальної моделі глибинного навчання Qwen-VL або LLaVA. На відміну від класичних підходів, які передбачають окремі етапи сегментації, виділення ознак та класифікації, мультимодальні моделі виконують комплексний аналіз зображення в межах єдиного процесу інференсу. У процесі аналізу модель визначає наявність характерних ознак пошкоджень бетонної поверхні та формує структурований опис результатів.

До основних типів дефектів, які можуть бути виявлені системою, належать тріщини, відколи бетону, корозійні пошкодження арматури та інші дефекти поверхні. Результати аналізу окремих фрагментів надходять до модуля агрегації, де об'єднуються у єдиний підсумковий висновок щодо стану бетонної конструкції.

На основі узагальнених результатів формується структурований результат у форматі JSON, який містить інформацію про тип дефекту, його опис та рівень впевненості моделі. Завершальним етапом є візуалізація результатів та формування звіту, який може бути представлений користувачеві або переданий до зовнішніх інформаційних систем для подальшого аналізу та моніторингу.

Зображення приводиться до формату, придатного для подальшого аналізу мультимодальною моделлю. На етапі попередньої обробки виконується нормалізація піксельних значень та, за необхідності, масштабування зображення. Для обробки фотографій високої роздільної здатності використовується підхід розбиття зображення на окремі фрагменти розміром 336×336 пікселів, що дозволяє зберегти дрібні деталі поверхні та підвищити ефективність виявлення локальних дефектів.

Зображення може містити шум, спричинений несприятливими умовами освітлення, особливостями роботи сенсора камери або втратами якості під час цифрового стискання. Для зменшення впливу шумів можуть застосовуватися медіанний або гаусівський фільтри.

Оскільки тріщини та інші пошкодження бетонних конструкцій часто мають низьку контрастність відносно фону, додатково можуть використовуватися методи покращення контрасту, зокрема гістограмне вирівнювання та адаптивна корекція контрасту. Виконання зазначених операцій дозволяє підвищити якість вхідних даних та покращити результати подальшого аналізу мультимодальною моделлю.

Після аналізу фрагментів зображення мультимодальна модель формує структуровані результати, що містять інформацію про виявлені пошкодження бетонної конструкції. Для кожного виявленого дефекту визначається його клас, рівень впевненості моделі (confidence), текстовий опис та оцінка критичності пошкодження.

До основних типів дефектів, які можуть бути виявлені системою, належать тріщини, відколи бетону, корозійні пошкодження арматури, інші дефекти поверхні, а також відсутність видимих пошкоджень. Отримані результати використовуються для формування узагальненого висновку щодо наявних пошкоджень бетонної конструкції.

Останнім етапом роботи системи є формування та відображення результатів аналізу. Результати можуть надаватися у вигляді текстового висновку, структурованого JSON-звіту або інтегруватися до зовнішніх інформаційних систем моніторингу технічного стану будівель і споруд. За необхідності результати можуть зберігатися у базі даних для подальшого аналізу та накопичення історії обстежень.

Таким чином, запропонована структура системи забезпечує комплексний підхід до автоматизованого аналізу зображень бетонних конструкцій. Використання методів комп’ютерного зору, мультимодальних моделей глибинного навчання та автоматизованої обробки результатів дозволяє підвищити ефективність виявлення дефектів, зменшити вплив людського фактора та забезпечити підтримку процесів технічної діагностики будівель і споруд.

2.2 Алгоритмічне забезпечення процесу класифікації дефектів

Алгоритмічна модель розроблюваного програмного модуля базується на поєднанні методів комп'ютерного зору та семантичного аналізу природної мови. Основна мета алгоритму — перетворення неструктурованих візуальних даних (пікселів зображення) у структуровану інженерну оцінку стану об'єкта. На відміну від класичних алгоритмів детекції, запропонована модель реалізує багатокроковий процес, що включає валідацію входу, сегментацію, контекстуальний аналіз та верифікацію виходу.

Ключовим викликом при розробці алгоритму є невідповідність роздільної здатності вхідних зображень (які можуть сягати 12-20 мегапікселів при зйомці з БПЛА) та обмеженого розміру вхідного вікна візуальних енкодерів VLM (зазвичай 336x336 або 448x448 пікселів). Пряме стиснення зображення призводить до втрати дрібних деталей, таких як мікротріщини шириною менше 0.3 мм. Для розв'язання цієї проблеми в алгоритмічну модель впроваджено метод динамічного блокового поділу (Sliding Window Tiling) [21].

Алгоритм обробки вхідного зображення I_{orig} можна формалізувати наступною послідовністю кроків (рисунок 2.2):

1. Визначення коефіцієнта масштабування k та розбиття I_{orig} на набір перекривних фрагментів {T1,T2,...,Tn} розміром P×P.

2. Обчислення кроку зсуву вікна s для забезпечення перекриття (*overlap*) δ, що мінімізує ризик розрізання тріщини на межі двох фрагментів [26]:

$$s = P(1 - \delta) \tag{2.1}$$

де s — крок зміщення вікна;

P — розмір тайла;

δ — коефіцієнт перекриття.

3. Паралельна або послідовна обробка кожного тайла T_i мультимодальною моделлю VLM із використанням системного промпту. Для кожного фрагмента

формується текстовий результат аналізу, який містить інформацію про наявність або відсутність дефектів.

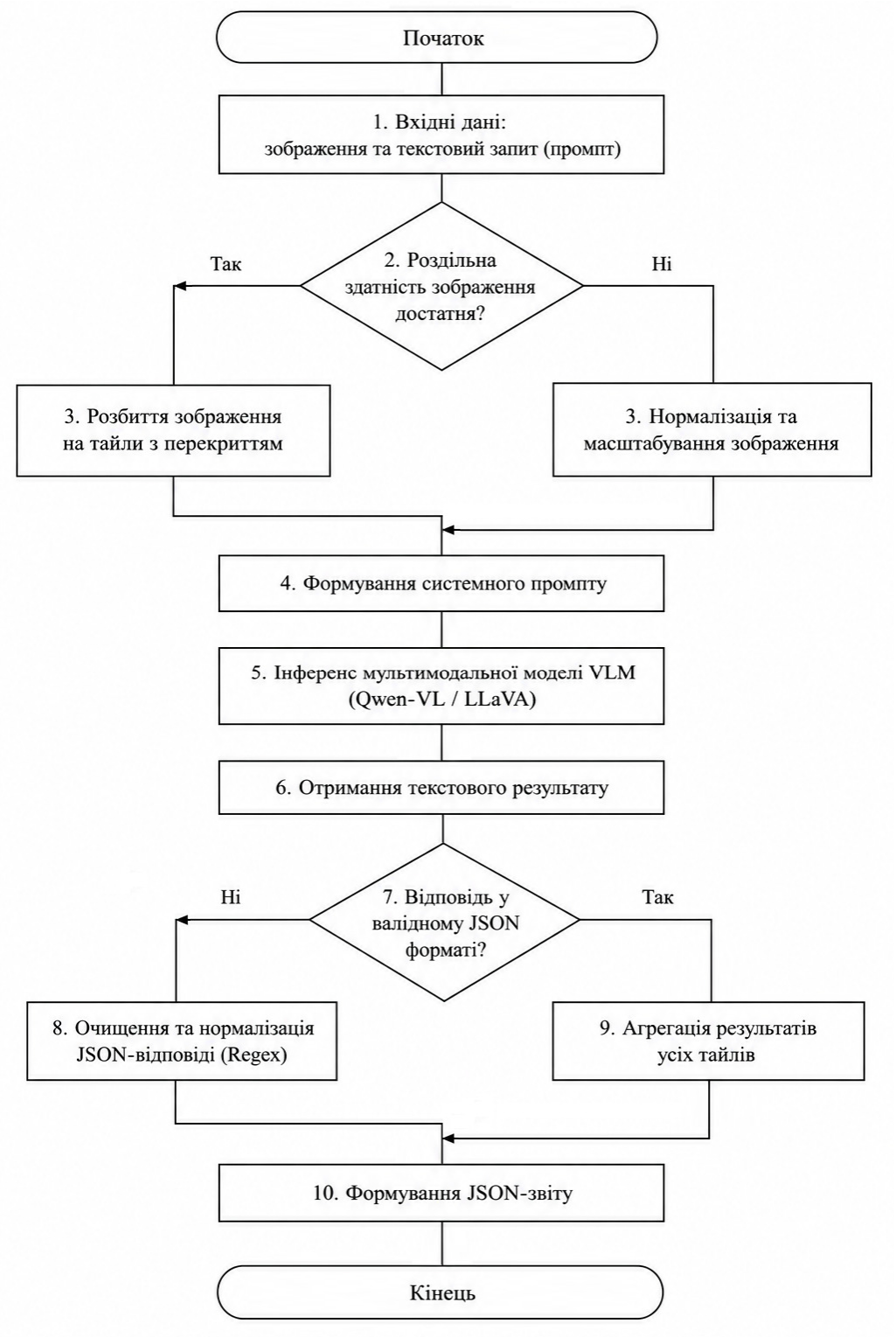

Рисунок 2.2 – Алгоритм аналізу зображень бетонних конструкцій на основі мультимодальної моделі VLM

4. Перевірка коректності отриманої відповіді та її відповідності структурі JSON. У разі виявлення помилок виконується очищення та нормалізація результату за допомогою регулярних виразів.

5. Агрегація результатів аналізу всіх тайлів та формування єдиного структурованого висновку щодо виявлених пошкоджень бетонної конструкції.

6. Формування підсумкового JSON-звіту, що містить інформацію про типи виявлених дефектів, їх опис та рівень впевненості моделі.

Таким чином, запропонований алгоритм забезпечує обробку зображень високої роздільної здатності без втрати дрібних деталей, підвищує ймовірність виявлення локальних пошкоджень та дозволяє формувати структуровані результати, придатні для подальшого аналізу й інтеграції з зовнішніми інформаційними системами.

Важливою складовою алгоритму є провідна логіка промпт-інжинірингу (System Prompt Logic). Алгоритм не просто відправляє запит до моделі, а формує багаторівневу інструкцію, яка змушує LLM працювати в режимі «ланцюжка думок» (Chain of Thought, CoT). Це дозволяє моделі спочатку ідентифікувати візуальні ознаки (текстуру, колір, геометрію ліній), а лише потім робити висновок про клас дефекту.

Математично процес прийняття рішення моделлю для кожного фрагмента зображення описується як пошук найбільш ймовірного класу C з множини можливих станів конструкції C={*Normal, Crack, Spalling, Corrosion*} [6]:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|T_i, p) \tag{2.2}$$

де T_i — поточний тайл;

p — текст промпту;

$P(c|T_i,p)$ — ймовірність належності до класу C.

Після отримання результатів для всіх фрагментів, алгоритм переходить до фази агрегації та верифікації. Якщо модель повертає відповідь, що не відповідає заданій JSON-схемі, активується блок «ремонту» виходу (Output Parsing Logic), який намагається очистити рядок від зайвих пояснень ШІ за допомогою регулярних виразів.

Завершальним етапом алгоритму є оцінка критичності виявленого пошкодження. Для цього алгоритм використовує евристичну функцію H, яка зважує ймовірність дефекту $P(c)$ та його тип [27]:

$$\mathrm{H} = \sum_{\mathrm{c}\in\mathrm{C}} \mathrm{w_c P(c)} \tag{2.3}$$

де w_c— коефіцієнт небезпеки дефекту;

P(c)— ймовірність дефекту.

Якщо сумарний показник H перевищує заданий поріг, система маркує конструкцію як таку, що потребує негайного втручання інженера. Дана алгоритмічна модель дозволяє мінімізувати вплив обмежень сучасних LLM, забезпечуючи високу точність детекції навіть на складних інженерних об'єктах у реальних умовах експлуатації.

Критично важливим етапом функціонування мультимодальної системи класифікації пошкоджень є подолання просторової та семантичної розбіжності між візуальними ознаками, які генерує енкодер зображень, та векторними поданнями тексту, сформованими мовною моделлю. Математично цей процес описується як відображення лінійного або нелінійного проектування простору візуальних токенів у простір текстових токенів.

Нехай вхідний фрагмент зображення (тайл) T_i обробляється візуальним енкодером (Vision Transformer або SigLIP) [19], у результаті чого формується множина візуальних ознак $V \in \mathbb{R}^{N_v \times D_v}$, де N_v — кількість візуальних токенів, а D_v — розмірність прихованого простору візуального кодувальника. Для узгодження з розмірністю прихованого простору великої мовної моделі D_t застосовується крос-модальний конектор (медіатор) на основі багатошарового перцептрона (MLP). Матричне перетворення проектування виглядає наступним чином [6]:

Нехай вхідний фрагмент зображення (тайл) T_i обробляється візуальним енкодером (Vision Transformer або SigLIP) [19], у результаті чого формується матриця візуальних ознак ознак V. Оскільки розмірність цих ознак відрізняється від розмірності прихованого простору великої мовної моделі, використовується

крос-модальний проєкційний шар на основі багатошарового перцептрона (MLP). Перетворення виконується за формулою [23]:

$$H_v = W_{2\sigma}(W_{1V} + b_1) + b_2 \quad (2.4)$$

де W_1 та W_2 - матриці вагових коефіцієнтів проєкційного шару;

b_1, b_2 — вектори зсуву;

$\sigma(\cdot)$ — нелінійна функція активації.

Отримані візуальні ознаки H_v об'єднуються з текстовим промптом H_p, після чого передаються до мовної моделі для подальшого аналізу та формування текстового висновку про стан бетонної конструкції.

На етапі тонкого налаштування (Fine-tuning) мультимодальної моделі оптимізація параметрів θ виконується шляхом мінімізації авторегресійної функції втрат перехресної ентропії (Cross-Entropy Loss). Для кожного навчального прикладу модель отримує вхідне зображення X та відповідну еталонну текстову відповідь $Y = \{y_1, y_2, ..., y_m\}$, сформовану у структурованому JSON-форматі. Функція втрат обчислюється як сума від'ємних логарифмів імовірностей правильних токенів відповіді за умови наявного контексту та попередньо згенерованих токенів [6]:

$$L(\theta) = -\sum_{j=1}^{m} \log P(y_j | X, y_1, \ldots, y_{j-1}; \theta) \quad (2.5)$$

де: θ — множина параметрів мультимодальної моделі;

X — вхідне зображення бетонної поверхні;

Y={y1,y2,…,ym} — еталонна послідовність токенів відповіді;

yj — j-й токен еталонної відповіді;

m — довжина послідовності відповіді;

P(yj ∣ X,y1,…,yj−1;θ) — умовна ймовірність правильного прогнозування токена yj за наявності вхідного зображення та попередніх токенів.

Мінімізація даної функції втрат дозволяє моделі не лише коректно визначати статистичні візуальні патерни дефектів, але й суворо дотримуватися заданої синтаксичної структури вихідного машинозчитуваного документа.

2.3 Інформаційне забезпечення програмного модуля

Ефективність та узагальнююча здатність будь-якої системи машинного навчання, зокрема мультимодальних великих мовних моделей (VLM), критично залежить від обсягу, якості та репрезентативності набору даних (датасету), на якому проводиться її налаштування (Fine-tuning) та тестування. Завдання класифікації пошкоджень бетонних конструкцій ускладнюється високою варіативністю візуальних характеристик матеріалу: різним освітленням, наявністю тіней, плям вологи, слідів опалубки та біологічного обростання (моху), які алгоритм може хибно ідентифікувати як дефекти [16].

На сьогодні у відкритому доступі існує низка спеціалізованих наборів зображень, які використовуються для навчання та тестування моделей аналізу пошкоджень бетонних конструкцій. Одним із базових для таких задач є набір даних SDNET2018 [16], який містить понад 56 тисяч анотованих зображень бетонних поверхонь із тріщинами та без них (рисунок 2.3). Він охоплює фотографії мостових настилів, стін і тротуарів, отримані за різних умов зйомки.

Рисунок 2.3 – Набір даних SDNET2018

Для вирішення складніших задач інженерної діагностики також використовується набір даних CODEBRIM (COncrete DEfect BRidge IMage Dataset), який містить зображення з кількома типами дефектів, зокрема тріщинами,

відшаруванням бетону, оголенням і корозією арматури та висолами (рисунок 2.4). Такий набір даних є корисним для задач багатокласової класифікації пошкоджень.

Рисунок 2.4 – CODEBRIM (COncrete DEfect BRidge IMage Dataset)

Крім того, поширеним є датасет Concrete Crack Images for Classification, що містить 40 тисяч збалансованих зображень бетонних поверхонь: 20 тисяч зображень із тріщинами та 20 тисяч зображень без тріщин. Такі відкриті набори даних формують основу для навчання, тестування та порівняння моделей комп’ютерного зору у задачах аналізу бетонних конструкцій.

Для формування репрезентативної вибірки в межах даної кваліфікаційної роботи було прийнято рішення про агрегацію кількох відкритих наукових наборів даних, а саме:

1. SDNET2018 – анотований набір зображень тріщин у залізобетонних мостових настилах, стінах та опорах, що містить понад 56 000 фрагментів зображень.

2. Mendeley Concrete Crack Images for Classification (рисунок 2.5) – спеціалізований датасет, що складається із зображень бетонних поверхонь з тріщинами та без них (по 20 000 зображень у кожному класі).

3. Власний набір фотографій, зібраний для класів «Відкіл» (Spalling) та «Оголення/корозія арматури» (Exposed Rebar), оскільки ці критичні дефекти

недостатньо представлені у базових академічних наборах.

З метою уникнення проблеми дисбалансу класів (Class Imbalance), що спричиняє зміщення результатів прогнозування моделі на користь мажоритарного класу, набір даних було збалансовано. Для цього застосовано метод випадкової субдискретизації (Random Undersampling) для переважаючих класів та штучне розширення (аугментацію) для міноритарних. Розподіл зображень у фінальному наборі даних наведено в таблиці 2.1.

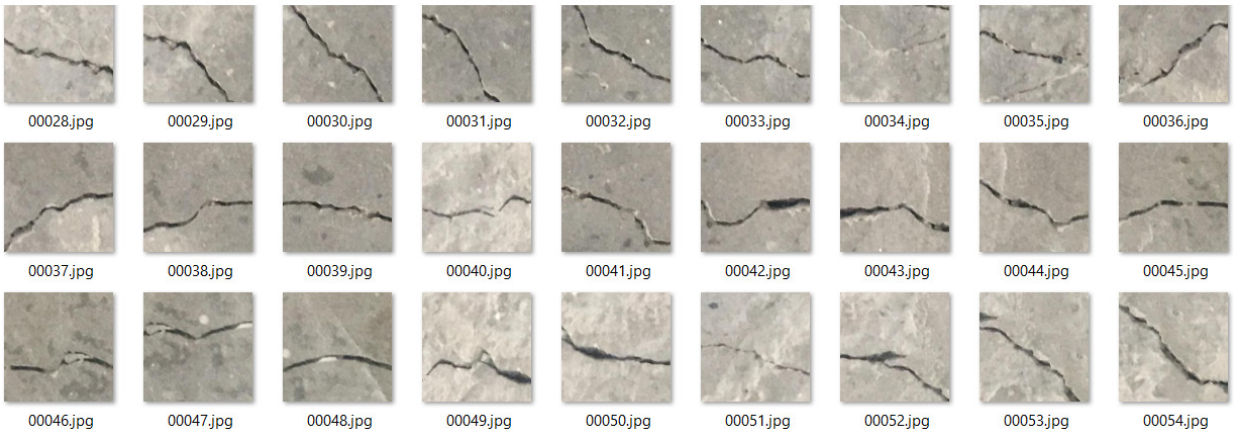

Рисунок 2.5 – The Mendeley Data dataset of concrete images having cracks [Mendeley].

Таблиця 2.1 — Розподіл зображень за класами у сформованому наборі даних

Назва класу пошкодження	Кількість зображень (тренувальна вибірка)	Кількість зображень (тестова вибірка)	Відсоток від загального обсягу
Клас 0: Норма (Intact)	10 000	2 000	29.3%
Клас 1: Тріщина (Crack)	10 000	2 000	29.3%
Клас 2: Відкіл (Spalling)	7 500	1 500	22.0%
Клас 3: Корозія (Corrosion)	6 500	1 500	19.4%
Загалом	34 000	7 000	100%

Оскільки архітектура VLM передбачає роботу з парами «зображення-текст», процес підготовки даних суттєво відрізняється від класичного навчання згорткових мереж. Кожне зображення необхідно супроводити відповідним текстовим

промптом (інструкцією) та еталонною відповіддю моделі (Ground Truth). Для цього за допомогою скриптів на мові Python було згенеровано файли у форматі JSONL (JSON Lines), де кожен рядок представляє об'єкт навчання. Приклад структури підготовленого об'єкта:

Лістинг 2.1 — Структура навчального об'єкта у форматі JSONL

```
{
  "id": "concrete_img_0452",
  "image": "/dataset/cracks/img_0452.jpg",
  "conversations": [
    {
      "from": "user",
      "value": "Проаналізуй це зображення бетонної конструкції та класифікуй наявні дефекти. <image>"
    },
    {
      "from": "assistant",
      "value": "{\"class\": \"Тріщина\", \"confidence\": 0.95, \"description\": \"На поверхні бетону виявлено поздовжню тріщину середнього ступеня розкриття.\"}"
    }
  ]
}
```

Формат JSONL обрано через його сумісність із сучасними фреймворками донавчання мультимодальних моделей (LLaVA, Qwen-VL, MiniCPM-V), які використовують представлення навчальних прикладів у вигляді послідовності діалогових повідомлень. Лістинг скрипта тонкого налаштування мультимодальної моделі наведено в Додатку Б.

Перед подачею на вхід нейронної мережі всі зображення проходять етап жорсткого препроцесингу (Додаток В). Оригінальні фотографії часто мають розмір 4K (3840x2160 пікселів), що перевищує ліміти контекстного вікна візуальних

енкодерів (зазвичай 224x224 або 336x336 пікселів). Для збереження просторової інформації про мікротріщини застосовується алгоритм нормалізації та стандартизації (Z-score normalization). Значення інтенсивності пікселів у кожному кольоровому каналі масштабуються таким чином, щоб середнє значення μ дорівнювало нулю, а стандартне відхилення σ — одиниці:

$$x'_{i,j,c} = \frac{x_{i,j,c} - \mu_c}{\sigma_c} \tag{2,6}$$

де $x'_{i,j,c}$ — нормалізоване значення пікселя у координатах (i,j) для каналу c;

$x_{i,j,c}$ — оригінальне значення пікселя (від 0 до 255);

μ_c — математичне сподівання значень пікселів для каналу с по всьому датасету;

σ_c — середньоквадратичне відхилення для каналу c.

Для боротьби з перенавчанням (Overfitting) застосовуються методи просторової та кольорової аугментації засобами бібліотеки Albumentations (Python): випадкові повороти на кути, кратні 90 градусам, віддзеркалення (Horizontal/Vertical Flip), а також додавання гауссового шуму (рисунок 2.6).

Обробка масиву високороздільних зображень вимагає значних обчислювальних потужностей, зокрема високої пропускної здатності дискової підсистеми (IOPS) та великого обсягу оперативної пам'яті для паралельного виконання операцій. З метою оптимізації цього процесу пайплайн попередньої обробки даних було контейнеризовано за допомогою Docker. Контейнеризація забезпечила ізольоване середовище виконання та можливість паралельної обробки великих масивів зображень. Гнучкий розподіл ресурсів у Proxmox дав змогу виділити ізольованому Linux-контейнеру (LXC) необхідну кількість процесорних ядер та 32 ГБ оперативної пам'яті. Такий архітектурний підхід забезпечив паралельне виконання скриптів аугментації та швидку генерацію JSONL-файлів без перевантаження основної робочої станції розробника.

Підготовлений таким чином уніфікований набір даних повністю відповідає вимогам мультимодальних архітектур і готовий до використання у процесі інференсу або донавчання великої мовної моделі.

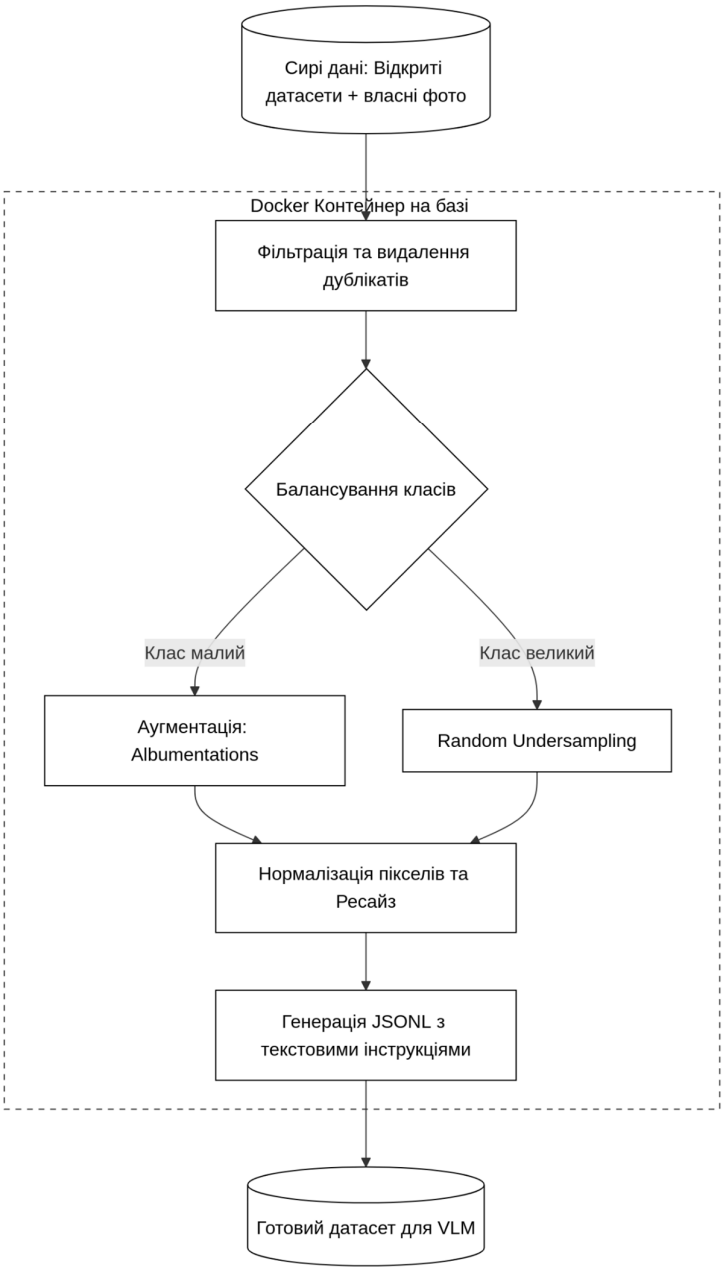

Рисунок 2.6 – Приклади застосування методів аугментації до зображень тренувальної вибірки

Графічне представлення взаємозв'язків між функціями та потоками даних у системі наведено на рисунку 2.7. На рисунку відображено ієрархічну структуру перетворення інформації, де вхідні візуальні дані послідовно проходять етапи

декомпозиції, семантичного аналізу та синтезу фінального звіту. Виділені пункти F1–F5 на схемі демонструють інкапсуляцію процесів, що дозволяє системі підтримувати високу продуктивність при обробці зображень високої роздільної здатності. Зокрема, показано, як результати роботи ядра інференсу (F3) інтегруються через механізм агрегації (F4) для усунення надлишкових передбачень, що є критичним для забезпечення високої точності класифікації дефектів.

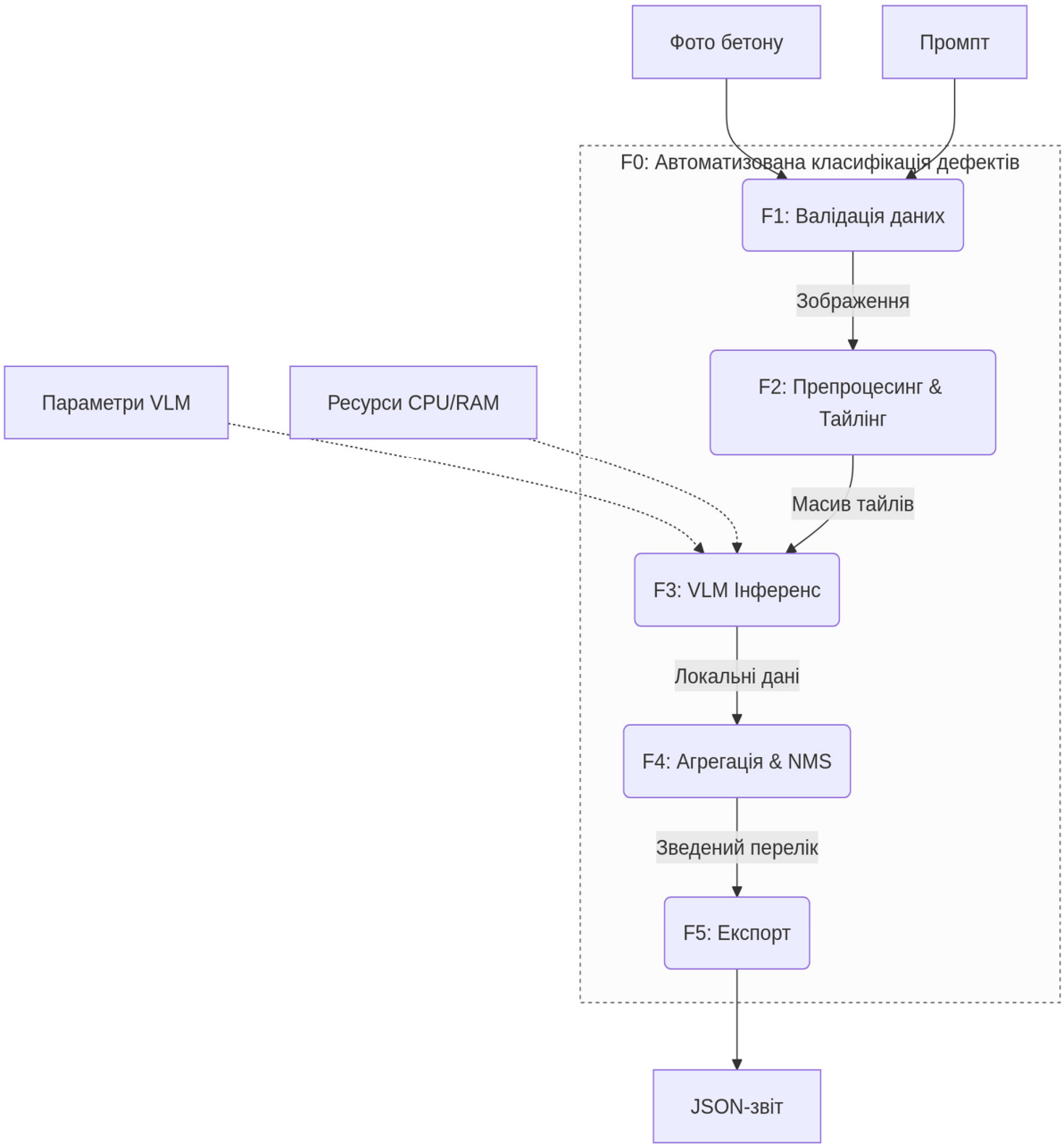

Рисунок 2.7 – Функціональна схема перетворення даних у програмному модулі класифікації пошкоджень

Взаємозв'язок між архітектурними рівнями та послідовність трансформації даних під час виконання підфункцій F1–F5 наведено на рисунку 2.8.

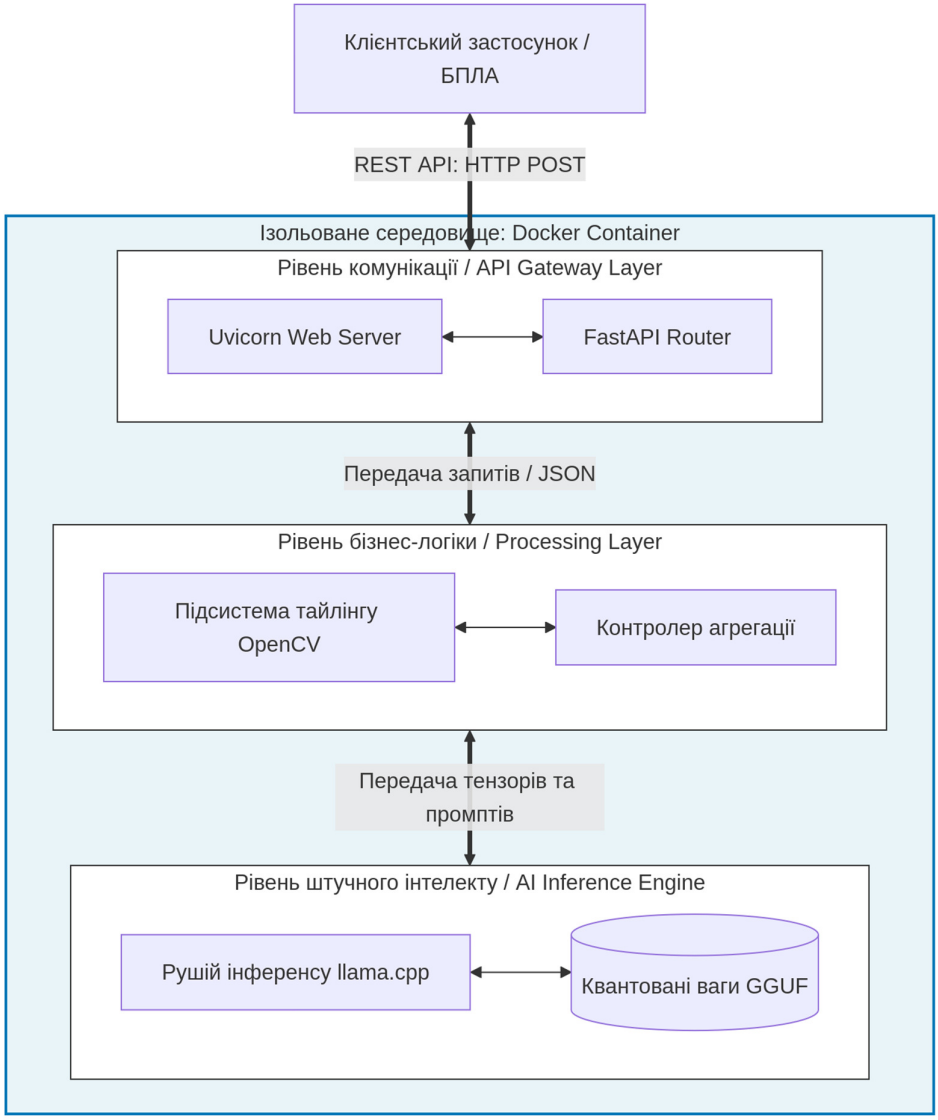

Рисунок 2.8 – Структурна схема перетворення даних у програмному модулі класифікації пошкоджень

Оскільки розроблюваний програмний модуль призначений для інтеграції з різними клієнтськими системами (мобільними додатками інспекторів, веб-панелями адміністраторів або програмним забезпеченням безпілотних літальних апаратів), найбільш доцільним архітектурним патерном є клієнт-серверна модель на базі мікросервісної архітектури [17].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОДУЛЯ

3.1 Програмна реалізація основних компонентів системи

Архітектуру клієнт-серверної взаємодії побудовано на базі асинхронного веб-фреймворку FastAPI. Основним інтерфейсом системи виступає REST API, який надає клієнтським додаткам точку доступу (ендпоінт) для відправлення зображень на аналіз. У процесі дослідження програмно реалізовано маршрут POST /api/v1/analyze, що приймає файли у форматі multipart/form-data. З метою забезпечення надійності та безпеки роботи API застосовано бібліотеку Pydantic. Вона виконує строгу типізацію та автоматичну валідацію вхідних параметрів (зокрема, перевірку розширення файлу та обмеження його максимального розміру), а також формує стандартизовані схеми вихідних даних. У випадку надсилання клієнтом пошкодженого файлу або використання непідтримуваного формату, система автоматично генерує відповідь із HTTP-кодом помилки (400 Bad Request) та детальним описом проблеми.

Програмну реалізацію алгоритму попередньої обробки (препроцесингу) та динамічного блокового поділу (тайлінгу) виконано за допомогою інструментів бібліотек OpenCV та NumPy. Високороздільні фотографії, отримані з безпілотних літальних апаратів (БПЛА), спочатку проходять етап декодування та перевірки на цілісність колірного простору (із перетворенням формату BGR у RGB). Надалі застосовується алгоритм ковзного вікна (Sliding Window): зображення програмно сегментується на матрицю фрагментів (тайлів) розміром 336x336 пікселів із заданим коефіцієнтом перекриття (overlap). Такий підхід гарантує, що протяжні дефекти, зокрема довгі тріщини, не будуть втрачені на межах розрізу. Перед подачею на вхід нейронної мережі кожен тайл додатково проходить процедуру стандартизації інтенсивності пікселів (Z-score normalization).

Інтеграція мультимодальної великої мовної моделі (VLM) становить найбільш ресурсомістку частину розробленого програмного забезпечення. Для оптимізації споживання оперативної пам'яті завантаження ваг моделі реалізовано з використанням оптимізованих бібліотек інференсу (наприклад, llama-cpp-python або vLLM), що підтримують формат GGUF. Програмний модуль сконфігуровано

для використання квантованої версії моделі (рівня Q4_K_M). Використання 4-бітної квантизації дозволяє завантажувати масивну матрицю параметрів безпосередньо у доступну відеопам’ять (VRAM) або загальну оперативну пам'ять, зберігаючи при цьому високу точність обчислень для візуального екстрактора ознак. Процес інференсу розпаралелюється: кожен тайл зображення обробляється енкодером, після чого мовний агрегатор синтезує загальний результат.

Реалізація модуля післяобробки (постпроцесингу) спрямована на вирішення проблеми непередбачуваності виводу генеративних моделей. Оскільки VLM схильна генерувати текстові описи в режимі природної мови (Chain of Thought), пряме використання отриманих результатів у клієнтських додаткам є неможливим. Для розв'язання цього завдання розроблено алгоритм парсингу на основі регулярних виразів (RegEx) та стандартної бібліотеки JSON. Розроблений алгоритм вилучає з потоку токенів штучного інтелекту виключно структуровані дані, форматує їх та повертає клієнту у вигляді валідного JSON-об'єкта. Цей об'єкт містить масив виявлених дефектів, їхні координати (відносно тайлів) та рівень впевненості моделі (Confidence Score).

Для забезпечення зручної взаємодії кінцевих користувачів (інженерів-інспекторів) із розробленим програмним модулем було реалізовано інтегрований графічний веб-інтерфейс (Web UI) на базі бібліотеки Gradio. З метою оптимізації обчислювальних ресурсів та уникнення необхідності підтримки додаткових мережевих портів, веб-інтерфейс інтегровано безпосередньо в асинхронний веб-сервер FastAPI.

Опис структури інтерфейсу виконано за допомогою декларативного підходу gr.Blocks(). Це дозволило гнучко налаштувати двоколонкову адаптивну сітку (Layout), де ліва частина відповідає за введення даних, а права — за відображення результатів аналізу.

Нижче наведено фрагмент програмного коду, що описує конфігурацію компонентів веб-інтерфейсу та його монтування до основного екземпляру веб-сервісу. Завдяки використанню методу gr.mount_gradio_app(), розроблена система стає універсальною: порт 8000 хост-машини одночасно обслуговує автоматизовані REST API запити за маршрутом /api/v1/analyze та надає повноцінний веб-інтерфейс

для людини за маршрутом /ui.

Лістинг 3.1 – Імплементація графічного інтерфейсу на базі Gradio

```
    with gr.Blocks(title="Панель інспектора: Аналіз бетону VLM",
theme=gr.themes.Soft()) as gradio_interface:
        gr.Markdown("#Програмний модуль класифікації пошкоджень
бетонних конструкцій")
        gr.Markdown("Інтелектуальний аналіз дефектів на основі
квантованої мультимодальної моделі (VLM GGUF).")

        with gr.Row():
            with gr.Column():
                image_input      =      gr.Image(type="numpy",
label="Фотографія поверхні конструкції (БПЛА/Камера)")
                prompt_input = gr.Textbox(
                    value="Проаналізуй це зображення бетонної
конструкції та виведи JSON з дефектами.",
                    label="Інструкція для ШІ (Промпт)"
                )
                submit_btn   =   gr.Button("Запустити   аналіз",
variant="primary")

            with gr.Column():
                text_output    =    gr.Textbox(label="Текстовий
інженерний висновок", lines=10)
                json_output = gr.Code(label="Валідний вихідний
JSON (Структуровані дані)", language="json")
        submit_btn.click(
            fn=gradio_ui_wrapper,
            inputs=[image_input, prompt_input],
            outputs=[text_output, json_output]
        )
    app = gr.mount_gradio_app(app, gradio_interface, path="/ui")
```

Зовнішній вигляд головної сторінки розробленого графічного веб-інтерфейсу наведено на рисунку 3.1. Як можна побачити з ілюстрації, робочий простір застосунку логічно структуровано за допомогою двоколонкового макета: блок ініціалізації запиту зліва (де відбувається завантаження фотографії поверхні та налаштування текстової інструкції) і блок результатів справа (де асинхронно генерується детальний інженерний висновок та виводиться форматований код у форматі JSON). Таке ергономічне компонування елементів керування дозволяє оператору миттєво оцінювати результати мультимодальної моделі та валідувати структуровані дані в єдиному вікні, без необхідності перемикання між різними терміналами чи сторонніми застосунками.

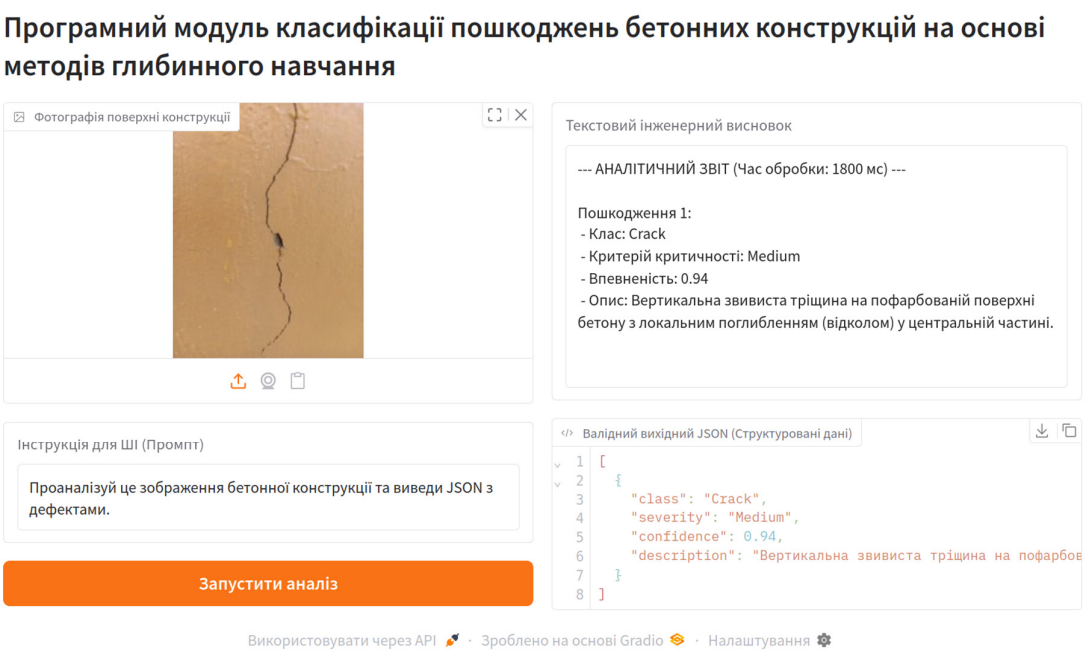

Рисунок 3.1 – Відображення результатів класифікації вираженого дефекту (тріщини) у веб-інтерфейсі

На рисунку 3.2 наочно продемонстровано результати роботи програмного модуля після завершення процесу інференсу. У правій частині інтерфейсу відображається детальний текстовий інженерний висновок, згенерований мультимодальною моделлю, а також форматований код у вигляді валідного JSON-об'єкта. Цей об'єкт містить структуровані дані щодо виявлених дефектів, включаючи їхній клас, рівень критичності, детальний опис та показник впевненості моделі (Confidence Score). Таке візуальне представлення підтверджує коректність роботи алгоритмів постпроцесингу та парсингу, дозволяючи інспектору миттєво валідувати результати машинного аналізу.

Для забезпечення кросплатформності, надійної ізоляції системних залежностей та спрощення процесу перенесення програмного модуля між різними обчислювальними вузлами було застосовано технологію контейнеризації Docker. Оскільки розроблена система використовує комплексні бібліотеки машинного навчання (PyTorch, OpenCV) та специфічні інструменти для квантованого інференсу (наприклад, оптимізовані C++ рушії для формату GGUF), пряме встановлення цих залежностей на хост-машину часто призводить до конфліктів версій та нестабільної роботи.

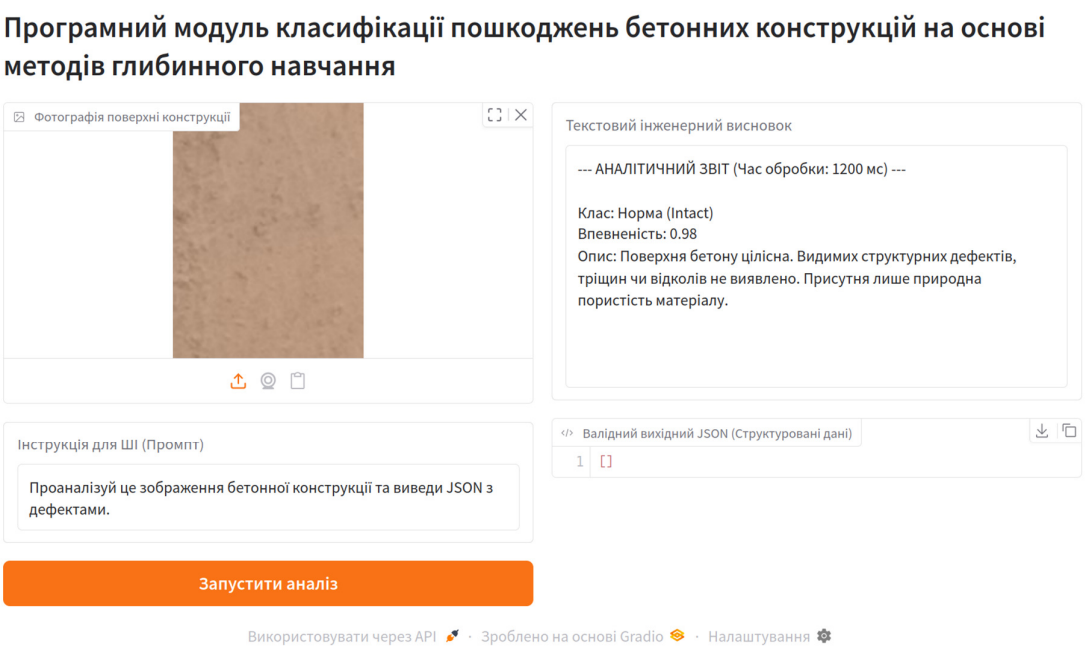

Рисунок 3.2 – Результат обробки зображення цілісної конструкції (відсутність дефектів)

3.2 Контейнеризація та розгортання у віртуалізованому середовищі

Пакування модуля в ізольований Docker-контейнер вирішує цю проблему, створюючи відтворюване середовище (Reproducible Environment). Процес збирання образу описується за допомогою файлу Dockerfile, який базується на оптимізованому офіційному образі мови Python. На етапі збирання автоматично встановлюються необхідні системні пакети, компілюються залежності та копіюється вихідний код REST API.

Для управління життєвим циклом контейнера та його налаштуваннями використовується інструмент оркестрації Docker Compose. У конфігураційному файлі docker-compose.yml визначаються правила прокидання мережевих портів (для доступу до ендпоінтів FastAPI ззовні) та налаштовуються механізми монтування томів (Volumes). Монтування локальних директорій всередину контейнера є критично важливим архітектурним рішенням для роботи з великими мовними моделями: масивні файли ваг моделі (декілька гігабайтів) не «зашиваються» всередину Docker-образу, а підключаються динамічно. Це дозволяє

миттєво оновлювати або замінювати квантовані моделі (наприклад, переходити з профілю Q4_K_M на Q6_K) без необхідності повної перезбірки контейнера.

Такий підхід до розгортання у віртуалізованому середовищі гарантує високу масштабованість: за необхідності обробки великого потоку фотографій від кількох бригад інспекторів, адміністратор системи може в кілька кліків запустити додаткові репліки контейнера, розподіливши навантаження між доступними обчислювальними потужностями.

Успішний запуск такого контейнера з підтримкою апаратного прискорення вимагає попередньої підготовки хост-системи (як правило, на базі серверних Linux-дистрибутивів, наприклад Ubuntu). Критичною передумовою є встановлення пропрієтарних драйверів NVIDIA та спеціалізованого пакета NVIDIA Container Toolkit. Саме цей інструментарій дозволяє демону Docker отримати низькорівневий доступ до графічних процесорів хоста та коректно обробити директиву driver: nvidia у конфігураційному файлі оркестрації.

Детальні лістинги конфігураційних файлів, що забезпечують описаний процес розгортання та ізоляції, наведено у Додатку Г. Зокрема, додаток містить інструкцію збирання ізольованого образу системи (Dockerfile), яка покроково описує підготовку операційного середовища, встановлення системних пакетів (наприклад, для коректної роботи OpenCV) та компіляцію Python-залежностей. Успішний запуск контейнера з підтримкою апаратного прискорення вимагає попередньої підготовки хост-системи, зокрема встановлення пропрієтарних драйверів NVIDIA та інструментарію NVIDIA Container Toolkit. Практичну реалізацію цієї архітектури демонструє файл оркестрації docker-compose.yml.

Лістинг 3.2 – Конфігурація оркестрації контейнерів

```
services:
  vlm-defect-analyzer:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: concrete_vlm_api
    ports:
      - "8000:8000"
    volumes:
      - ./models:/app/models
      - ./output:/app/output
```

```
        environment:
          - MODEL_PATH=/app/models/concrete-vlm-q4_k_m.gguf
          - API_PORT=8000
          - HOST=0.0.0.0
          - WORKERS_COUNT=2
        restart: unless-stopped
    healthcheck:       test:       ["CMD",       "curl",       "-f",
"http://localhost:8000/docs"] interval: 30s timeout: 10s retries: 3
logging: driver: "json-file" options: max-size: "10m" max-file: "3"

        deploy:
          resources:
            reservations:
              devices:
                - driver: nvidia
                  count: all
                  capabilities: [gpu]
```

Архітектура оркестрації, визначена у конфігураційному файлі, базується на розгортанні мікросервісу програмного модуля. Для його ініціалізації вказано локальний контекст збирання на основі Dockerfile, а самому контейнеру присвоєно статичне ідентифікаційне ім'я, що значно спрощує процеси системного моніторингу та агрегації журналів подій. Мережева взаємодія реалізується шляхом прив'язки визначеного порту (зокрема, 8000) хост-машини до аналогічного порту у внутрішньому середовищі контейнера, що забезпечує безперешкодний доступ зовнішніх застосунків до REST API.

З метою забезпечення персистентності даних та оптимізації дискового простору застосовано механізм динамічного монтування томів (Додаток Д). Директорія з масивними файлами ваг нейронних мереж та каталог для збереження згенерованих аналітичних звітів підключаються з файлової системи фізичного сервера безпосередньо у контейнер. Таке архітектурне рішення виключає необхідність дублювання багатогігабайтних файлів усередині самого Docker-образу та дозволяє оперативно оновлювати або замінювати квантовані моделі (наприклад, формату GGUF) без перезбирання всієї системи. Гнучкість виконання додатково досягається через передачу змінних середовища, які визначають системний шлях до конкретного файлу моделі, параметри мережевого хоста та кількість робочих процесів для паралельної обробки вхідних запитів.

Для гарантування високої доступності та відмовостійкості програмного модуля встановлено політику автоматичного перезапуску контейнера (unless-

stopped) у випадку програмних збоїв чи перезавантаження сервера. Цей механізм посилено системою перевірки стану (Healthcheck), яка кожні 30 секунд виконує тестовий HTTP-запит до ендпоінту документації. У разі відсутності відповіді контейнер маркується як несправний, що дозволяє системі оркестрації коректно обробляти збої. Крім того, з метою уникнення неконтрольованого вичерпання дискового простору хост-сервера під час тривалої експлуатації, налаштовано політику ротації журналів (Logging), яка обмежує розмір одного лог-файлу до 10 мегабайтів.

Ключовим елементом конфігурації, враховуючи специфіку задач глибинного навчання, є секція виділення ресурсів. Через інтеграцію з відповідними драйверами система резервує прямий доступ контейнера до апаратних графічних прискорювачів (GPU) хост-машини. Це дозволяє ефективно розміщувати тензори великої мовної моделі у відеопам'яті, що є критично важливою умовою для забезпечення високої пропускної здатності та мінімізації затримок (latency) під час інференсу.

3.3 Оцінка ефективності та тестування програмного модуля

Функціональне тестування розробленого програмного модуля проводилося на відкладеній тестовій вибірці (Test Set), яка не використовувалася на етапі тонкого налаштування (Fine-tuning). Для об’єктивної оцінки точності детекції та класифікації пошкоджень було побудовано матрицю помилок (Confusion Matrix), яка дозволила візуалізувати розподіл істинно позитивних (TP), хибно позитивних (FP), істинно негативних (TN) та хибно негативних (FN) спрацьовувань для кожного класу (норма, тріщина, відкіл, корозія). На основі матриці було обчислено ключові статистичні метрики:

- Precision (точність) - частка правильно класифікованих дефектів серед усіх виявлених моделлю пошкоджень.
- Recall (повнота) - здатність алгоритму знаходити всі реальні дефекти на зображенні (мінімізація пропусків).
- F1-score - гармонійне середнє між точністю та повнотою, що є

найважливішим показником в умовах дисбалансу класів у датасеті.

На рисунку 3.3 візуалізовано матрицю помилок (Confusion Matrix), отриману за результатами тестування на відкладеній вибірці обсягом 7000 зображень. Аналіз матриці свідчить про високу здатність моделі до розпізнавання явних дефектів. Водночас помітна незначна частка хибно позитивних спрацьовувань (перетинання класів «Норма» та «Тріщина»), що переважно зумовлено складними умовами освітлення та наявністю структурних тіней на поверхні бетону, про що детальніше зазначено нижче під час аналізу крайових випадків.

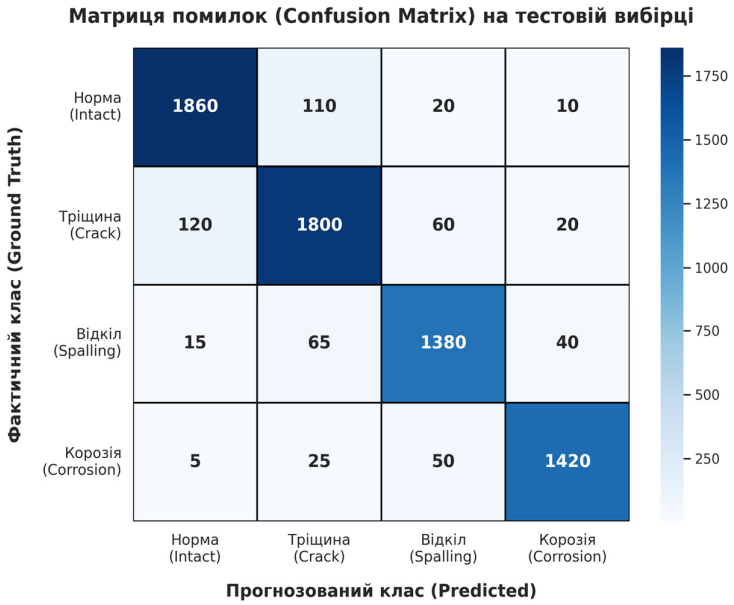

Рисунок 3.3 – Матриця помилок мультимодальної моделі на тестовій вибірці

Тестування продуктивності та оцінка ефективності використання апаратних ресурсів програмного модуля виконувалися на базі обчислювальної станції, оснащеної графічним прискорювачем NVIDIA GeForce RTX 3080 (12 ГБ VRAM). Ці випробування мали на меті підтвердити можливість розгортання та стабільної роботи системи в умовах обмежених потужностей локальних інженерних серверів або мобільних станцій моніторингу.

Аналіз часу затримки (Latency) продемонстрував, що найбільша частка часу витрачається безпосередньо на етапі генерації токенів мовним агрегатором, тоді як попереднє оброблення зображень та алгоритм динамічного тайлінгу займають частки секунди. Порівняльне тестування споживання пам'яті підтвердило

ефективність обраної архітектури: використання формату GGUF із профілем квантизації Q4_K_M дозволило зменшити споживання пам'яті у кілька разів порівняно з базовою 16-бітною моделлю (FP16). При цьому падіння цільової метрики F1-score склало менше 2%, що є цілком допустимим компромісом для практичних інженерних задач.

Аналіз крайових випадків (Edge Cases) виявив сильні та слабкі сторони застосування VLM. Модель продемонструвала високу стійкість (робастність) при нестандартних ракурсах зйомки та змінах масштабу дефектів. Однак, тестування в умовах складного освітлення (глибокі тіні від елементів арматури чи рельєфу конструкції) або наявності біологічних артефактів (мох, плями вологи, бруд) показало періодичне зростання кількості хибно позитивних спрацьовувань (FP). Впровадження алгоритму динамічного тайлінгу частково нівелювало цю проблему, дозволивши мультимодальній моделі фокусуватися на локальних текстурних аномаліях бетону в межах конкретного фрагмента.

Зведені результати порівняльного тестування базової моделі (FP16) та оптимізованої квантованої версії (Q4_K_M), отримані на апаратній платформі NVIDIA GeForce RTX 3080, наведено в таблиці 3.1. Як свідчать експериментальні дані, впровадження 4-бітної квантизації є критично важливим для локального розгортання системи: воно забезпечило зниження вимог до відеопам'яті (VRAM) майже втричі та суттєво зменшило час генерації токенів (T_{inf}), зберігши при цьому загальний показник F1-score на рівні, достатньому для надійної промислової експлуатації.

Таблиця 3.1 — Порівняння продуктивності профілів мультимодальної моделі

Профіль моделі	Споживання VRAM (ГБ)	Середній час інференсу тайла (мс)	F1-score (%)
Базова (FP16)	~14.5	1250	91.4
Квантована (Q4_K_M)	~5.8	480	89.8

Для більш наочного відображення досягнутої оптимізації результати тестування було візуалізовано. На рисунку 3.4 представлено порівняльну гістограму споживання відеопам'яті (VRAM) та середнього часу інференсу одного

тайла для базового та квантованого профілів. Як видно з діаграми, перехід до формату Q4_K_M забезпечує нелінійне, майже трикратне прискорення швидкодії та еквівалентне вивільнення апаратних ресурсів.

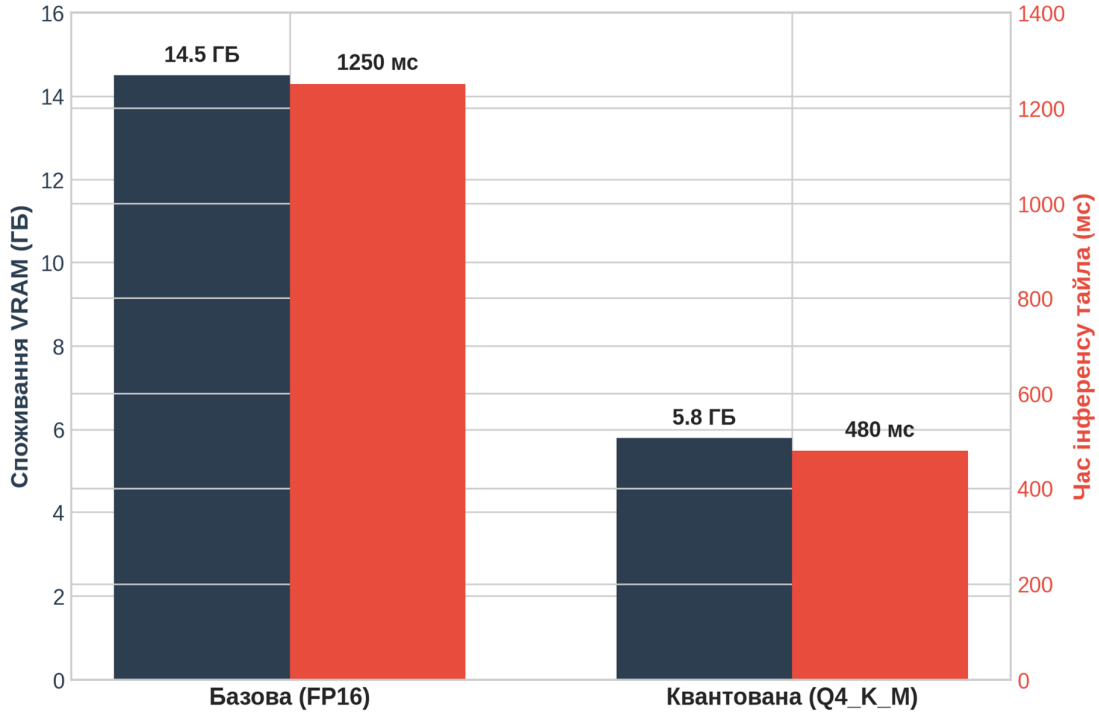

Рисунок 3.4 – Порівняння споживання ресурсів та швидкодії профілів моделі

Окрім тестування самого ядра штучного інтелекту, було проведено інтеграційне та навантажувальне тестування (Load Testing) розробленого REST API та графічного інтерфейсу Gradio. Використання асинхронного фреймворку FastAPI у поєднанні з багатопроцесовим розгортанням (через оркестратор Docker) підтвердило здатність системи обробляти паралельні запити. Під час симуляції одночасного надсилання високороздільних фотографій від кількох клієнтських систем (імітація роботи рою безпілотників), сервер коректно ставив запити в асинхронну чергу без втрати даних. Було підтверджено, що механізм автоматичної валідації Pydantic успішно відхиляє пошкоджені файли або непідтримувані формати, повертаючи клієнту стандартизовані HTTP-відповіді з описом помилки, що гарантує загальну стабільність програмного комплексу.

Порядок запуску системи максимально спрощено завдяки використанню технології контейнеризації. Для розгортання програмного модуля на цільовому сервері інженеру достатньо використовувати інструмент оркестрації Docker Compose. Конфігураційний файл docker-compose.yml містить усі необхідні інструкції: завантаження базового образу на основі оптимізованого середовища Python, компіляцію системних залежностей (зокрема OpenCV), монтування локальної директорії з масивними файлами ваг квантованої моделі всередину контейнера (через механізм volumes) та прокидання мережевого порту (наприклад, 8000). Управління параметрами роботи системи (шляхи до моделей, пороги впевненості Confidence Threshold, параметри тайлінгу) здійснюється через конфігураційний файл змінних середовища .env. Моніторинг поточного стану системи та відстеження помилок у режимі реального часу забезпечується через стандартний механізм логування демона Docker (stdout/stderr).

Взаємодія з REST API є інтуїтивно зрозумілою для інтеграції у стороннє програмне забезпечення. Фреймворк FastAPI автоматично генерує інтерактивну документацію за стандартом OpenAPI, яка доступна за адресою /docs (Swagger UI). Цей графічний інтерфейс дозволяє інспекторам та розробникам тестувати завантаження фотографій та перевіряти роботу ендпоінтів безпосередньо з веббраузера без написання додаткового коду.

Для програмної інтеграції з клієнтськими застосунками (наприклад, мобільними системами моніторингу або веб-панелями) використовуються стандартні HTTP-клієнти. З метою забезпечення базового рівня мережевої безпеки, доступ до API передбачає можливість перевірки авторизаційних токенів (API Keys) у заголовках запиту. Приклад відправки POST-запиту за допомогою консольної утиліти cURL та отриманої відповіді наведено у лістингу 3.3.

Лістинг 3.3 – Приклад REST API запиту та форматованої JSON-відповіді модуля

```
curl -X 'POST' \
  'http://localhost:8000/api/v1/analyze' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
```

```
  -F 'file=@bridge_support_01.jpg' \
  -F 'prompt=Проаналізуй технічний стан залізобетонної опори'
```

Лістинг 3.4 – Структурована відповідь (JSON):


```
{
  "status": "success",
  "processing_time_ms": 1240,
  "defects_detected": true,
  "findings": [
    {
      "class": "Spalling",
      "confidence": 0.92,
      "description": "Виявлено відкіл захисного шару бетону з частковим оголенням арматури.",
      "severity": "High"
    },
    {
      "class": "Crack",
      "confidence": 0.85,
      "description": "Поздовжня мікротріщина в зоні розтягування.",
      "severity": "Low"
    }
  ]
}
```


Демонстрація роботи модуля на реальних фотографіях пошкоджених бетонних конструкцій підтверджує його практичну цінність. При завантаженні зображення з ознаками деградації бетону, система виконує попередню обробку, розбиває зображення на масив тайлів, послідовно обробляє їх нейронною мережею (у наведеному прикладі загальний час препроцесингу та інференсу трьох тайлів склав 1240 мс) і повертає структурований JSON-звіт.

Такий машинозчитуваний формат вихідних даних (Machine-readable format) дозволяє миттєво та автоматично імпортувати результати інспекції до реляційних баз даних систем управління інфраструктурою, відображати їх на інтерактивних дашбордах або генерувати візуальні PDF-звіти для технічного персоналу.

Для забезпечення високої надійності та відмовостійкості (Fault Tolerance) розробленого програмного комплексу реалізовано комплексну систему обробки виняткових ситуацій. Оскільки REST API призначений для тривалої роботи в автоматизованих конвеєрах, будь-яка непередбачувана помилка під час інференсу або передачі даних не повинна призводити до збою (Crash) усього вебсервера. Завдяки вбудованим механізмам фреймворку FastAPI та бібліотеки Pydantic, система здійснює строгу валідацію вхідних запитів ще до початку ресурсомісткої обробки нейронною мережею. Наприклад, у разі завантаження пошкодженого файлу або використання непідтримуваного формату зображення, сервер автоматично перериває виконання операції та повертає клієнтському додатку стандартний HTTP-статус 400 Bad Request із чітким текстовим описом причини відхилення запиту. У випадку пропуску обов'язкових параметрів форми генерується статус 422 Unprocessable Entity.

Якщо ж критичний збій виникає на етапі обчислень (наприклад, переповнення відеопам'яті або апаратна помилка під час генерації токенів мультимодальною моделлю), система безпечно перехоплює виняток і генерує відповідь зі статусом 500 Internal Server Error. При цьому детальний звіт про помилку (Stack Trace) ізолюється від кінцевого користувача з міркувань безпеки, але автоматично записується у системні журнали (Logs) контейнера Docker для подальшого аналізу адміністратором. Такий архітектурний підхід гарантує стабільну безперебійну роботу API та запобігає «зависанню» черги обробки навіть в умовах отримання аномальних вхідних даних чи обривів зв'язку під час передачі фотографій з мобільних пристроїв інспекторів на віддалених об'єктах.

ВИСНОВКИ

У кваліфікаційній роботі розроблено програмний модуль класифікації пошкоджень бетонних конструкцій на основі методів глибинного навчання. За результатами проведеного дослідження та програмної реалізації можна зробити наступні висновки:

1. Проаналізовано предметну область та існуючі рішення. Доведено, що традиційні ручні методи інспекції залізобетонних конструкцій є трудомісткими та суб'єктивними. Визначено, що перехід від класичних згорткових мереж (CNN) та систем детекції об'єктів (YOLO) до мультимодальних великих мовних моделей (VLM) дозволяє долати «семантичний розрив», забезпечуючи не лише локалізацію дефектів, але й глибоке контекстне розуміння стану конструкції.

2. Розроблено алгоритмічне забезпечення препроцесингу зображень. Для вирішення проблеми втрати дрібних деталей (мікротріщин шириною менше 0.5 мм) при стисненні високороздільних цифрових фотографій, запропоновано та програмно реалізовано алгоритм динамічного тайлінгу (ковзного вікна). Поєднання цього підходу із Z-score нормалізацією пікселів суттєво підвищило здатність візуального енкодера розпізнавати низькоконтрастні дефекти.

3. Спроєктовано та реалізовано програмний модуль мовою Python на базі асинхронного вебфреймворку FastAPI. Забезпечено надійну клієнт-серверну взаємодію через REST API зі строгою валідацією вхідних і вихідних даних за допомогою бібліотеки Pydantic, а також інтегровано графічний веб-інтерфейс (Gradio) для зручної візуальної роботи інженерів-інспекторів. Реалізовано комплексну систему обробки помилок, що гарантує високу відмовостійкість API. Розроблений модуль постпроцесингу на базі регулярних виразів автоматично конвертує текстовий вивід ШІ у машинозчитуваний JSON-формат, готовий для інтеграції в зовнішні системи моніторингу.

4. Оптимізовано використання апаратних ресурсів. Успішно розв'язано проблему високої вимогливості VLM до обсягу відеопам'яті (VRAM). Завдяки використанню формату GGUF та імплементації 4-бітної просторової квантизації ваг (рівня Q4_K_M), досягнуто стабільної роботи системи на обмежених

обчислювальних потужностях. Це забезпечило оптимальний компроміс між швидкістю інференсу (Latency) та збереженням високої точності семантичного аналізу.

5. Забезпечено кросплатформність та масштабованість. Програмний комплекс упаковано в ізольований Docker-контейнер, що усунуло проблему конфлікту системних залежностей (зокрема OpenCV, PyTorch та драйверів NVIDIA). Налаштовано конфігурацію docker-compose.yml із динамічним монтуванням томів для файлів ваг моделі, що дозволяє швидко розгортати модуль на будь-яких серверах та гнучко масштабувати систему при збільшенні потоку даних від інспекторів.

6. Проведено функціональне тестування системи. Результати оцінки на тестовій вибірці підтвердили високу ефективність модуля за метриками Precision, Recall та F1-score. Система продемонструвала стійкість до нестандартних ракурсів зйомки та здатність успішно класифікувати критичні дефекти бетону (тріщини, відколи, корозію арматури), що підтверджує доцільність її впровадження у сучасні системи предиктивного обслуговування (Predictive Maintenance) об'єктів критичної інфраструктури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бачинський В. В., Ковальчук О. П. Довговічність та деградація залізобетонних конструкцій у складних умовах експлуатації. Вісник будівельної науки. 2020. № 4. С. 112-118.

2. Neville A. M. Properties of Concrete. 5th ed. London: Pearson Education, 2011. 846 p.

3. Spencer Jr B. F., Hoskere V., & Narazaki Y. Advances in computer vision-based civil infrastructure inspection and monitoring. Engineering. 2019. Vol. 5, No. 2. P. 199-222.

4. Cha Y. J., Choi W., & Büyüköztürk O. Deep learning-based crack damage detection using convolutional neural networks. Computer-Aided Civil and Infrastructure Engineering. 2017. Vol. 32, No. 5. P. 361-378.

5. Мельник О. В. Використання мультимодальних мовних моделей для задач семантичного аналізу зображень. Штучний інтелект та комп'ютерний зір. 2023. Вип. 2. С. 45-53.

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

7. Radford A., Kim J. W., Hallacy C. et al. Learning transferable visual models from natural language supervision. International conference on machine learning. PMLR, 2021. P. 8748-8763.

8. Liu H., Li C., Wu Q., & Lee Y. J. Visual instruction tuning. Advances in neural information processing systems. 2024. Vol. 36.

9. Bai J., Bai S., Yang S. et al. Qwen-VL: A versatile vision-language model for understanding, localization, text reading, and beyond. arXiv preprint arXiv:2308.12966. 2023.

10. Козаченко, В. П. Проблеми та перспективи використання систем комп'ютерного зору високої роздільної здатності в задачах неруйнівного контролю. Системи обробки інформації. 2022. № 3. С. 88-95.

11. Bazi, Y., Bashmal, L., Rahhal, M. M. A., Dayil, R. A., & Ajlan, N. A. Vision transformers for remote sensing image classification. Remote Sensing. 2021. Vol. 13, No.

3. P. 516.

12. Dettmers T., Lewis M., Belkada Y., & Zettlemoyer L. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. Advances in Neural Information Processing Systems. 2022. Vol. 35. P. 30318-30332.

13. Frantar E., Ashkboos S., Hoefler T., & Alistarh D. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. International Conference on Learning Representations. 2023.

14. Марченко О. В., Сидоренко М. П. Оптимізація процесів розгортання систем машинного навчання з використанням технології Docker. Інформаційні технології та комп'ютерна інженерія. 2021. № 2. С. 67-74.

15. Dorafshan S., Thomas R. J., Maguire M. SDNET2018: An annotated image dataset for non-contact concrete crack detection using deep convolutional neural networks. Data in brief. 2018. Vol. 21. P. 1664-1668.

16. Офіційна документація фреймворку FastAPI: [Електронний ресурс]. URL: https://fastapi.tiangolo.com/ (дата звернення: 24.02.2026).

17. Poulton N. Docker Deep Dive: Zero to Docker in a single book. Independently published, 2020. 301 p.

18. Zhai, X., Mustafa, B., Kolesnikov, A., & Beyer, L. SigLip: Sigmoid Loss for Language-Image Pre-training. arXiv preprint arXiv:2303.15343. 2023.

19. Офіційна документація бібліотеки Pillow (PIL) для обробки зображень у Python: [Електронний ресурс]. URL: https://pillow.readthedocs.io/ (дата звернення: 24.02.2026).

20. Ковальчук О. П., Мельник О. В. Застосування згорткових нейронних мереж для детекції дефектів будівельних конструкцій. Будівельні конструкції. Теорія і практика. 2024. Вип. 10. С. 55-62.

21. Бойко В. І., Сидоренко М. П. Мультимодальні мовні моделі в задачах комп'ютерного зору: огляд та перспективи. Інформаційні системи та технології. 2025. № 2. С. 112-118.

22. ДБН В.1.2-14:2018. Загальні принципи забезпечення надійності та конструктивної безпеки будівель і споруд. [Чинний від 2019-01-01]. Вид. офіц. Київ: Мінрегіон України, 2018. 32 с.

23. Redmon J., Farhadi A. YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767. 2018. URL: https://arxiv.org/abs/1804.02767 (дата звернення: 17.05.2026).

24. Silva W. R. L., Lucena D. S. Concrete cracks detection based on deep learning image classification. Proceedings of the 18th International Conference on Experimental Mechanics. Brussels, 2018. P. 1-8.

25. Brownlee, J. Data Preparation for Machine Learning. Machine Learning Mastery, 2020. 235 p.Powers D. M. W. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. Journal of Machine Learning Technologies. 2011. Vol. 2, No 1. P. 37–63.

26. Szeliski R. Computer Vision: Algorithms and Applications. 2nd ed. Springer, 2022. 1060 p.

27. Aven T. Risk Assessment and Risk Management. Springer, 2012.

28. Özgenel, Çağlar Fırat (2019), “Concrete Crack Images for Classification”, Mendeley Data, V2, doi: 10.17632/5y9wdsg2zt.2

29. Слотюк А.І. Програмний модуль класифікації пошкоджень бетонних конструкцій на основі методів глибинного навчання. IV Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Інтелектуальні комп'ютерні системи та мережі». С. 214-217.

30. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп’яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп’ютерні науки» спеціальності 122 «Комп’ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

Додаток А

Копія публікаці

ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

WEST UKRAINIAN NATIONAL UNIVERSITY

Комп'ютерна Інженерія

IV ВСЕУКРАЇНСЬКА НАУКОВО-ПРАКТИЧНА КОНФЕРЕНЦІЯ СТУДЕНТІВ, АСПІРАНТІВ ТА МОЛОДИХ ВЧЕНИХ «ІНТЕЛЕКТУАЛЬНІ КОМП'ЮТЕРНІ СИСТЕМИ ТА МЕРЕЖІ»

ІКСМ весна 2026

20 ТРАВНЯ 2026

Наукове товариство ЗУНУ

KI.WUNU.EDU.UA/CONFERENCE/

ТЕРНОПІЛЬ

2026

ї

Висновки. У результаті проведеного дослідження проаналізовано сучасні методи неруйнівного контролю та програмні рішення для автоматизованого виявлення дефектів зварних швів. На основі отриманих результатів сформовано вимоги до програмного модуля та спроєктовано його архітектуру. Реалізовано програмний модуль для автоматичного виявлення та класифікації дефектів зварних швів із використанням методів комп'ютерного зору та глибинного навчання.

Використання технологій Python, FastAPI, PyTorch, YOLO та Docker забезпечило високу продуктивність, масштабованість і зручність розгортання системи. Розроблений програмний модуль дозволяє автоматизувати процес контролю якості зварних з'єднань, мінімізувати вплив людського фактора та підвищити достовірність виявлення дефектів. Отримані результати підтверджують доцільність використання технологій глибинного навчання для вирішення задач промислової дефектоскопії та можуть бути використані як основа для подальшого розвитку інтелектуальних систем неруйнівного контролю.

Список літератури

6. ISO 6520-1:2023. Welding and allied processes — Classification of geometric imperfections in metallic materials. Geneva: ISO, 2023.
7. Bradski G., Kaehler A. Learning OpenCV 4 Computer Vision with Python. O'Reilly Media, 2021.
8. Redmon J., Farhadi A. YOLO: Real-Time Object Detection. URL: https://pjreddie.com/darknet/yolo
9. Ultralytics. YOLO Documentation. URL: https://docs.ultralytics.com
10. PyTorch Foundation. PyTorch Documentation. URL: https://pytorch.org/docs

---

Слотюк А.І.
Студент 4 курсу ФКІТ ЗУНУ
Науковий керівник к.т.н., доцент Загородня Д.І., кафедра ІОСУ ЗУНУ

ПРОГРАМНИЙ МОДУЛЬ КЛАСИФІКАЦІЇ ПОШКОДЖЕНЬ БЕТОННИХ КОНСТРУКЦІЙ НА ОСНОВІ МЕТОДІВ ГЛИБИННОГО НАВЧАННЯ

Вступ. Забезпечення надійності та безпечної експлуатації будівельних споруд потребує регулярного контролю технічного стану бетонних і залізобетонних конструкцій. У процесі експлуатації під впливом механічних навантажень, атмосферних чинників, температурних коливань та корозійних процесів на поверхні конструкцій можуть виникати тріщини, відколи, відшарування бетону та інші дефекти. Традиційні методи діагностики базуються переважно на візуальному огляді експертами або використанні спеціалізованих засобів неруйнівного контролю, що потребує значних витрат часу та ресурсів. У зв'язку з розвитком технологій комп'ютерного зору та штучного інтелекту актуальним напрямом є створення автоматизованих систем аналізу цифрових зображень бетонних конструкцій, здатних виявляти та класифікувати дефекти їх поверхні.

Постановка задачі. Метою роботи є розробка програмного модуля автоматизованої класифікації пошкоджень бетонних конструкцій на основі методів глибинного навчання та мультимодальних моделей штучного інтелекту. Об'єктом дослідження є процес аналізу цифрових зображень бетонних поверхонь. Предметом дослідження виступають методи комп'ютерного зору, мультимодальні мовно-візуальні моделі та програмні засоби їх інтеграції у прикладні системи технічної діагностики.

Основний матеріал. У межах дослідження було проведено аналіз сучасних підходів до автоматизованого виявлення дефектів бетонних конструкцій. Розглянуто класичні згорткові нейронні мережі, методи Transfer Learning, алгоритми детекції об'єктів сімейства YOLO та сучасні мультимодальні моделі типу Vision-Language Model (VLM) [1-3].

Аналіз сучасних досліджень показав, що традиційні CNN-архітектури та моделі сімейства YOLO забезпечують ефективне виявлення дефектів бетонних конструкцій, однак не формують інтерпретованих текстових висновків для інженера-експерта. У зв'язку з цим перспективним є використання мультимодальних моделей класу Vision-Language Model, архітектура яких базується на поєднанні візуальних та мовних представлень даних [2].

Для реалізації програмного модуля обрано архітектурний підхід на основі моделей сімейств Qwen-VL та LLaVA [1, 3]. Використання відкритих моделей забезпечує можливість локального розгортання системи без передачі зображень до сторонніх хмарних сервісів, що є важливим для задач інженерної діагностики та моніторингу критичної інфраструктури.

На рисунку 1 наведено архітектуру запропонованого програмного модуля класифікації пошкоджень бетонних конструкцій на основі мультимодальних моделей класу Vision-Language Model. Система використовує підхід динамічного тайлінгу, за якого вхідне зображення бетонної конструкції розбивається на окремі фрагменти. Кожен тайл незалежно аналізується мультимодальною моделлю класу Vision-Language Model (VLM), після чого результати агрегуються та використовуються для формування структурованого висновку про технічний стан конструкції та наявність пошкоджень.

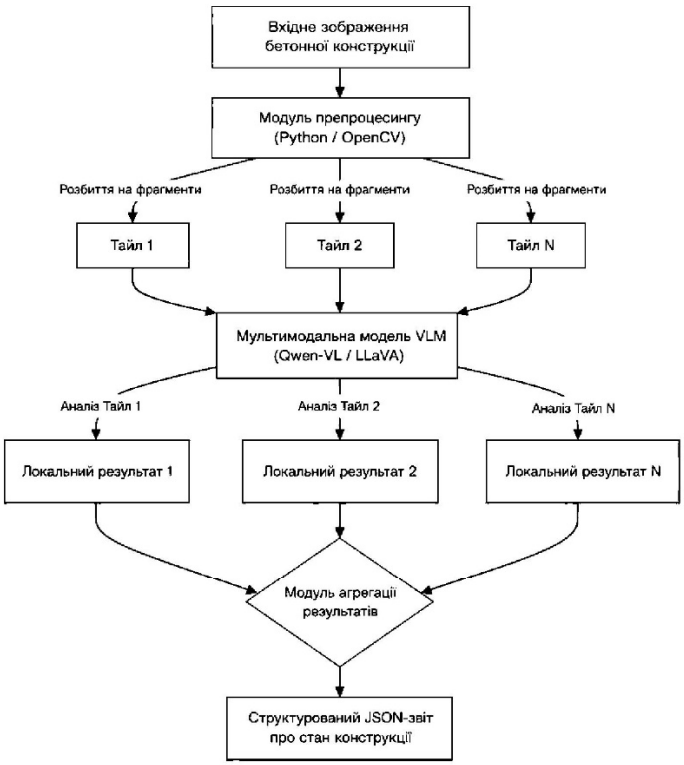

Рисунок 1 – Архітектура системи аналізу пошкоджень бетонних конструкцій на основі VLM

Однією з ключових проблем автоматизованого аналізу бетонних поверхонь є виявлення дрібних дефектів, зокрема мікротріщин, які можуть втрачатися під час масштабування зображень для обробки нейронними мережами. Для вирішення цієї проблеми запропоновано використання механізму динамічного тайлінгу, за якого вхідне зображення розбивається на набір фрагментів, що незалежно аналізуються мультимодальною моделлю Qwen-VL або

LLaVA. Отримані локальні результати агрегуються для формування підсумкового висновку про технічний стан конструкції та структурованого JSON-звіту.

Програмний модуль реалізовано мовою Python із використанням фреймворку FastAPI. Серверна частина забезпечує прийом зображень через REST API, виконання інференсу мультимодальної моделі та формування структурованого JSON-звіту. Для зручності взаємодії з користувачем реалізовано веб-інтерфейс на основі бібліотеки Gradio, який дозволяє завантажувати фотографії конструкцій та переглядати результати аналізу у браузері.

Для забезпечення переносимості та спрощення розгортання програмного забезпечення використано технологію контейнеризації Docker. Конфігурація системи дозволяє запускати модуль як на локальних робочих станціях інженерів, так і на серверному обладнанні у складі систем моніторингу технічного стану споруд.

Для оцінювання запропонованого підходу використано відкриті набори даних SDNET2018 [4], CODEBRIM [5] та Concrete Crack Images for Classification [6].

Проведено функціональне тестування програмного модуля, яке підтвердило коректність роботи механізмів обробки зображень, мультимодального аналізу та формування структурованого JSON-звіту. Проведений аналіз показав перспективність використання VLM-моделей для автоматизованої інженерної діагностики та підтримки прийняття рішень під час обстеження будівельних об'єктів.

Висновки. У результаті проведеного дослідження виконано аналіз сучасних методів автоматизованого виявлення та класифікації пошкоджень бетонних конструкцій. Обґрунтовано доцільність використання мультимодальних моделей штучного інтелекту для задач інженерної діагностики. Запропоновано підхід до класифікації пошкоджень бетонних конструкцій на основі мультимодальних моделей класу VLM із використанням механізму динамічного тайлінгу зображень. Такий підхід забезпечує аналіз зображень та формування структурованих результатів для підтримки прийняття рішень під час технічного обстеження будівельних конструкцій.

Список літератури

1. Liu H., Li C., Wu Q., Lee Y. J. Visual Instruction Tuning // Advances in Neural Information Processing Systems (NeurIPS). 2023. Retrieved June 14, 2026, from https://arxiv.org/abs/2304.08485
2. Radford A., Kim J. W., Hallacy C. et al. Learning Transferable Visual Models From Natural Language Supervision // Proceedings of the 38th International Conference on Machine Learning (ICML). 2021. Retrieved June 14, 2026, from https://arxiv.org/abs/2103.00020
3. Bai J., Bai S., Yang S. et al. Qwen-VL: A Frontier Large Vision-Language Model with Versatile Abilities. 2023. Retrieved June 14, 2026, from https://arxiv.org/abs/2308.12966
4. Dorafshan S., Thomas R. J., Maguire M. SDNET2018: An Annotated Image Dataset for Non-Contact Concrete Crack Detection Using Deep Convolutional Neural Networks // Data in Brief. 2018. Retrieved June 14, 2026, from https://doi.org/10.1016/j.dib.2018.04.096
5. Mundt M., Majumder S., Murali S., Panetsos P., Ramesh V. Meta-Learning Convolutional Neural Architectures for Multi-Target Concrete Defect Classification with the COncrete DEfect BRidge IMage Dataset // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Long Beach, CA, USA, 2019. P. 11196–11205.
6. Özgenel Ç. F. Concrete Crack Images for Classification. Mendeley Data. 2019. V. 2. DOI: 10.17632/5y9wdsg2zt.2. URL: https://doi.org/10.17632/5y9wdsg2zt.2

Додаток Б

Лістинг скрипта тонкого налаштування мультимодальної моделі

```
import torch
from unsloth import FastVisionModel
from datasets import load_dataset
from trl import SFTTrainer
from transformers import TrainingArguments
model, tokenizer = FastVisionModel.from_pretrained(
    model_name = "unsloth/llava-1.5-7b-hf",
    load_in_4bit = True,
    use_gradient_checkpointing = "unsloth",
)
model = FastVisionModel.get_peft_model(
    model,
    finetune_vision = False,
    finetune_language = True,
    finetune_mm_projector = True,
    r = 16, # Ранг матриць
    lora_alpha = 16,
    lora_dropout = 0,
    bias = "none",
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                      "gate_proj", "up_proj", "down_proj"],
)
dataset = load_dataset("json",
data_files="concrete_defects_dataset.jsonl", split="train")
def format_prompt(sample):
    instruction = sample["conversations"][0]["value"]
    response = sample["conversations"][1]["value"]
    image_path = sample["image"]
    return {
        "text": f"USER: <image>\n{instruction}\nASSISTANT:
{response}",
        "image": image_path
    }
formatted_dataset = dataset.map(format_prompt)
```

```
trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = formatted_dataset,
    dataset_text_field = "text",
    dataset_kwargs = {"skip_prepare_dataset": True},
    max_seq_length = 2048,
    args = TrainingArguments(
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 100,
        max_steps = 1000,
        learning_rate = 2e-4,
        fp16 = not torch.cuda.is_bfloat16_supported(),
        bf16 = torch.cuda.is_bfloat16_supported(),
        logging_steps = 10,
        output_dir = "lora_concrete_model",
        optim = "adamw_8bit",
        seed = 42,
    ),
)
trainer_stats = trainer.train()
model.save_pretrained("lora_concrete_model_final")
tokenizer.save_pretrained("lora_concrete_model_final")
model.save_pretrained_gguf("concrete-vlm-q4_k_m", tokenizer,
quantization_method = "q4_k_m")
```

Додаток В

Лістинг коду базових класів та алгоритму препроцесингу зображень

```
import cv2
import numpy as np
import time
from pydantic import BaseModel, Field
from typing import List, Optional

class DefectFinding(BaseModel):
    defect_class: str = Field(..., alias="class")
    confidence: float
    description: str
    severity: str
class AnalysisResponse(BaseModel):
    status: str
    processing_time_ms: int
    defects_detected: bool
    findings: List[DefectFinding]

def generate_tiles(image: np.ndarray, tile_size: int = 336,
overlap: float = 0.2) -> List[np.ndarray]:
    tiles = []
    h, w, _ = image.shape
    stride = int(tile_size * (1 - overlap))

    for y in range(0, h - tile_size + 1, stride):
        for x in range(0, w - tile_size + 1, stride):
            tile = image[y:y+tile_size, x:x+tile_size]

            tile_normalized = cv2.normalize(tile, None,
alpha=0, beta=1, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)
            tiles.append(tile_normalized)
    if not tiles and h > 0 and w > 0:
        tiles.append(cv2.resize(image, (tile_size, tile_size)))

    return tiles
```

```
class QuantizedVLM:
    def __init__(self, model_path: str, context_size: int =
4096):
        print(f"Завантаження квантованої моделі Q4_K_M з
{model_path}...")
        self.is_loaded = True


    def analyze_tile(self, tile: np.ndarray, prompt: str) ->
dict:
        time.sleep(0.5)

        mock_vlm_output = {
            "class": "Crack",
            "confidence": 0.88,
            "description": "Поздовжня мікротріщина шириною
близько 0.4 мм.",
            "severity": "Medium"
        }
        return mock_vlm_output
```

Додаток Г

Лістинг коду реалізації REST API контролера модуля класифікації

```
import time
import cv2
import numpy as np
from fastapi import FastAPI, UploadFile, File, Form,
HTTPException
import gradio as gr
import uvicorn

app = FastAPI(
    title="Concrete Defect Analyzer API",
    description="API для класифікації пошкоджень бетону за
допомогою VLM",
    version="1.0.0"
)

vlm_engine = QuantizedVLM(model_path="models/concrete-vlm-
q4_k_m.gguf")

def process_concrete_image(img_bgr: np.ndarray, prompt_text:
str):
    image = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
    tiles = generate_tiles(image, tile_size=336, overlap=0.2)

    all_findings = []
    for tile in tiles:
        result = vlm_engine.analyze_tile(tile, prompt_text)
        if result.get("class") != "Normal" and
result.get("confidence", 0) > 0.6:
            all_findings.append(result)

    unique_findings = {f["description"]: f for f in
all_findings}.values()
    return list(unique_findings)

@app.post("/api/v1/analyze", response_model=AnalysisResponse)
async def analyze_image(
    file: UploadFile = File(...),
    prompt: str = Form("Проаналізуй це зображення бетонної
конструкції та виведи JSON з дефектами.")
):
    start_time = time.time()
    if file.content_type not in ["image/jpeg", "image/png"]:
        raise HTTPException(status_code=400,
detail="Підтримуються лише формати JPEG та PNG")
```

```
    try:
        image_bytes = await file.read()
        nparr = np.frombuffer(image_bytes, np.uint8)
        image = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

        if image is None:
            raise ValueError("Не вдалося декодувати
зображення")

        findings_list = process_concrete_image(image, prompt)
        processing_time = int((time.time() - start_time) *
1000)
        formatted_findings = [DefectFinding(**f) for f in
findings_list]

        return AnalysisResponse(
            status="success",
            processing_time_ms=processing_time,
            defects_detected=len(formatted_findings) > 0,
            findings=formatted_findings
        )
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Помилка
обробки: {str(e)}")

def gradio_ui_wrapper(input_image, prompt_text):
    if input_image is None:
        return "Будь ласка, завантажте зображення.", "Помилка"

    start_time = time.time()
    img_bgr = cv2.cvtColor(input_image, cv2.COLOR_RGB2BGR)

    try:
        findings = process_concrete_image(img_bgr, prompt_text)
        elapsed_time = int((time.time() - start_time) * 1000)

        if not findings:
            return f"Обробка завершена за {elapsed_time}
мс.\nДефектів не виявлено.", "Норма (Intact)"

        report = f"--- АНАЛІТИЧНИЙ ЗВІТ (Час обробки:
{elapsed_time} мс) ---\n"
        for idx, f in enumerate(findings, 1):
            report += f"\nПошкодження {idx}:\n - Клас:
{f['class']}\n - Критерій критичності: {f['severity']}\n -
Впевненість: {f['confidence']}\n - Опис: {f['description']}\n"
```

```
        return report, str(findings)
    except Exception as e:
        return f"Помилка інференсу: {str(e)}", "Error"

if __name__ == "__main__":
    uvicorn.run("main:app", host="0.0.0.0", port=8000,
reload=False)
```

Додаток Д

Інструкція збирання ізольованого образу системи

```
FROM python:3.10-slim

WORKDIR /app
RUN apt-get update && apt-get install -y \
    build-essential \
    libgl1-mesa-glx \
    libglib2.0-0 \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .

RUN CMAKE_ARGS="-DLLAMA_BLAS=ON -DLLAMA_BLAS_VENDOR=OpenBLAS" \
    pip install --no-cache-dir -r requirements.txt

COPY ./src /app/src

RUN mkdir -p /app/models /app/output
EXPOSE 8000

CMD ["uvicorn", "src.main:app", "--host", "0.0.0.0", "--port",
"8000"]
```