НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Програмне забезпечення для кодування зображень з використанням глибинних нейронних мереж

Виконав студент IV курсу, групи ІП-13
(шифр групи)

Музичук Віталій Андрійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент кафедри ІПІ, к.т.н., доцент, Олійник Ю. О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант ст. вик., Вітковська І. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІСТ, к.ф.-м.н., доцент, Рибачук Л. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Музичуку Віталію Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Програмне забезпечення для кодування зображень з використанням глибинних нейронних мереж

керівник проєкту Олійник Юрій Олександрович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «23» травня 2025 р. №1705-с

2. Термін подання студентом проєкту «16» червня 2025 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Передпроєктне обстеження предметної області: постановка завдання дипломного проєктування, аналіз предметної області, аналіз існуючих рішень, опис бізнес-процесів.
2) Розроблення вимог до програмного забезпечення: опис варіантів використання, розбролення функціональних і нефункціональних вимог, аналіз системних вимог, аналіз економічних показників програмного забезпечення, постановка завдання на розробку
3) Конструювання та розроблення програмного забезпечення: архітектура програмного забезпечення, обґрунтування вибору засобів розробки, конструювання програмного забезпечення, аналіз безпеки даних.
4) Аналіз якості та тестування програмного забезпечення: аналіз якості ПЗ, опис процесів тестування, опис контрольного прикладу.
5) Розгортання та супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань _____

2) Схема структурна компонентів _____

3) Схема структурна послідовності _____

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «15» березня 2025 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	15.03.2025	
2	Аналіз існуючих методів розв’язання задачі	22.03.2025	
3	Постановка та формалізація задачі	29.03.2025	
4	Розробка інформаційного забезпечення	06.04.2025	
5	Алгоритмізація задачі	13.04.2025	
6	Обґрунтування вибору використаних технічних засобів	20.04.2025	
7	Розробка програмного забезпечення	29.04.2025	
8	Налагодження програми	06.05.2025	
9	Виконання графічних документів	16.05.2025	
10	Оформлення пояснювальної записки	24.05.2025	
11	Подання ДП на попередній захист	02.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

Студент _____ Віталій МУЗИЧУК

(підпис) (ініціали, прізвище)

Керівник _____ Юрій ОЛІЙНИК

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 28 таблиць, 32 рисунків та 15 джерел – загалом 76 сторінки.

Дипломний проєкт присвячений розробці програмного забезпечення з кодування зображень з використанням глибинних нейронних мереж.

Метою дипломного проєкту є удосконалення алгоритму кодування растрових зображень на основі подібності фрагментів шляхом скорочення часу обробки та розширення його функціональних можливостей, зокрема шляхом впровадження механізмів відновлення втрачених фрагментів зображення.

У розділі передпроєктне обстеження предметної області було постановлено завдання дипломного проєктування надані основні визначення та терміни, описано предметну область, проаналізовано відомі алгоритмічні рішення, описано основні бізнес-процеси.

Розділ розроблення вимог до програмного забезпечення призначений формулюванню варіантів використання, функціональних та нефункціональних вимог, аналізу системних вимог до розроблюваного програмного забезпечення, аналізу економічних показників програмного забезпечення й виконано постановку задачі.

У розділі конструювання та розроблення програмного забезпечення описано архітектуру та конструювання програмного забезпечення, обґрунтовано вибір засобів розробки, виконано аналіз безпеки даних.

Розділ аналізу якості та тестування програмного забезпечення присвячений метрикам якості, опису процесів тестування та контрольного прикладу програмного забезпечення.

Розділ розгортання та супроводу має опис процесів розгортання, публікації та супроводу розробленого програмного забезпечення.

КЛЮЧОВІ СЛОВА: КОДУВАННЯ ЗОБРАЖЕНЬ, ВИЗНАЧЕННЯ ПОДІБНОСТІ, GOOGLE CLOUD, PYTHON, TENSORFLOW, ВЕКТОРНЕ ДЕРЕВО, SSIM.

ABSTRACT

The explanatory note of the diploma project consists of five sections, includes 28 tables, 32 images, and 15 sources — in total 76 pages.

The diploma project is dedicated to the development of software for image encoding using deep neural networks.

The objective of this diploma project is to improve the raster image encoding algorithm based on fragment similarity by reducing processing time and extending its functionality, in particular by introducing mechanisms for restoring lost image fragments.

The Pre-project analysis of the subject area section provides key definitions and terms, describes the subject area, analyzes known algorithmic solutions, and outlines the main business processes.

The Software requirements development section is focused on the formulation of use cases, functional and non-functional requirements, analysis of system requirements for the developed software, economic indicators assessment, and task formulation.

The Software design and development section describes the architecture and construction of the software, justifies the choice of development tools, and includes data security analysis.

The Software quality analysis and testing section is dedicated to quality metrics, the description of testing processes, and a test case of the developed software.

The Deployment and maintenance section outlines the processes of deployment, publication, and maintenance of the developed software.

KEYWORDS: IMAGE ENCODING, SIMILARITY DETECTION, GOOGLE CLOUD, PYTHON, TENSORFLOW, VECTOR TREE, SSIM.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-1322.045490.01.91	Технічне завдання	14	
3	А4	КПІ.ІП-1322.045490.02.81	Пояснювальна записка	76	
4	А4	КПІ.ІП-1322.045490.03.12	Текст програми	28	
5	А4	КПІ.ІП-1322.045490.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-1322.045490.05.34	Керівництво користувача	13	
7	А3	КПІ.ІП-1322.045490.06.99	Графічний матеріал	3	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1322.045490.00.90		
Розроб.		Музичук В. А.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Олійник Ю. О.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-13	
Н.контр.		Вітковська І. І.					
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2025 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ КОДУВАННЯ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ГЛИБИННИХ НЕЙРОННИХ МЕРЕЖ

Технічне завдання

КПІ.ІП-1322.045490.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Юрій ОЛІЙНИК

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Віталій МУЗИЧУК

Київ – 2025

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмне забезпечення для кодування зображень з використанням глибинних нейронних мереж.

Галузь застосування: обробка зображень.

Наведене технічне завдання поширюється на розробку “Програмне забезпечення для кодування зображень з використанням глибинних нейронних мереж” [045490], котре використовується для кодування растрових зображень шляхом поділу їх на фрагменти, виділення векторів ознак й побудови векторного дерева пошуку та призначена для зменшення кількості даних необхідних для збереження растрових зображень, забезпечення захищеної передачі через канали зв’язку та можливості доповнення зображення.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки програмного забезпечення для кодування зображень з використанням глибинних нейронних мереж є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для скорочення обсягу пам’яті, потрібної для збереження зображень, а також підвищення ефективності й забезпечення захисту їх передавання через мережеві канали.

Метою розробки є вдосконалення методу кодування растрових зображень на основі подібності фрагментів шляхом впровадження механізмів відновлення втрачених фрагментів зображення.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– завантаження зображення для додання фрагментів у базу (рисунок 4.1);

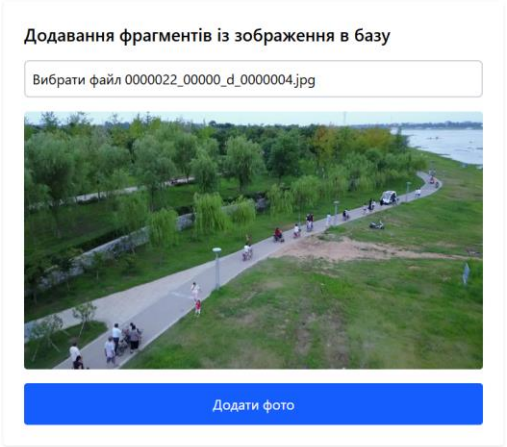


Рисунок 4.1 – Прототип сторінки для завантаження зображення у базу

– завантаження зображення для кодування (рисунок 4.2);

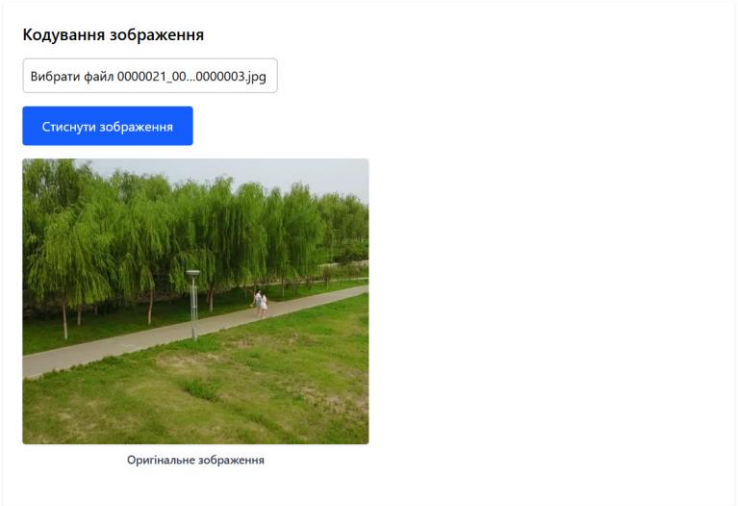


Рисунок 4.2 – Прототип сторінки кодування зображення

– декодування зображення з бінарного файлу (рисунок 4.3);

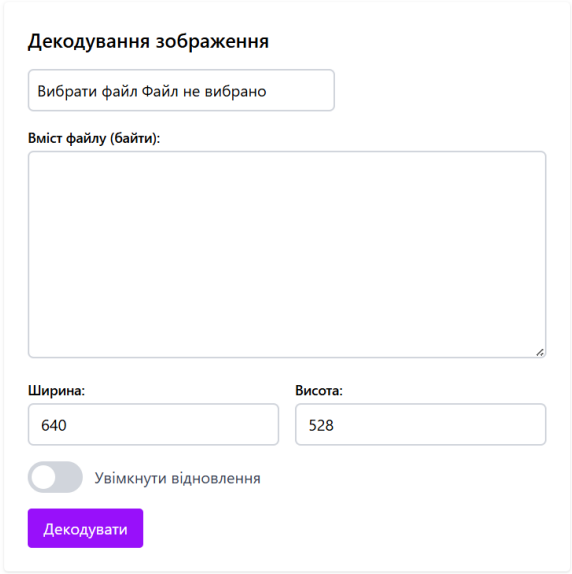


Рисунок 4.3 – Прототип сторінки декодування зображення

– отримання індексу схожості між двома зображеннями (рисунок 4.4);

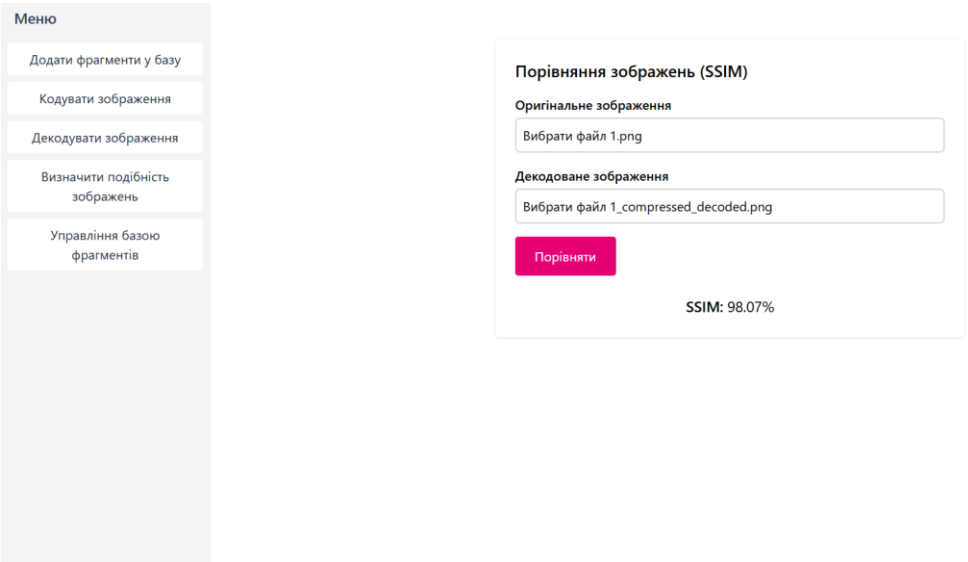


Рисунок 4.4 – Прототип сторінки індексу схожості двох зображень

– збереження поточної бази фрагментів (рисунок 4.5);

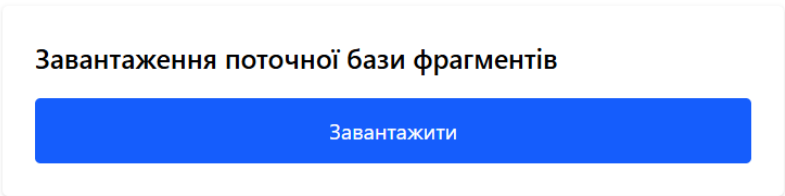


Рисунок 4.5 – Прототип сторінки завантаження бази фрагментів із сервера

– завантаження власної бази фрагментів із сервера (рисунок 4.6).

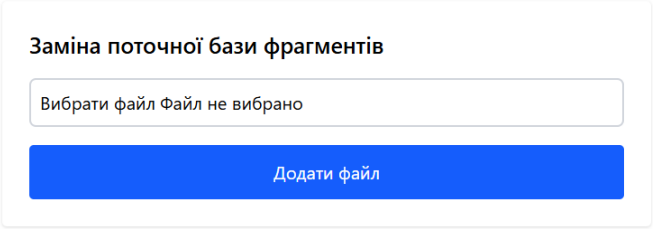


Рисунок 4.6 – Прототип сторінки завантаження власної бази фрагментів

4.1.2 Для користувача

– завантаження й збереження файлу у стандартизованому форматі поточної бази фрагментів на власному носії;

– здійснення декодування й можливість завантаження бінарного файлу стиснутого зображення;

– отримання статистики щодо виконання операції стиснення: відсоток стиснення, подібність декодованого зображення й час виконання;

– надсилання стандартизованої бази фрагментів для оновлення її на сервері.

4.1.3 Для серверу

– сформувати файл з інформацією про поточні фрагменти у базі;

– оновлювати базу фрагментів згідно надісланих файлів користувача.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесора: архітектура x86-64 або ARM64, щонайменше 32 потоки обробки;

– об’єм ОЗП: 8 ГБ;

– графічний процесор (GPU): сумісний з CUDA, рекомендовано NVIDIA Tesla T4;

– швидкість підключення до мережі Інтернет: від 20 Мбіт/с;

– програмне забезпечення: встановлений CUDA Toolkit, сумісний із відповідною версією TensorFlow.

Рекомендована конфігурація технічних:

– тип процесора: архітектура x86-64 або ARM64, щонайменше 64 потоки обробки;

– об’єм ОЗП: 32 ГБ;

– графічний процесор (GPU): сумісний з CUDA, рекомендовано NVIDIA L4 або вище;

– швидкість підключення до мережі Інтернет: від 100 Мбіт/с;

– програмне забезпечення: встановлений CUDA Toolkit, сумісний із відповідною версією TensorFlow.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN32 (Windows 10, 11) або Unix (MacOS, Linux).

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі: растрове зображення формату JPEG, PNG.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі: набір байт певної структури.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Python.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі PyCharm.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді у вигляді посилання на репозиторій на платформі GitHub, а також у додатках до пояснювальної записки.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема структурна компонентів програмного забезпечення;
– архітектура програмного забезпечення;
– схема структурна діаграми послідовності.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проєкту	15.03	
2.	Розробка технічного завдання	29.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	12.04	Специфікації програмного забезпечення
4.	Проєктування структури програмного забезпечення, проєктування компонентів	19.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	20.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	25.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проєкту	27.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проєкту	01.06	Графічний матеріал проєкту
9.	Оформлення технічної документації проєкту	04.06	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: **Програмне забезпечення для кодування зображень з використанням глибинних нейронних мереж**

КПІ.ІП-1322.045490.02.81

Київ – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
DCT	–	Discrete Cosine Transform – алгоритм перетворення сигналу.
RL	–	Run-Length Encoding – метод стиснення даних.
SSIM	–	Structure Similarity – індекс структурної подібності.
JPEG	–	Joint Photographic Experts Group – метод стиснення зображень.
CNN	–	Convolutional neural network – згорткова нейронна мережа.
API	–	Application programming interface, прикладний програмний Інтерфейс.
GCP	–	Google Cloud Platform – провайдер хмарної інфраструктури.
GPU	–	Graphics Processing Unit – графічний процесор
ER	–	Entity-Relation diagram.
БПЛА	–	Безпілотний літальний апарат

ВСТУП

У сучасному цифровому середовищі значні обсяги візуальних даних створюють потребу в ефективних методах їх зберігання, передачі та обробки. Особливо актуальним стає завдання кодування зображень із можливістю зменшення обсягу даних без втрати важливої інформації, а також забезпечення швидкого доступу до окремих фрагментів.

Провідні наукові установи та компанії, зокрема Google AI та MIT, активно досліджують застосування глибинного навчання для покращення методів стискання і реконструкції зображень. Сучасні підходи включають використання нейронних мереж для аналізу структури зображення, виділення ключових ознак та пошуку подібних фрагментів.

Метою даного дипломного проєкту є удосконалення алгоритму кодування растрових зображень на основі подібності фрагментів шляхом скорочення часу обробки та розширення його функціональних можливостей, зокрема шляхом впровадження механізмів відновлення втрачених фрагментів зображення. Запропоноване рішення спрямоване на підвищення ефективності зберігання зображень та може бути застосоване в БПЛА системах, дронах, архівах, системах моніторингу тощо.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання дипломного проєктування

У рамках дипломного проєктування передбачається комплексне виконання низки завдань спрямованих на розробку програмного забезпечення для кодування зображень із використанням глибинних нейронних мереж. Для реалізації даного завдання необхідно виконати:

– аналіз предметної області та визначення загального завдання розробки у рамках ДП;

– аналіз наявних алгоритмічних чи технічних рішень щодо кодування й стиснення растрових зображень;

– аналіз та моделювання бізнес-процесів;

– розроблення функціональних, нефункціональних та системних вимог до програмного забезпечення;

– постановка завдання на розробку програмного забезпечення ДП;

– покращення алгоритмічної частини методу кодування зображень;

– імплементація розробленого алгоритмічного рішення та обґрунтування вибору засобів розробки програмного забезпечення;

– розробка прототипу програмного забезпечення для кодування зображень;

– аналіз якості та тестування програмного забезпечення;

– створення супроводжувальної документації до розробленого програмного забезпечення.

1.2 Аналіз предметної області

Зображення є одним із найдавніших засобів збереження та передачі візуальної інформації. Людське око сприймає зображення як сукупність світлових сигналів, що мають різну довжину хвилі та інтенсивність, формуючи цілісну картину навколишнього світу. З розвитком цифрових технологій виникла потреба у представленні зображень у формі, придатній для

обробки комп’ютером. У сучасному світі, де цифрова трансформація охоплює практично всі сфери діяльності, стало можливим кодувати та зберігати зображення в електронному вигляді. Існують два основні способи цифрового представлення зображень — растровий і векторний [1].

Растрові зображення є найпоширенішим типом цифрової графіки, який використовується для передачі складних візуальних об’єктів таких як фотографії, скановані зображення та ілюстрації, створені в графічних редакторах. Растрове зображення задається сіткою з пікселів, де кожен елемент має свій колір. Щільність розміщення визначає роздільну здатність, що вимірюється в dpi (dots per inch). Недоліком растрової графіки є погана масштабованість та присутність дефектів й артефактів при збільшенні зображення. Серед популярних форматів растрової графіки — JPEG, PNG, BMP, TIFF і GIF.

Векторні зображення формуються не з пікселів, а з геометричних примітивів — ліній, кривих, багатокутників, що описуються математичними формулами. Завдяки цьому вони не втрачають якості при масштабуванні, що робить їх ідеальними для створення логотипів, схем, іконок та технічних ілюстрацій. Векторна графіка забезпечує високу точність відображення форм та контурів, проте не підходить для реалістичного відтворення фотографій. Найпоширеніші формати — SVG, EPS, AI, PDF, а для редагування використовуються такі програми, як Adobe Illustrator, CorelDRAW та Inkscape.

Відповідно до постановки завдання до дипломного проєктування зосередимо увагу більш детально саме на растрових зображеннях, оскільки вони і є предметом дослідження даної роботи.

У сучасну епоху великих даних проблема обсягу графічної інформації стає дедалі актуальнішою. Через обмеження у пропускній здатності мереж та обсягах пам’яті виникає потреба у стисненні даних для зменшення розміру файлів, які потрібно зберігати чи передавати. Особливо це стосується растрових зображень, у яких зберігається інформація про кожен окремий

піксель. У необробленому вигляді такі файли мають великі розміри, що ускладнює їх збереження та пересилання [2].

Стиснення зображень — це процес зменшення обсягу графічного файлу без суттєвої втрати якості або з допустимим її зниженням. Основна мета кодування полягає у виявленні та видаленні надмірної або другорядної інформації, залишаючи тільки необхідну для якісного відтворення зображення.

У цифрових зображеннях існує три основні типи надмірностей [3]:

– психовізуальна надмірність — базується на обмеженнях людського зору, який менш чутливий до деяких деталей, кольорів або частот;

– міжпіксельна надмірність — відображає кореляцію між сусідніми пікселями, особливо в однорідних областях зображення;

– кодувальна надмірність — виникає при використанні фіксованої довжини коду для пікселів, що не завжди ефективно.

Однак традиційні методи стиснення зазвичай працюють лише в межах одного зображення, ігноруючи можливі схожості між різними файлами. Це обмеження призводить до повторного кодування схожих фрагментів у багатьох зображеннях. Сучасні підходи, зокрема на основі глибинного навчання та згорткових нейронних мереж (CNN), дозволяють виявляти та перевикористовувати спільні ознаки між зображеннями. Це відкриває нові можливості для підвищення ефективності стиснення і зменшення зайвих витрат ресурсів у цифрових системах, які й будуть досліджені в даній роботі.

1.3 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації програмного забезпечення для кодування растрових зображень з використання глибинних нейронних мереж. Далі будуть розглянуті вже існуючи на даний момент алгоритмічні рішення й підходи до проблеми кодування зображень.

1.3.1 Аналіз відомих програмних продуктів

Варто зазначити, що окремих програмних продуктів, призначених виключно для стиснення зображень, існує небагато. Це пояснюється тим, що більшість сучасних алгоритмів кодування та декодування графічних даних уже інтегровані в основне програмне забезпечення, яке працює із зображеннями — зокрема, у графічні редактори, фоторедактори та вебплатформи.

Проте можна виділити окремі спеціалізовані онлайн-сервіси, що використовують штучний інтелект, зокрема нейронні мережі, для ефективного стиснення зображень.

Nero AI [4] — це вебсервіс для обробки растрових зображень, одним з функціональних модулів якого є стиснення графічних файлів (форматів JPG, PNG, WebP, SVG). Його ключовими можливостями є пакетне стиснення декількох зображень одночасно, що дозволяє економити час при обробці великих обсягів даних, швидка обробка зображень, а також доступ до API для інтеграції у сторонні сервіси. Сервіс працює за платною моделлю, але пропонує безкоштовний пробний період. Водночас він не підтримує формат .bmp (зображення автоматично конвертуються в JPEG) і не дозволяє регулювати рівень стиснення вручну. На рисунку 1.1 показано приклад результату стиснення: при значному збільшенні помітні артефакти JPEG-кодування.

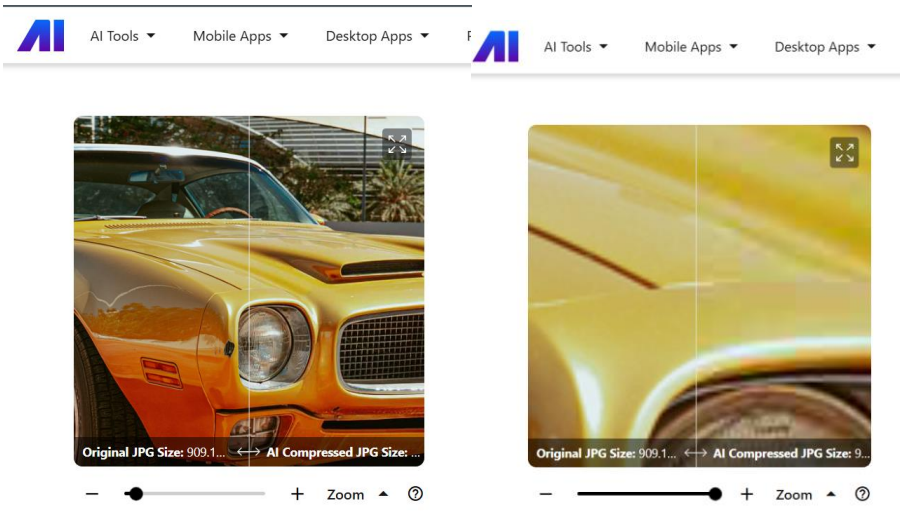


Рисунок 1.1 – Демонстрація стиснення на сайті Nero AI

VanceAI [5] — ще один онлайн-сервіс, який позиціонує себе як комплексне рішення для обробки зображень. Він орієнтований на дизайнерів, маркетологів та розробників. Його компонент **VanceAI Image Compressor** має схожий функціонал до Nero AI, але також включає декілька унікальних особливостей. Сервіс обіцяє до 80% зменшення розміру зображення без істотної втрати якості, підтримує пакетну обробку (до 20 файлів одночасно), надає API та гарантує конфіденційність — всі зображення автоматично видаляються із серверів через 24 години. Безкоштовний функціонал доступний через графічний інтерфейс, однак для комерційного використання та доступу до API необхідна платна підписка. Приклад стиснення зображення з VanceAI зображений на рисунку 1.2.

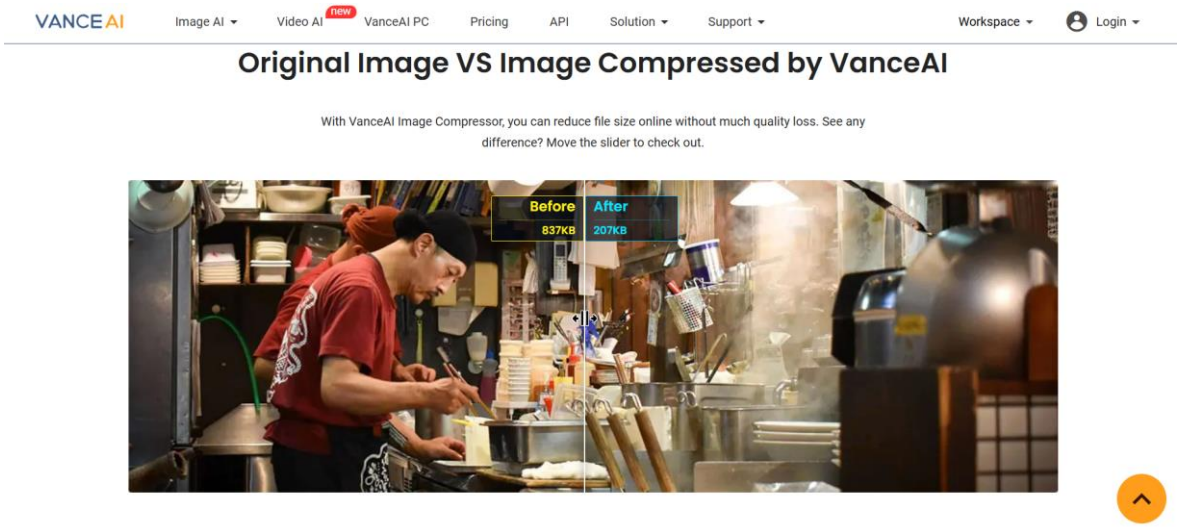


Рисунок 1.2 – Демонстрація стиснення фото на сервісі VanceAI

Ключова відмінність наступних описаних сервісів від запропонованої розробки у дипломному проєкті полягає у тому, що вони націлені зменшити розмір зображення й зберегти його в стандартному кодуванні (JPEG, JPG), що сильно обмежує відсоток стиснення та зменшує якість зображення після компресії. Для порівняння проєкту з аналогом можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння аналогів

Функціонал	Дипломний проєкт	NeroAI	VanceAI
Відсоток стиснення зображення	~98%	Динамічно від 20% до 80%	До 80%
Подібність зображень за SSIM	~97%	93-98%	92-96%
Ціна продукту	Open-source продукт	Демо версія + платна підписка	Безкоштовний обмежений функціонал + пакетні підписки
Доступне API	Відкритий код дає можливість розгорнути API самостійно для себе	Сервіс надає готовий доступ до API	Сервіс надає готовий доступ до API
Можливість обробки декількох фотографій одночасно	-	+	+

Продовження таблиці 1.1

Функціонал	Дипломний проєкт	NeroAI	VanceAI
Збереження вихідного зображення у BMP форматі	+ (Можна перетворити закодовоне зображення у BMP формат)	-	-

1.3.2 Аналіз відомих алгоритмічних та технічних рішень

Протягом останніх десятиліть було створено безліч алгоритмів стиснення, які можна умовно поділити на два типи: стиснення з втратами (lossy) та стиснення без втрат (lossless). Вибір між ними залежить від конкретних вимог до якості та цілісності даних у певному застосуванні [3].

Стиснення без втрат зберігає всі дані оригінального зображення, забезпечуючи повну його цілісність. Такий підхід зазвичай застосовується для важливих файлів, де навіть незначна втрата інформації є неприйнятною. Хоча стиснення без втрат забезпечує вищу якість, розмір файлу при цьому зменшується незначно.

З іншої сторони, стиснення з втратами, навпаки, досягає значного зменшення розміру файлу за рахунок видалення частини інформації. Цей процес є незворотнім — після стиснення деякі деталі зображення остаточно втрачаються, що може впливати на його чіткість і якість. Тим не менш, стиснення з втратами є ефективним для зберігання або передачі великої кількості графічних даних, де допустиме певне погіршення якості.

Прикладами алгоритмів стиснення з втратами є:

- дискретне вейвлет-перетворення (DWT);
- фрактальне стиснення;
- кодування з використанням трансформацій.

Одним з найпоширеніших форматів стиснення з втратами є **JPEG**. Він особливо підходить для зображень і фотографій, що не потребують прозорості, і забезпечує прийнятний компроміс між якістю та розміром файлу.

Алгоритм JPEG (Joint Photographic Experts Group) [7] є стандартизованим механізмом стиснення зображень. Він розроблений для ефективного стиснення кольорових (24-бітних) і градацій сірих (8-бітних) зображень. Найкраще JPEG працює з фотографіями, природними сценами та художніми зображеннями, однак він менш ефективний для мультфільмів чи зображень із чіткими контурами, таких як креслення.

JPEG побудований з урахуванням особливостей людського зору: наприклад, людське око менш чутливе до дрібних змін кольору, ніж до змін яскравості, тому алгоритм приділяє більше уваги саме яскравості пікселів.

Одна з ключових переваг JPEG [10] — можливість регулювати ступінь втрати якості шляхом зміни параметрів стиснення (переважно через квантування). Це дозволяє балансувати між розміром файлу та візуальною якістю, залежно від потреб користувача. Процес кодування JPEG із втратами зображений на рисунку 1.3.

Рисунок 1.3 – Схема JPEG кодування з втратами

Цей процес кодування складається з кількох основних етапів, які формують чотири підсистеми. На першому етапі зображення проходить перетворення кольорового простору (наприклад, з RGB у YCbCr) та

субсемплінг — зменшення просторової роздільної здатності для компонентів кольору, що менш критичні для людського зору.

Далі зображення ділиться на блоки розміром 8×8 пікселів, які готуються для подальшої обробки. Кожен блок проходить дискретне косинусне перетворення (DCT), яке переводить дані з просторової області у частотну. Це дозволяє представити зображення у вигляді коефіцієнтів, що описують різні частоти — від низьких до високих.

Після цього застосовується квантування — ключовий етап у стисненні з втратами. У цьому процесі значення високочастотних коефіцієнтів (які найменше впливають на якість зображення) наближуються до нуля або повністю відкидаються, що дозволяє значно зменшити обсяг даних.

Потім виконується ентропійне кодування — наприклад, кодування Хаффмана у поєднанні з Run-Length кодуванням. Для компонентів постійного струму (DC) використовується попереднє диференціальне кодування, яке враховує зміну значень між сусідніми блоками. У результаті формується бітовий потік — стислий варіант зображення.

У сучасних підходах до стиснення зображень все частіше застосовуються гібридні методи, які поєднують класичні алгоритми (такі як JPEG, JPEG2000, DCT, DWT, ентропійне кодування) з глибокими нейронними мережами, зокрема згортковими нейронними мережами (CNN), автоенкодерами, рекурентними мережами (RNN) та генеративними змагальними мережами (GAN) [8]. Така інтеграція дозволяє усунути недоліки традиційних методів, які на низьких бітрейтах можуть створювати артефакти у вигляді розмиття, блокування чи різких переходів. Завдяки гнучкості та здатності до навчання, нейромережі забезпечують кращу адаптацію до структури зображення та зменшення втрат важливої інформації при стисненні [2].

Особливої уваги заслуговує використання автоенкодерів, які можуть витягати компактне, бінарне представлення зображення, що дає змогу досягати більш високих показників якості порівняно з класичними методами.

Такі підходи демонструють переваги в задачах, де важливо зберегти ключову візуальну інформацію, наприклад, у медичній візуалізації [9] або відеоаналітиці. У літературі також відзначають, що останні огляди присвячені саме гібридним моделям, що комбінують переваги традиційних та сучасних методів стиснення, сприяючи створенню більш ефективних, адаптивних і якісних алгоритмів для різноманітних застосувань.

Усі описані вище методи — алгоритмічні, нейронні та гібридні — орієнтовані на стиснення надмірностей в окремих зображеннях. Проте в сучасних умовах часто виникає потреба зберігати великі об'єми візуально схожих зображень зі спільними елементами, наприклад: супутникові знімки, щоденні записи з камер відеоспостереження, фіксація правопорушень або передача даних з безпілотників. Для ефективного стиснення таких даних можна застосовувати кодування растрових зображень на основі подібності фрагментів [6].

Метод стиснення зображень Frasim [6] (fragment similarity) базується на використанні подібності фрагментів та включає два основні процеси: наповнення бази даних фрагментів і кодування зображення. Спочатку вхідне зображення перетворюється на масив пікселів і за потреби змінюється його розширення. Потім зображення розділяється на фрагменти попередньо заданого фіксованого розміру, до прикладу 32x32 пікселі й виділяється багато фрагментів за допомогою ковзного вікна для отримання максимальної кількості варіантів фрагментів й врахування перехідних моментів на зображенні. Кожен фрагмент обробляється нейронною мережею, яка виділяє його ознаки у вигляді числового вектору. Разом із самими фрагментами та їх характеристиками ці ознаки зберігаються в базу даних, яка надалі слугує основою для пошуку подібних елементів при кодуванні нових зображень. Спрощена схема процесу кодування наведена на рисунку 1.4.

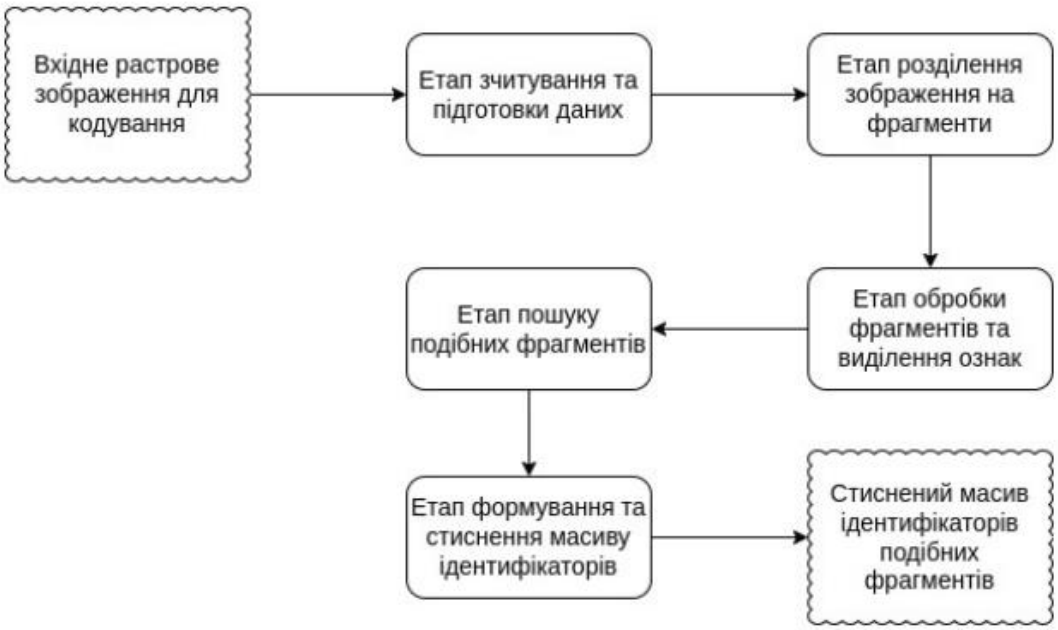


Рисунок 1.4 – Схема процесу кодування зображення

У процесі кодування [6] нове зображення також ділиться на фрагменти, для кожного з яких виділяються ознаки. Порівнюючи їх із ознаками фрагментів із бази даних (з урахуванням однакового розміру фрагментів), за допомогою евклідової відстані знаходиться найбільш схожий фрагмент, і зберігається його ідентифікатор. В результаті формується масив ідентифікаторів, який уже займає набагато менше місця, ніж оригінальне зображення. На завершальному етапі цей масив додатково стискається за допомогою методу lzma, що ще більше зменшує обсяг вихідних даних. Таким чином, даний метод забезпечує ефективне стиснення, зберігаючи при цьому можливість точного відновлення зображення за рахунок пошуку максимально схожих фрагментів.

Запропонований підхід є актуальним для вирішення окресленої проблематики та має значний потенціал для подальшого вдосконалення. Перспективними напрямами розвитку є використання високопродуктивних згорткових нейронних мереж для виділення ознак фрагментів, оптимізація методів збереження оброблених фрагментів у базах даних, застосування накопичених даних для предиктивного розширення області видимого зображення, а також удосконалення алгоритмів стиснення фрагментів перед їх збереженням. Також за допомогою використання дерев для пошуку між

векторними ознаками кожного фрагменту можна застосувати спосіб на основі регресійної або ж класифікаційної нейронної мережі для відновлення втрачених фрагментів або ж доповнення вже існуючих.

У межах даної роботи основну увагу буде приділено покращенню зазначених аспектів.

1.4 Аналіз та моделювання бізнес-процесів

Для моделювання бізнес-процесів використовуються BPMN моделі (рисунок 1.5-1.7).

Опис моделі бізнес-процесу заповнення бази даних додатку фрагментами (рисунок 1.5):

– користувач передає дані для заповнення на сервер;
– сервер обробляє дані й повертає статус виконаної роботи;
– користувач бачить статистичну інформацію.

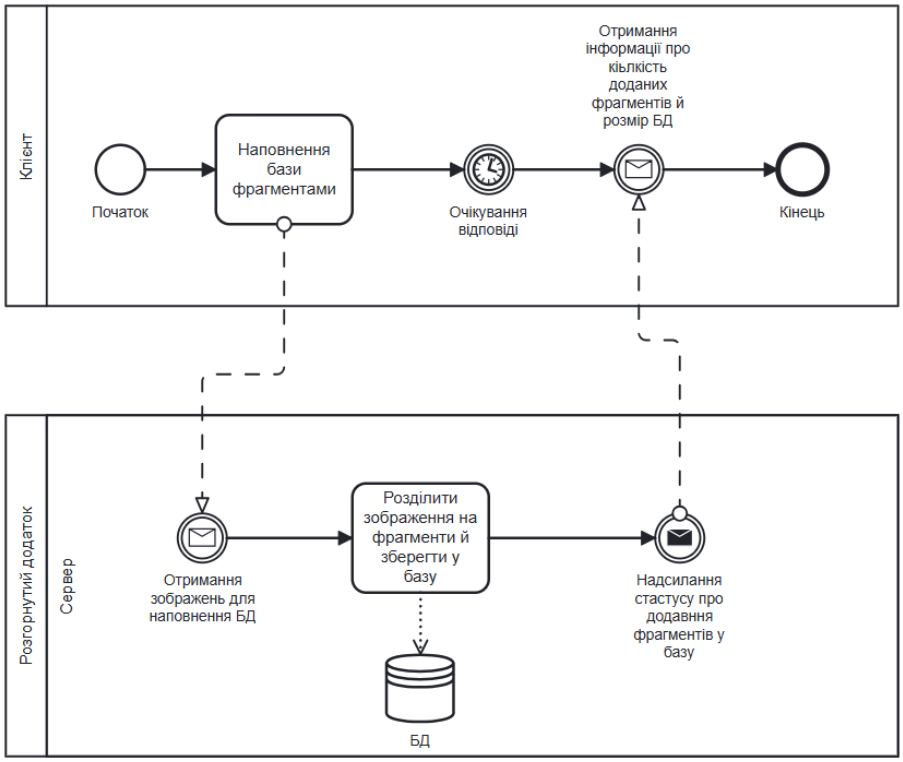


Рисунок 1.5 – Модель заповнення бази даних фрагментами

Опис моделі бізнес-процесу кодування й декодування зображення (рисунок 1.6):

– користувач передає фото для кодування на сервер;
– сервер обробляє фото й повертає у відповідь стиснуте зображення;
– користувач надсилає стиснене зображення на сервер;
– сервер відновлює фото з фрагментів на своїй стороні й повертає це фото користувачу.

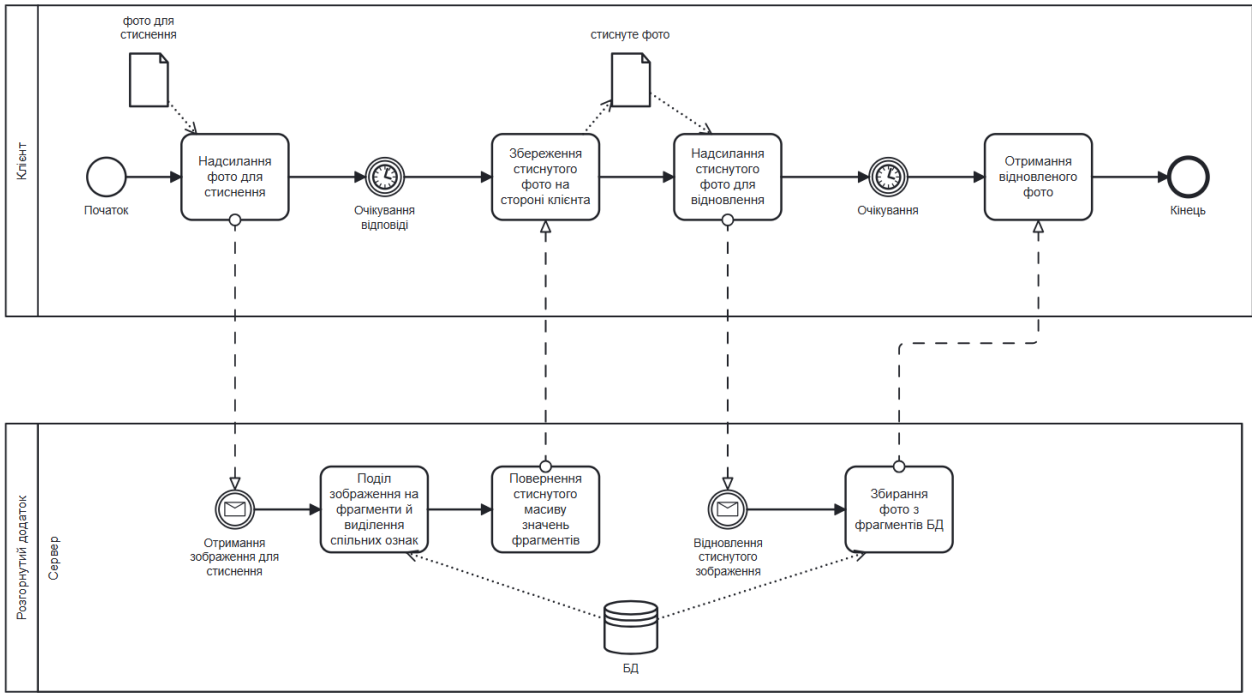


Рисунок 1.6 – Модель кодування та декодування зображення

Опис моделі бізнес-процесу збереження й завантаження існуючої бази фрагментів (рисунок 1.7):

– користувач передає на сервер збірку даних з фрагментами;
– сервер заміняє на своїй стороні базу з фрагментами;
– користувач надсилає повідомлення про бажання отримати поточну базу фрагментів;
– сервер зберігає фрагменти у сховище й передає користувачеві посилання на завантаження.

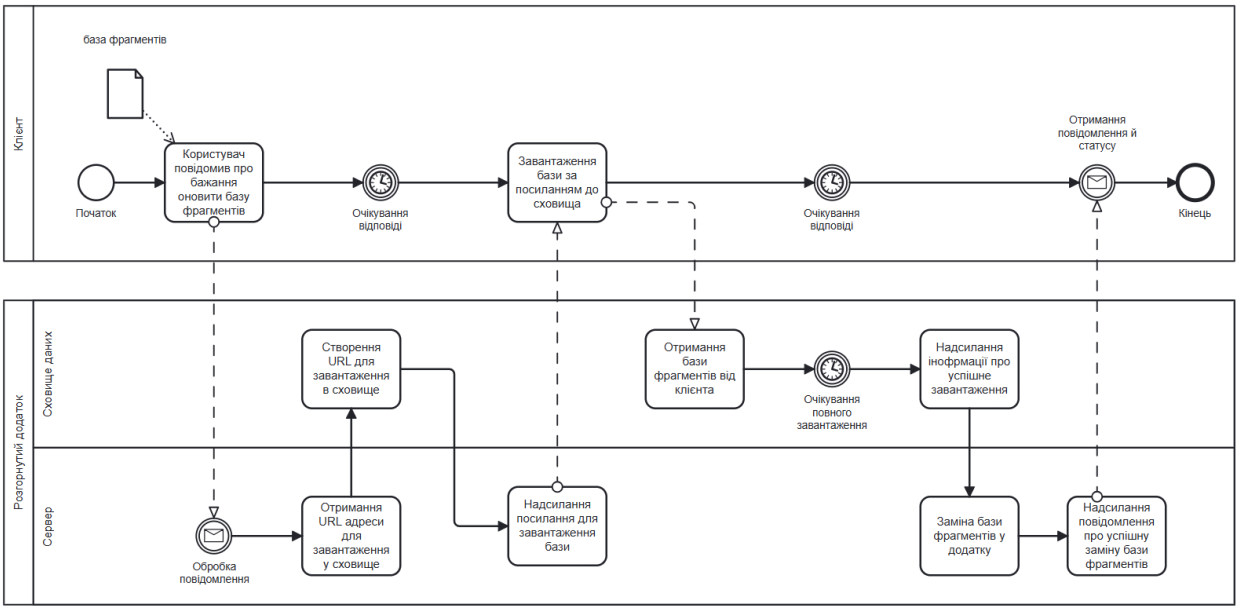


Рисунок 1.7 – Модель заміни бази даних фрагментів на сервері

Висновки до розділу

У цьому розділі було сформульовано постановку завдання для дипломного проєкту, включно з визначенням усіх ключових етапів і процесів, які мають бути реалізовані в межах роботи. Було проведено детальний аналіз предметної області проблеми стиснення растрових зображень, у ході якого виявлено потенційні можливості й способи для покращення вже існуючих алгоритмів кодування.

На завершення в розділі було представлено діаграми бізнес-процесів у нотації BPMN 2.0, які описують основні процеси, що мають бути реалізовані в межах розробки програмного забезпечення.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Головним функціоналом програмного забезпечення є кодування й декодування растрових зображень. Також користувач може здійснювати збереження й оновлення бази даних фрагментів, отримання метрики подібності між двома фотографіями, виділяти з зображення фрагменти й додавати його у поточну базу. Детальніше діаграма варіантів використання з ключовими акторами та основними функціями винесена до графічного матеріалу, креслення 1.

В таблицях 2.1-2.6 наведено опис варіантів використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-01

Use case name	Кодування зображення
Use case ID	UC-01
Goals	Отримати файл із закодованим зображенням
Actors	Користувач, API бекенд
Trigger	Користувач хоче закодувати зображення
Pre-conditions	Користувач має файл зображення у форматі png, jpeg, jpg.
Flow of Events	Користувач переходить на вкладку з кодуванням зображення. Далі він прикріплює файл із зображенням до відповідного поля й натискає кнопку «Стиснути». Через деякий час приходить відповідь і з’являється додаткова кнопка для завантаження бінарного файлу із закодованим зображенням. Якщо користувач натискає, то йому пропонується обрати місце для завантаження цього файлу
Extension	Якщо користувач обере неправильний тип файлу, він не зможе надіслати зображення для кодування. Якщо виникає помилка під час кодування в користувача просять зробити повторну спробу
Post-Condition	Користувач отримує бінарний файл із закодованим зображенням

Таблиця 2.2 – Варіант використання UC-02

Use case name	Декодування зображення
Use case ID	UC-02
Goals	Розкодувати попередньо закодоване зображення
Actors	Користувач, API бекенд
Trigger	Користувач хоче отримати розкодоване зображення
Pre-conditions	Користувач має файл закодованого зображення у форматі .bin
Flow of Events	Користувач переходить на вкладку з декодуванням зображення. Далі він прикріплює файл із стиснутим зображенням до відповідного поля, вказує висоту й ширину фото, яке він хоче отримати на виході й натискає кнопку «Декодувати». Далі для користувача з'являється передпоказ його відновленого зображення і кнопка «Завантажити зображення». Якщо користувач натискає цю кнопку, то починається завантаження відновленого зображення у форматі .png
Extension	Якщо користувач обирає файл з неправильним розширенням, він отримає помилку й не зможе розкодувати зображення. Якщо користувач вказує неправильну висоту чи ширину файлу, він отримає помилку про неправильно вказані розміри вихідного зображення
Post-Condition	Користувач отримує файл з декодованим зображенням

Таблиця 2.3 – Варіант використання UC-03

Use case name	Отримання метрики подібності двох зображень
Use case ID	UC-03
Goals	Дізнатися метрику подібності між оригінальним та декодованим зображенням
Actors	Користувач, API бекенд
Trigger	Користувач хоче дізнатися метрику подібності
Pre-conditions	Користувач має файл оригінального та декодованого зображення у форматі png, jpeg, jpg.

Продовження таблиці 2.3

Flow of Events	Користувач переходить на вкладку з отриманням метрики подібності зображення. Далі він прикріплює оригінальне та декодоване зображення й натискає цільову кнопку. Потім користувач бачить на екран метрику подібності між 2 зображеннями у відсотках
Extension	Якщо користувач обирає неправильний тип файлу, він не наді- шле зображення. Якщо виникає помилка під час роботи на сервері в користувача просять зробити повторну спробу
Post-Condition	Користувач бачить метрику подібності

Таблиця 2.4 – Варіант використання UC-04

Use case name	Додавання фрагментів із зображення у базу
Use case ID	UC-04
Goals	Розширити базу фрагментів новими фрагментами із зображення
Actors	Користувач, API бекенд, Сховище даних
Trigger	Користувач хоче розширити базу фрагментів
Pre-conditions	Користувач має файл зображення у форматі png, jpeg, jpg.
Flow of Events	Користувач переходить на вкладку з додаванням фрагментів у базу. Далі він завантажує файл у вказане поле й натискає кнопку додавання фрагментів. Після успішного додавання користувач бачить статистику додавання фрагментів у базу
Extension	Якщо користувач обере неправильний тип файлу, він не зможе надіслати зображення
Post-Condition	

Таблиця 2.5 – Варіант використання UC-05

Use case name	Завантаження бази фрагментів
Use case ID	UC-05
Goals	Зберегти поточний стан бази фрагментів
Actors	Користувач, API бекенд, Сховище даних

Продовження таблиці 2.5

Trigger	Користувач хоче отримати базу фрагментів щоб використовувати її в іншому місці
Pre-conditions	
Flow of Events	Користувач переходить на вкладку з роботою з базою фрагментів й натискає кнопку отримання бази фрагментів. Далі для нього з’являється поле натиснувши на яке він зможе отримати завантажити готову базу фрагментів й фіксованому форматі
Extension	Якщо виникне помилка на стороні серверу в користувача попросять повторити спробу пізніше
Post-Condition	Користувач має збережену базу фрагментів

Таблиця 2.6 – Варіант використання UC-06

Use case name	Заміна бази фрагментів
Use case ID	UC-06
Goals	Змінити базу фрагментів, щоб мати можливість закодовувати більше зображень
Actors	Користувач, API бекенд, Сховище даних
Trigger	Користувач хоче замінити наявну базу фрагментів
Pre-conditions	Користувач має файл попередньо збережену базу фрагментів у визначеному форматі
Flow of Events	Користувач переходить на вкладку з роботою з базою фрагментів й натискає кнопку про заміну бази фрагментів. Для користувача відкривається нове поле де він завантажує необхідний файл й натискає кнопку завантажити. Після успішного завантаження користувач бачить повідомлення про успішне оновлення бази
Extension	Якщо користувач обере неправильний тип файлу, він не зможе його надіслати
Post-Condition	

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.7 наведено загальну модель вимог, а в таблицях 2.8 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.1.

Таблиця 2.7 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритети	Ризики
1	Валідація й перевірка цілісності зображень	FR-1	Високий	Високий
2	Виділення фрагментів і їх ознак із зображення	FR-2	Високий	Високий
3	Збереження фрагментів й ознак у спільній базі фрагментів	FR-3	Високий	Високий
4	Кодування зображення використовуючи подібність фрагментів	FR-4	Високий	Високий
5	Відновлення зображення з набору ідентифікаторів	FR-5	Високий	Високий
6	Доповнення зображень на основі збереженої бази фрагментів	FR-6	Середній	Середній
7	Відновлення втрачених частин зображення	FR-7	Середній	Середній
8	Можливість збереження поточної бази фрагментів	FR-8	Середній	Середній
9	Можливість заміни поточної бази фрагментів	FR-9	Середній	Середній
10	Додавання фрагментів із зображень у базу	FR-10	Високий	Високий

Таблиця 2.8 – Перелік функціональних вимог

Назва	Опис
FR-1	Валідація й перевірка цілісності зображень Програма повинна перевіряти всі зображення, які буду потрапляти у систему на відповідність формату .png, .jpg, jpeg й кодуванню відповідно до цих форматів для забезпечення стабільності роботи додатку
FR-2	Виділення фрагментів і їх ознак із зображення Потрібно правильно розділяти вхідні зображення на частини фіксованого розміру та виділяти з них вектор ознак за допомогою згорткової нейронної мережі
FR-3	Збереження фрагментів й ознак у спільній базі фрагментів Фрагменти повинні зберігатися у спільній для забезпечення кодування інших зображень
FR-4	Кодування зображення використовуючи подібність фрагментів Програма повинне вміти шукати подібні фрагменти між зображенням й базою фрагментів
FR-5	Відновлення зображення з набору ідентифікаторів Програма повинна вміти декодувати зображення на основі переданого масиву фрагментів
FR-6	Доповнення зображень на основі збереженої бази фрагментів Програма повинна передбачати фрагменти поза зображеннями на основі наявної бази фрагментів
FR-7	Відновлення втрачених частин зображення При отриманні неповної інформації про закодовані фрагменти зображення програма повинна відновлювати пропуски подібними фрагментами з бази
FR-8	Можливість збереження поточної бази фрагментів Необхідно мати можливість зберігати поточну базу фрагментів у файл

Продовження таблиці 2.8

Назва	Опис
FR-9	Можливість заміни поточної бази фрагментів Необхідно мати можливість заміняти поточну базу фрагментів з файлу іншої бази фрагментів
FR-10	Додавання фрагментів із зображень у базу Програма повинна вміти додавати нові фрагменти із зображень у наявну базу

	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6
FR-1	+	+	+	+	+	+
FR-2	+	+				
FR-3		+	+			
FR-4	+					
FR-5		+				
FR-6		+				
FR-7		+				
FR-8				+		
FR-9					+	
FR-10	+					+

Рисунок 2.1 – Матриця трасування вимог

2.3 Розроблення нефункціональних вимог

У цьому підрозділі наведено нефункціональні вимоги до розроблюваного програмного забезпечення, згруповані за категоріями.

а) Надійність і стабільність.

1) Програмне забезпечення повинно бути стійким до некоректних або непередбачуваних дій користувача.

2) Усі вхідні дані повинні проходити обов’язкову валідацію для запобігання помилкам під час обробки.

3) У разі виникнення помилки користувач повинен отримати зрозуміле повідомлення про причину та спосіб її усунення.

б) Продуктивність і масштабованість.

1) Тривалість операції кодування одного зображення не повинна перевищувати 30 секунд при середньому навантаженні.

2) Програмне забезпечення повинно підтримувати використання графічних процесорів (GPU) для підвищення швидкодії моделей глибинного навчання.

3) Архітектура програмного забезпечення повинна дозволяти горизонтальне масштабування — розширення ресурсів при зростанні кількості запитів.

в) Збереження та керування даними.

1) Основні версії бази фрагментів, включаючи ознаки зображень, повинні зберігатися у хмарних сховищах даних (наприклад, Google Cloud Storage або AWS S3).

2) Дані повинні бути організовані з урахуванням вимог до швидкого пошуку, збереження цілісності та можливості масштабування.

г) Інфраструктура та розгортання.

1) Програмне забезпечення повинно бути розгорнуте в середовищі Google Cloud Platform.

2) Повинна бути реалізована підтримка безперервної інтеграції та доставки (CI/CD), зокрема через сервіси на зразок GitHub Actions або Cloud Build.

3) Базові параметри моделей кодування (наприклад, розмір фрагментів або границя подібності фрагментів) повинні бути конфігурованими без потреби повторного релізу застосунку.

2.4 Аналіз системних вимог

Клієнтська частина застосунку повинна бути доступною на персональних комп'ютерах та мобільних пристроях з наступними операційними системами:

- Windows (Windows 11, Windows 10);
- Unix-подібні системи (MacOS, Linux);
- Android;
- iOS.

Фронтенд застосунок повинен коректно працювати з основними сучасними веббраузерами:

- Google Chrome (останньої версії);
- Mozilla Firefox;
- Microsoft Edge;
- Safari.

Мінімальна конфігурація для розгортання серверної частини застосунку повинна бути:

- архітектура процесора: x86-64 або ARM64, з мінімум 32 потоками обробки;
- об'єм оперативної пам'яті (RAM): не менше 8 ГБ;
- швидкість з'єднання з мережею Інтернет: не нижче 20 Мбіт/с;
- графічний процесор (GPU): сумісний з CUDA, рекомендовано NVIDIA Tesla T4;
- програмне забезпечення: встановлений CUDA Toolkit, сумісний з обраною версією TensorFlow.

Рекомендована конфігурація для розгортання бекенд частини застосунку повинна бути:

– архітектура процесора: x86-64 або ARM64, з мінімум 64 потоками обробки;
– об’єм оперативної пам’яті (RAM): не менше 32 ГБ;
– швидкість з’єднання з мережею Інтернет: не нижче 100 Мбіт/с;
– графічний процесор (GPU): сумісний з CUDA, рекомендовано NVIDIA L4 або вище.

Програмне забезпечення: встановлений CUDA Toolkit, сумісний з обраною версією TensorFlow.

2.5 Аналіз економічних показників програмного забезпечення

Оцінку основних економічних показників даного проєкту будемо здійснювати за допомогою способу Use-Case Points [11] оскільки він підходить під визначені вимоги до програмного забезпечення у цьому розділі.

Для того щоб оцінити проєкт за методикою Use-Case Points потрібно мати таблицю усіх Use-Case у системі (таблиця 2.1) й оцінити їх за складністю. Також необхідно скласти таблицю акторів й надати їм рівень складності (таблиця 2.9)

Таблиця 2.9 – Опис акторів для системи та їх складність

Назва актора	Опис	Складність
Користувач	Користувач, що входить в систему для створення, управління рахунками	Складний
API бекенд сервісу	Користувач, що керує системою та має доступ до адміністрування	Простий
Сховище даних	Система, що інтегрується для обробки платежів	Середній

В даній таблиці простий актор представляє іншу систему з визначеним API інтерфейсом програмування, середній актор це інша система, що

взаємодіє через протокол, такий як TCP/IP, і складний актор може бути особа, яка взаємодіє через графічний інтерфейс або вебсторінку.

Для визначення оцінки Unadjusted Actor Weight (UAW) скористаємося наступною таблицею 2.10.

Таблиця 2.10 – Оцінка акторів (UAW)

Тип	Коефіцієнт
Простий (використовує систему по перерозподіленому API)	1
Середній (використовує більш складний або гнучкий API)	2
Складний (в основному означає взаємодію з кінцевим користувачем)	3

Звідси маємо, що UAW за формулою 2.1 дорівнює

$$UAW = 1 * 3 + 1 * 2 + 1 * 1 = 6 \quad (2.1)$$

Тепер необхідно оцінити складність всіх Use-Cases, які присутні у програмному забезпеченні. Складність Use-Case будемо визначати за наступною таблицею 2.11.

Таблиця 2.11 – Критерії складності Use-Case

Складність Use-Case	Кількість транзакцій	Ваговий коефіцієнт
Простий	До 3	5
Середній	Від 4 до 7	10
Складний	Більше 7	15

Складемо таблицю 2.12 в якій зазначимо відповідності складності для всіх Use-Cases

Таблиця 2.12 – Оцінка складності вимог

Номер UC	Складність	Ваговий коефіцієнт
UC-01	Середній	10
UC-02	Середній	10
UC-03	Простий	5
UC-04	Простий	5

Продовження таблиці 2.12

Номер UC	Складність	Ваговий коефіцієнт
UC-05	Середній	10
UC-06	Простий	5

Визначивши всі варіанти використання і їхню складність порахуємо Unadjusted Use-Case Weight (UUCW) за формулою 2.2.

$$UUCW = 3 * 15 + 1 * 10 + 2 * 5 = 45 \quad (2.2)$$

Порахуємо оцінку Unadjusted Use-Case Points за формулою 2.3.

$$UUCP = UUCW + UAW = 45 + 6 = 51 \quad (2.3)$$

Далі для того щоб перейти від незваженої оцінки до зваженої спочатку треба оцінити технічну складність Technical Complexity Factor (TCF) за кожним з 13 показників у таблиці 2.13 нижче. Кожному показнику ставиться у відповідність оцінка від 0 до 5.

Таблиця 2.13 – Оцінка TCF

Фактор	Опис	Вага	Призначена оцінка	Вага х оцінка
T1	Розподілена система	2.0	4	8
T2	Цільові показники часу відгуку / продуктивності	1.0	3	3
T3	Ефективність для кінцевого користувача	1.0	5	5
T4	Складність внутрішньої обробки	1.0	5	5
T5	Повторне використання коду	1.0	3	5
T6	Простота встановлення	0.5	2	1
T7	Зручність у використанні	0.5	4	2
T8	Портованість на інші платформи	2.0	3	6
T9	Підтримка системи	1.0	3	3

Продовження таблиці 2.13

Фактор	Опис	Вага	Призначена оцінка	Вага х оцінка
T10	Паралельна / багатопотокова обробка	1.0	3	3
T11	Функціональність безпеки	1.0	5	5
T12	Доступ третіх сторін (через API або інтерфейси)	1.0	3	3
T13	Навчання кінцевих користувачів	1.0	2	2
			Загальний (TF):	51

Формула для обчислення TCF скористаємося формулою 2.4:

$$TCF = 0.6 + \left(\frac{TF}{100}\right) = 0.6 + 0.51 = 1.11 \quad (2.4)$$

Environmental Factor або ж фактор середовища (EF) обчислюється множенням значення кожного фактора (F1- F8) за його вагою та додаванням результатів, щоб отримати суму, що позначається EFactor.

Таблиця 2.14 – Оцінка критерію EFactor

Фактор	Опис	Вага	Призначена оцінка	Вага х оцінка
F1	Обізнаність із застосованим процесом розробки	1.5	3	4.5
F2	Досвід роботи з подібними застосунками	0.5	3	1.5
F3	Досвід команди у використанні об’єктно-орієнтованого програмування	1.0	4	4

Продовження таблиці 2.14

Фактор	Опис	Вага	Призначена оцінка	Вага x оцінка
F4	Кваліфікація провідного аналітика	0.5	2	1
F5	Мотивація команди	1.0	5	5
F6	Стабільність вимог	2.0	4	8
F7	Наявність працівників з частковою зайнятістю	-1.0	0	0
F8	Складність використовуваної мови програмування	-1.0	1	-1
			Загальний (EFactor):	22

Звідси Environmental Factor розраховується за формулою 2.5.

$$EF = 1.4 + (-0.03 * EFactor) = 0.74 \quad (2.5)$$

Тепер маючи всі показники можна розрахувати UCP (Use Case Points) за наступною формулою 2.6.

$$UCP = (UUCW + UAW) * TCF * ECF = 51 * 1.11 * 0.74 = 41.9 \quad (2.6)$$

Цей показник свідчить про середню складність проєкту, яка включає реалізацію декількох функціональних сценаріїв користувача, обробку графічної інформації за допомогою глибинних нейронних мереж, інтеграцію з хмарною інфраструктурою, а також необхідність реалізації масштабованого вебсервісу.

Виходячи з прийнятого коефіцієнта продуктивності (наприклад, 3 людино-годин на один UCP), загальна трудомісткість оцінюється на рівні приблизно 125 людино-годин. Це дозволяє здійснити попереднє планування термінів розробки програмного продукту, а також оцінити орієнтовний бюджет проєкту з урахуванням витрат на працю.

2.6 Постановка завдання на розробку програмного забезпечення

Метою даного дипломного проєкту є розробка розподіленого програмного забезпечення для стиснення та відновлення растрових зображень із застосуванням сучасних методів комп’ютерного зору та глибинного навчання. Основна ідея полягає у створенні ефективного інструменту кодування, який базується на поділі зображень на фрагменти, ідентифікації їхніх ознак за допомогою згорткових нейронних мереж та збереженні цих фрагментів у спеціальній базі для подальшого відновлення зображень.

У рамках проєкту передбачається реалізувати повноцінний розподілений додаток, що складається з фронтенд і бекенд частин, які будуть взаємодіяти через сучасні вебтехнології. Фронтенд має бути простим і зручним у використанні, забезпечувати користувачу необхідну статистику щодо процесу кодування, таку як ступінь подібності фотографій, відсоток стиснення, час кодування та інші параметри.

Серверна частина програми повинна включати механізми обробки зображень з використанням згорткових нейронних мереж для визначення ознак фрагментів, зберігання їх у базі даних, а також реалізовувати алгоритми передбачення фрагментів за допомогою векторних дерев та нейронних мереж. Особлива увага приділяється можливості роботи з базами фрагментів, які можуть зберігатися і замінюватися залежно від конкретних завдань і ситуацій.

Програмний продукт повинен бути розгорнутий у хмарному середовищі Google Cloud Platform із використанням графічних процесорів (GPU) для підвищення продуктивності та швидкодії.

Висновки до розділу

У цьому розділі було проведено всебічний аналіз та формування вимог до розроблюваного програмного забезпечення.

Спочатку було визначено варіанти використання для програмного забезпечення та розроблено їхню діаграму. Кожен Use-Case був чітко описаний, що дозволяє окреслити межі функціональності та призначення додатку.

Далі було розроблено функціональні вимоги, що дало змогу визначити ключові функції, які програмне забезпечення повинно виконувати, зокрема алгоритми кодування зображень, роботу з базою фрагментів та обробку запитів користувача. Також було сформовано нефункціональні вимоги, які включають якість, надійність, для додатку а також вимоги до масштабованості, безпеки та сумісності з платформами.

Аналіз системних вимог охопив технічні характеристики, необхідні для ефективного розгортання як клієнтської, так і серверної частин додатку.

Наступним було проведено оцінку економічних показників, де було застосовано метод Use-Case Points для кількісної оцінки обсягу робіт, що дозволило зробити висновки щодо трудомісткості проєкту, його вартості та ресурсних потреб.

Таким чином, другий розділ визначив комплексне розуміння вимог до програмного продукту, що є важливою основою для успішної розробки та впровадження системи.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Під час розробки програмного забезпечення для реалізації вимог до даного дипломного проєкту було використано принцип клієнт-серверної архітектури, де клієнт відповідає за графічне представлення інтерфейсу й можливого функціоналу додатку, а серверна частина в свою чергу виконує фактичну обробку запитів користувача, визначає логіку й взаємодію з іншими сервісами. Застосунок використовує хмарну інфраструктуру Google Cloud Platform для розгортання. Застосунок складається з двох основних компонентів: клієнтської (фронтенд) та серверної (бекенд) частин, які взаємодіють між собою через HTTP-запити до API. Уся логіка та обробка даних реалізована на стороні бекенду, а зберігання структурованих та неструктурованих даних здійснюється за допомогою хмарних сервісів BigQuery та Cloud Storage.

Загальна структура веб-сервісу зображена на рисунку 3.1 за допомогою діаграми C4 рівня 2.

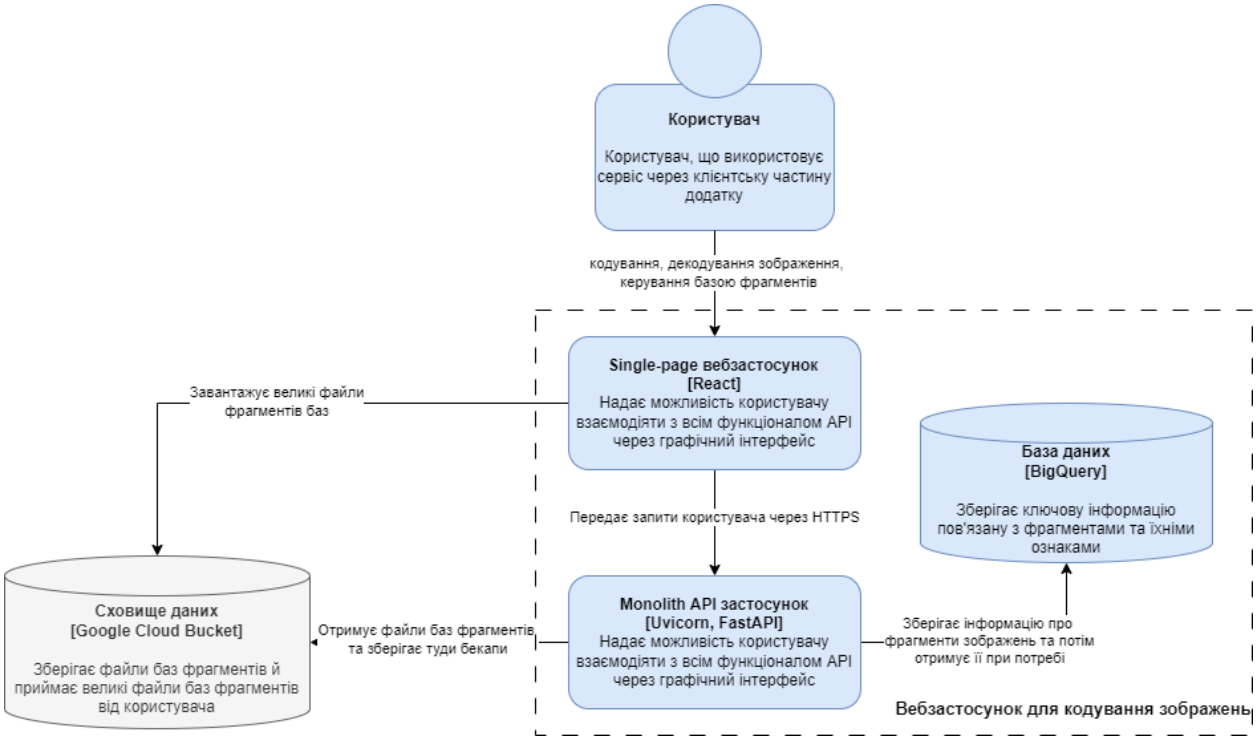


Рисунок 3.1 – Діаграма C4 рівня 2

Основні функції клієнтської частини це:

- взаємодія з користувачем через графічний інтерфейс;
- надсилання HTTP-запитів до API бекенду (наприклад, для запуску кодування/декодування зображень, завантаження фрагментів, отримання статистики тощо);
- візуалізація результатів обробки (індикатори подібності, стиснення, час тощо);
- робота у веббраузерах на ПК та мобільних пристроях.

Серверна частина додатку в свою чергу реалізована з використанням монолітної архітектури, що виконує всю бізнес-логіку:

- обробка запитів користувачів через REST API;
- кодування зображень (розбиття на фрагменти, обчислення ознак за допомогою згорткових нейронних мереж);
- декодування зображень з використанням ідентифікаторів фрагментів;
- робота з векторними структурами для передбачення втрачених чи зовнішніх фрагментів;
- керування збереженими базами фрагментів (завантаження, заміна, використання);
- формування статистики кодування.

У якості зовнішніх сервісів у застосунку використовуються можливості хмарної платформи Google Cloud, зокрема сервіси BigQuery та Cloud Storage. BigQuery виконує роль бази даних, у якій зберігається інформація про ознаки зображень і фрагментів, забезпечується ефективне виконання запитів для пошуку подібних фрагментів. Сервіс Cloud Storage використовується для зберігання файлів баз фрагментів, приймання великих файлів від користувача, а також для обміну даними та резервного копіювання.

Детальніший опис роботи монолітної серверної частини додатку виконаний за допомогою діаграми компонентів у UML нотації, яка наведена у графічному матеріалі, креслення 2.

Застосунок на сервері побудований за принципом монолітної архітектури, де всі основні компоненти об’єднані в одну систему. Головний компонент — це Server API, який відповідає за обробку запитів від клієнтської частини, налаштування маршрутів і виклик відповідної логіки застосунку.

Основну логіку роботи з зображеннями (їх кодування, декодування, пошук схожих фрагментів тощо) реалізовано в модулі Encoder. Саме цей модуль використовується компонент Server API для обробки запитів, які стосуються зображень.

Окремим важливим елементом є блок GCP Utils, який відповідає за взаємодію з хмарними сервісами Google Cloud. Він в свою чергу складається з 3 підкомпонентів:

- BigQuery Utils — для роботи з базою BigQuery, у якій зберігаються ознаки фрагментів;
- Cloud Bucket — для збереження та отримання файлів з Cloud Storage;
- GCP Credentials — для зберігання даних авторизації до сервісів Google Cloud.

Окремо використовується компонент, який не виділений на діаграмі – це Decorators. Він містить обгортки — спеціальні функції, які допомагають повторно використовувати загальні частини логіки (наприклад, вимір часу виконання або обробка помилок) у різних місцях коду.

Уся система працює узгоджено: Server API приймає запити, викликає Encoder для виконання логіки, а той у разі потреби звертається до GCP Utils. Така структура забезпечує зрозумілу побудову і полегшує подальший розвиток проєкту.

3.2 Архітектурні рішення та обґрунтування вибору засобів розробки

Під час розробки програмного забезпечення було прийнято кілька ключових архітектурних рішень, що забезпечують ефективність, масштабованість та відповідність функціональним і нефункціональним вимогам системи.

В першу чергу застосунок базується на платформі Google Cloud, як було вже описано вище та взаємодії з такими їхніми ключовими сервісами як Google Cloud Storage та Google BigQuery. Бекенд взаємодіє за допомогою офіційних Python SDK із цими зовнішніми сервісами.

Для зберігання файлів використовується Google Cloud Storage, а для обробки та зберігання табличних даних — Google BigQuery. Комунікація з цими сервісами відбувається через HTTPS із використанням сервісних акаунтів та авторизаційних ключів, які зберігаються в окремому модулі.

Для забезпечення нефункціональних вимог застосунок розгортається у хмарному середовищі Google Cloud з використанням сервісу Cloud Run, що дозволяє запускати сервер з підтримкою GPU. Це дає змогу ефективно виконувати обчислення, пов’язані з кодуванням і декодуванням зображень. Висока продуктивність досягається завдяки використанню асинхронного фреймворку FastAPI, а також обробці зображень із використанням графічного процесора. Надійність і відновлюваність реалізуються шляхом зберігання баз фрагментів у Google Cloud Storage з можливістю їхнього резервного копіювання. Масштабованість забезпечується гнучким керуванням ресурсами у хмарі, а підтримка коду спрощується завдяки модульній архітектурі проєкту.

Використання сервісів Google Cloud Platofrm не є обов’язковим для даного проєкту, сама система написана таким чином, що дозволяє розгортання на будь-яких інших хмарних сервісах таких як (Amazon, Microsoft Azure, VMWare та інші) або ж локально на власних серверних потужностях.

Типи баз даних, що використовуються у проєкті, включають:

- реляційну базу даних (BigQuery) — для швидкої обробки запитів до великої кількості фрагментів зображень та їх ознак;
- об’єктне сховище (Cloud Storage) — для збереження файлів баз фрагментів у форматах .json, .npy, які використовуються при кодуванні/декодуванні зображень.

Для розробки програмного забезпечення даної роботи було обрано сучасний, гнучкий технологічний стек, що дозволяє максимально точно реалізувати поставлені вимоги. Кожне обране технологічне рішення базувалося на порівнянні з найближчими аналогами та з урахуванням специфіки реалізації вимог.

Клієнтська частина була реалізована за допомогою бібліотеки React, яка є одним із найпопулярніших рішень для побудови односторінкових застосунків (SPA) завдяки високій продуктивності, великій спільноті та багатому екосистемному середовищу. У порівнянні з альтернативами, такими як Angular чи Vue, React забезпечує більшу гнучкість у побудові архітектури застосунку та простота в реалізації програмного коду. Для локальної розробки використовується Vite — сучасний збирач, який значно швидше застарілого Webpack виконує білд та запуск проєкту. TailwindCSS був обраний для стилізації інтерфейсу через свою простоту, яка дозволяє швидко створювати адаптивні, стильні інтерфейси без потреби писати багато CSS вручну.

Серверна частина була реалізована на основі FastAPI, асинхронного Python-фреймворку, що дозволяє швидко створювати RESTful API з високою продуктивністю. У порівнянні з Django або Flask, FastAPI надає повноцінну підтримку асинхронності, автоматичну генерацію OpenAPI документації та зручну систему типізації, що значно полегшує супровід коду. Сервер запускається через Uvicorn — легкий ASGI-сервер, оптимізований для асинхронних застосунків.

Для реалізації функцій машинного навчання та обробки зображень було використано TensorFlow, як одну з найпотужніших бібліотек глибинного навчання. Її вибір обґрунтований широким функціоналом, підтримкою GPU та тісною інтеграцією з екосистемою Google Cloud. Для виконання пошуку подібних фрагментів використовується Annoy — ефективна бібліотека для побудови векторних дерев, що дозволяє виконувати наближений пошук за ознаками зображень. Для обробки зображень було обрано OpenCV, оскільки це найпотужніша бібліотека для роботи з графікою у Python, яка має широку підтримку функцій, необхідних для обробки фрагментів. Стиснення даних здійснюється за допомогою LZMA, як одного з найефективніших алгоритмів без втрат.

Для зберігання та обробки даних у хмарі використано Google Cloud SDK, який надає API-доступ до сервісів Google Cloud, зокрема BigQuery (аналітична база даних для збереження та пошуку фрагментів за ознаками) і Cloud Storage (хмарне сховище для збереження та завантаження великих файлів баз фрагментів).

Контроль версій коду здійснюється за допомогою Git, а для автоматичного розгортання системи на сервері використовується GitHub Actions, що дозволяє налаштувати CI/CD-процес і забезпечити стабільність оновлень та тестування.

Інструменти розробки було обрано з урахуванням зручності та функціональності: PyCharm використовувався для бекенд-розробки завдяки підтримці WSL та можливості запуску коду з GPU на віртуальній машині, що критично для локальної обробки зображень. Для фронтенду застосовувався Visual Studio Code — легкий, але потужний редактор з підтримкою численних розширень і терміналу, що значно пришвидшує розробку UI-компонентів.

3.3 Конструювання програмного забезпечення

3.3.1 Опис алгоритму кодування

Однією з ключових функціональних вимог даного проєкту є модифікація з метою покращення вже існуючого алгоритму кодування зображення з використанням подібності фрагментів [6]. Загальний опис цього алгоритму разом зі схемою було наведено у пункті 1.3.2 (рисунок 1.4), а нижче подано його детальніше пояснення разом із запропонованими покращеннями.

По-перше, зображення, що подається на вхід, спочатку перетворюється та відновлюється до нестисненого формату, а саме — до матриці пікселів, де кожен елемент представляє піксель у форматі RGB. Ця операція є необхідною для подальших етапів, пов’язаних із розбиттям на фрагменти, виділенням ознак і відновленням зображення.

Після цього зображення розбивається на фрагменти фіксованого розміру, наприклад 16x16 пікселів. Для розбиття використовується схема ковзного вікна, коли кожен фрагмент частково перекриває попередній на задану кількість пікселів. Це дозволяє отримати більше фрагментів із зображення і краще охопити області з різкими переходами, змінами освітлення або контрасту. Процес розбиття зображення на фрагменти з ковзним вікном зображений на рисунку 3.2, де синім кольором позначену умовну зону для виділення фрагмента.

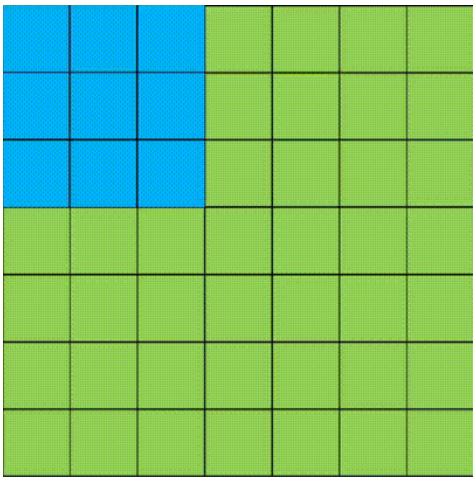


Рисунок 3.2 – Розбиття вхідного зображення на фрагменти

По-друге, здійснюється виділення якісних ознак з кожного фрагмента за допомогою згорткової нейронної мережі. Оскільки кожна архітектура нейромереж має свій стандартизований вхід (наприклад, для ResNet50 вхідне зображення повинно мати розмір 224x224 пікселів), фрагмент необхідно відповідним чином масштабувати. Після проходження крізь мережу отримується вектор ознак фіксованої довжини, що відображає ступінь відповідності фрагмента різним ознакам. Ці вектори зберігаються у структурі даних для подальшої обробки. Процес виділення ознак схематично зображений на рисунку 3.3.

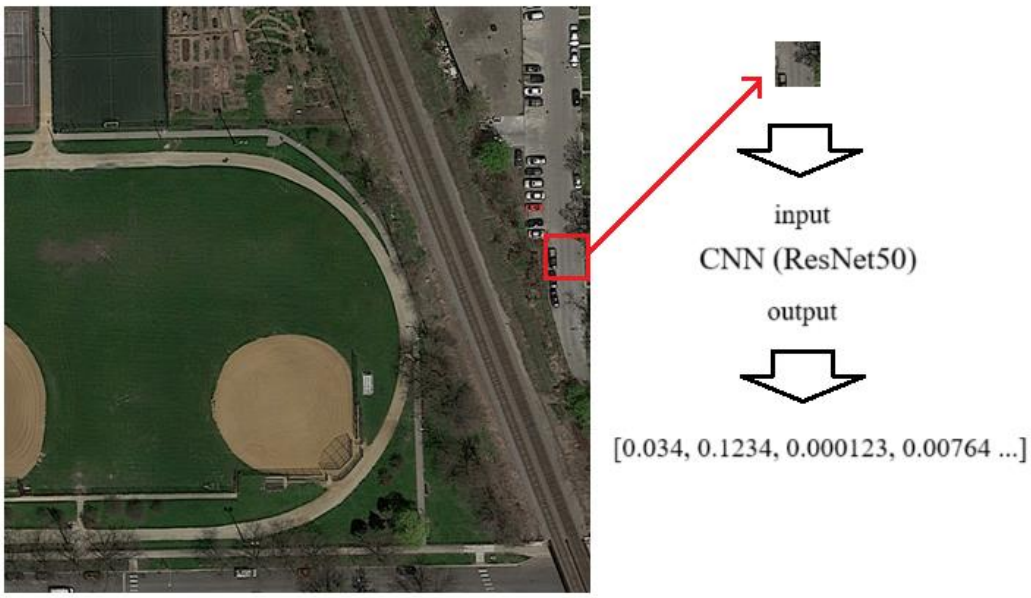


Рисунок 3.3 – Виділення вектору ознак з фрагмента за допомогою CNN

На наступному етапі після виділення ознак для кожного фрагмента виконується пошук подібних фрагментів у базі. Для цього використовують побудоване дерево векторних ознак, яке за допомогою евклідової відстані дозволяє з високою ймовірністю (понад 99%) [13] знайти найбільш схожий фрагмент.

Знайдений фрагмент додатково перевіряється за метриками схожості, наприклад SSIM чи MSE [12]. Якщо значення перевищує заданий поріг — до результату додається ID схожого фрагмента з бази. Інакше новий фрагмент додається до бази, а його ID також заноситься в результат. Це дозволяє зберігати унікальні частини зображення. Таким чином, кожна частина

зображення отримає унікальний ідентифікатор, за яким її можна буде відновити.

Далі, після формування масиву ідентифікаторів фрагментів з бази можна здійснити зворотне відновлення зображення. Для підвищення ступеня стиснення застосовується додаткове стиснення за допомогою універсального алгоритму LZMA. У результаті отримується компактне бінарне представлення зображення на основі унікальних фрагментів.

На останньому етапі циклу кодування-декодування відбувається відновлення зображення. Спочатку масив байтів декодується назад у список фрагментів за допомогою LZMA. Далі, за збереженими ідентифікаторами, відповідні фрагменти знаходяться у базі. З них формується масив пікселів, який конвертується у стандартний безвтратний формат зображення, наприклад PNG.

Тепер після опису основного алгоритму опишемо шляхи його модифікації.

3.3.2 Робота з втраченими фрагментами у зображенні

Як уже зазначалося, основною сферою застосування даного алгоритму є системи БПЛА, дрони, системи фотомоніторингу тощо. У більшості таких випадків передача інформації здійснюється через радіоканали, які часто не гарантують високу точність і стабільність сигналу. Тому, коли в процесі зйомки або передачі зображення частина даних втрачається — через перешкоди сигналу або перекриття об’єктиву сторонніми об’єктами — виникає потреба у відновленні втрачених фрагментів. Це можливо реалізувати за допомогою дерева векторних ознак, сформованого на основі бази зображень, що дозволяє знаходити та підставляти найбільш схожі фрагменти.

Для підвищення надійності зберігання та передачі даних необхідно змінити структуру вихідного файлу таким чином, щоб навіть у разі пошкодження можна було відновити якомога більше інформації про фрагменти. З цією метою було вирішено відмовитися від використання

алгоритму стиснення LZMA, оскільки втрата навіть одного байта стислого файлу унеможливлює декодування всього вмісту. Натомість застосовується серіалізований бінарний формат із власною структурою, яка дозволяє відновлювати окремі фрагменти незалежно один від одного. Формат одного запису у такому файлі має вигляд: «| 8 байт ID (INT64) | 8 байт X (INT64) | 8 байт Y (INT64) |». Детальніше структура описана в пункті 3.3.4.

Далі зобразимо алгоритм відновлення втрачених елементів за допомогою діаграми послідовності, що наведена у графічних матеріалах, креслення 3.

Алгоритм відновлення зображення умовно поділяється на три ключові етапи:

- виявлення пошкоджених зон;
- обчислення усередненого вектору ознак на основі сусідніх елементів;
- підбір найбільш відповідного фрагмента з бази за допомогою метрики SSIM.

На першому етапі виконується декодування бінарного файлу, отриманого від користувача. Після цього формується маска, яка відображає розташування заповнених і пошкоджених фрагментів зображення. На основі цієї маски визначаються координати відсутніх фрагментів.

Далі, для ефективного пошуку сусідів, з векторних ознак відомих фрагментів будується структура даних KDTree. KDTree — це ефективна структура для організації точок у багатовимірному просторі, яка забезпечує швидкий пошук найближчих сусідів за евклідовою або іншою відстанню.

Наступним кроком для кожного пошкодженого фрагмента запускається цикл, у межах якого за допомогою KDTree знаходяться найближчі 5 або 8 сусідів — залежно від того, чи розташований фрагмент по краю, чи всередині

зображення. З цих сусідніх фрагментів витягуються векторні ознаки, попередньо отримані з використанням згорткової нейронної мережі (CNN), і обчислюється їх усереднене значення.

На фінальному етапі виконується пошук k найбільш схожих фрагментів з бази за допомогою порівняння отриманого усередненого вектору ознак. Серед знайдених кандидатів обирається найкращий за метрикою SSIM (Structural Similarity Index) — шляхом обчислення середнього SSIM між кожним кандидатом і його оточенням за формулою (3.1)

$$SSIM_{score} = \frac{\sum_{i=1}^{k} SSIM(candidate, neighbour_i)}{k} \quad (3.1)$$

де SSIM(x, y) – функція, що обчислює метрику подібності між зображеннями x та y, k – кількість найближчих сусідів для втраченого елемента, candidate – потенційне зображення фрагменту для заміни втраченого та $neighbour_i$ сусідній елемент до втраченого з індексом i.

З даної формули фрагмент із найвищим середнім SSIM розміщується на місце втраченого. Цей процес повторюється для кожного відсутнього фрагмента окремо.

Приклад роботи алгоритму наведений на рисунку 3.4, який демонструє його ефективність у задачах реконструкції зображень з частковими пошкодженнями.

Рисунок 3.4 – Реконструкція втрачених фрагментів

3.3.3 Доповнення зображення подібними фрагментами з бази за схожими ознаками

Якщо необхідно передбачити, що могло знаходитись за межами наявного зображення, використовується той самий підхід: на основі усереднених ознак сусідніх фрагментів визначаються потенційні вектори ознак продовження, після чого за допомогою векторного дерева з бази вибираються найбільш схожі фрагменти. Далі за метрикою SSIM обирається фрагмент, який найкраще узгоджується з наявною сценою. У такий спосіб здійснюється логічне доповнення зображення фрагментами, які стилістично та структурно продовжують його.

Цей метод може бути корисним, зокрема, для генерації фонів, автоматичного кадрування, розширення сцен у зображеннях чи підготовки зображень до подальшої обробки в дизайнерських або машинних застосунках. Результат доповнення фото зображено на рисунку 3.5.

Рисунок 3.5 – Розширення правого краю зображення

3.3.4 Опис структури бази даних й сховища

В якості системи управління базами даних використовується хмарна платформа BigQuery. Вона добре підходить для подібних задач, оскільки не вимагає від розробника ручного контролю за продуктивністю — хмарні сервіси автоматично забезпечують масштабованість, налаштування та оптимізацію пошукових індексів, а також підтримують високу швидкодію.

База даних використовується для зберігання інформації про фрагменти зображень у спеціально визначеній структурі даних. Така ж структура застосовується і для збереження повної бази фрагментів у сховищі даних. Опис даної структури наведено у таблиці 3.1

Таблиця 3.1 – Опис структури для збереження інформації про фрагмент

Назва поля	Тип даних	Опис
id	int	Унікальний номер елементу
features	bytes	Перетворений у байтове представлення вектор ознак фрагмента
image	bytes	Стиснене й представлене у байтах зображення фрагменту

Як було зазначено раніше, для зберігання бази фрагментів використовується хмарний сервіс Google Cloud Storage, оскільки він краще пристосований до збереження та передачі великих обсягів даних. Інформація про фрагменти записується згідно зі заздалегідь визначеною структурою. Результатом збереження є файл з розширенням .npy, який у подальшому можна зчитати в коді.

Також, для коректного збереження закодованої інформації у бінарному файлі необхідно чітко визначити структуру даних, яка дозволить серіалізувати інформацію і безпечно її відновлювати. З цією метою було створено просту структуру, що складається з набору об’єктів. Кожен об’єкт містить такі атрибути: унікальний ідентифікатор фрагмента (ID), а також координати x і y, які вказують на позицію лівого верхнього кута фрагмента в межах зображення.

Усі ці значення перетворюються у байтовий формат типу INT64, що забезпечує сумісність із СКБД BigQuery, яка також використовує цей тип для збереження ідентифікаторів. Це дозволяє уникнути проблем з невідповідністю типів під час обміну або імпорту/експорту даних. Структура об'єкта фрагмента у бінарному файлі описана у таблиці 3.2

Таблиця 3.2 – Структура об'єкта фрагмента

Поле	Тип даних	Опис
id	INT64	Унікальний ідентифікатор фрагмента
x	INT64	Горизонтальна координата лівого верхнього кута фрагмента
y	INT64	Вертикальна координата лівого верхнього кута фрагмента

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

Для запобігання несанкціонованому доступу до серверної частини застосунку реалізовано механізм CORS (Cross-Origin Resource Sharing). Це протокол, заснований на HTTP-заголовках, який дозволяє серверу явно вказувати дозволені джерела (домен, схема або порт), з яких дозволено завантаження ресурсів у браузері, окрім власного. Така конфігурація знижує ризик атак з боку сторонніх веб-ресурсів.

Система бази даних, що використовується у проєкті, не є вразливою до SQL-ін’єкцій завдяки використанню типізованих запитів, а також валідації вхідних даних як на клієнтському, так і на серверному рівнях. Важливо також зазначити, що фронтенд-застосунок реалізовано таким чином, щоб унеможливити завантаження користувачем файлів, які не відповідають очікуваному формату. Наприклад, відправка зображень обмежена до допустимих типів (наприклад, .png, .jpg), що перевіряється ще до надсилання на сервер. У свою чергу, сервер додатково перевіряє MIME-тип і структуру отриманих файлів, що запобігає обробці потенційно шкідливого або некоректного вмісту.

Відкритий вихідний код застосунку забезпечує прозорість функціонування, дозволяючи стороннім розробникам перевіряти його безпеку

та за необхідності адаптувати до власних потреб. Це також сприяє дотриманню принципів open-source без ризику компрометації, оскільки користувачі можуть працювати з локальними копіями та модифікаціями, не відкриваючи їх публічно.

Висновки до розділу

Даний розділ присвячений конструюванню та розробленню програмного забезпечення. Розроблену архітектуру застосунку детально описано та наочно представлено з використанням діаграми C4 і діаграми компонентів за нотацією UML.

Далі обґрунтовано вибір архітектурних рішень, зокрема використання хмарних сервісів від Google, таких як Google Cloud Storage та BigQuery, які забезпечують масштабованість, надійність і зручність в обслуговуванні. Також проведено аргументацію щодо використання вибраного стеку технологій для реалізації функціональності застосунку.

Особливу увагу приділено алгоритмічній частині — детально описано принципи обробки та зберігання зображень, а також проведено модифікацію та адаптацію алгоритмів для підвищення ефективності роботи застосунку. Зокрема, було реалізовано механізм відновлення втрачених фрагментів зображення на основі векторних ознак, визначених за допомогою згорткової нейронної мережі (CNN), з використанням структури даних KDTree для пошуку найближчих сусідів та метрики SSIM для оцінки візуальної схожості фрагментів.

Наприкінці розділу розглянуто питання безпеки програмного забезпечення: описано реалізацію CORS-механізму, валідацію вхідних даних на клієнтській і серверній сторонах, захист від SQL-ін’єкцій, а також зменшення ризиків компрометації даних шляхом використання відкритих наборів даних без конфіденційної інформації.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

У цьому розділі проведено оцінювання якості програмного забезпечення, призначеного для кодування зображень з використанням глибинних нейронних мереж. Основною метою тестування є перевірка надійності, продуктивності та правильності роботи системи відповідно до заданих вимог.

Метою тестування є:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка роботи алгоритму кодування;

– перевірка правильності роботи з GCP платформою (розгортання додатку, запис даних у BigQuery);

– перевірка роботи з базою фрагментів.

Метриками для оцінки якості ПЗ обрано наступні:

– час кодування зображення;

– відсоток стиснення зображення;

– відповідність закодованого зображення до розкодованого;

– кількість пам’яті, що необхідно для збереження фрагменту в базі.

Опишемо значення кожної метрики та її покращення у пунктах 4.1.1 – 4.1.4.

4.1.1 Час кодування зображення

Однією з вимог дипломного проєктування є реалізація програмного забезпечення, що забезпечує підтримку використання графічних процесорів (GPU) з метою підвищення ефективності та швидкодії моделей глибинного навчання. У цьому підрозділі здійснюється оцінка продуктивності та

порівняльний аналіз часу кодування зображень при використанні центрального процесора (CPU) та графічного процесора (GPU).

Для експерименту використано базу фрагментів розміром 16×16, що містить 10 000 елементів, сформованих із супутникових знімків. [15] Було виконано кодування 100 різних тестових зображень розміром 400×400 пікселів із використанням 8-ми ядерного процесора на архітектурі Intel (CPU), також окремо для GPU Nvidia L4.

Обчислення середнього часу обробки одного зображення здійснюється за формулою (4.1):

$$V_{encoding} = \frac{\sum_{i=0}^{n} V_{i\ image}}{n} \quad (4.1)$$

Де n кількість тестових фотографій та $V_{i\ image}$ – час кодування i-ого зображення.

Результати роботи наведені у таблиці 4.1

Таблиця 4.1 – Час роботи метод з використання GPU та CPU

Час кодування	GPU	CPU
100 зображень 400x400	14.86 с	102.43 с

Як бачимо з результатів використання GPU пришвидшило роботу алгоритму приблизно у 8 раз.

4.1.2 Відсоток стиснення зображення

Для оцінки ефективності стиснення зображень доцільним є порівняння розробленого підходу з іншими поширеними форматами кодування зображень, зокрема JPEG та WebP. Окрім аналізу коефіцієнта стиснення, важливо також врахувати ступінь збереження візуальної якості зображення, що досягається шляхом порівняння оригінального (нестиснутого) зображення з його стиснутою версією. З цією метою використовується метрика

структурної подібності (SSIM), яка дозволяє оцінити збереження ключових візуальних характеристик.

Для проведення експериментального порівняння були обрані зображення нестиснутого формату PNG різних розмірів — від 256×256 до 608×608 пікселів. Методи кодування оцінювалися за двома основними критеріями: метрикою SSIM та коефіцієнтом стиснення, що визначається як відношення розміру оригінального зображення до розміру стиснутого (вхідного) зображення. Результати порівняння представлені в таблиці 4.2.

Таблиця 4.2 – Порівняльний аналіз методів стиснення зображень

Метод	Вхідний розмір зображення	Вихідний розмір, байт	Відсоток стиснення, %	Подібність
Подібність фрагментів	256x256	2 784	97,37%	99,89%
JPG	256x256	21 955	79,28%	97,99%
WebP	256x256	61 876	41,6%	99,91%
Подібність фрагментів	400x400	7 488	97,75%	98,96%
JPG	400x400	84 186	74,64%	98,11%
WebP	400x400	226 362	31,83%	99,92%
Подібність фрагментів	608x608	17 292	97,66%	98,84%
JPG	608x608	195 968	73,55%	99,89%
WebP	608x608	479 218	35,32%	99,96%

На основі даних з таблиці можна зробити висновок, що запропонований метод кодування на основі подібності фрагментів забезпечує значно вищий коефіцієнт стиснення порівняно з JPEG та WebP, зберігаючи при цьому високу якість зображення. Хоча WebP демонструє дещо вищі показники за метрикою SSIM, розмір вихідних файлів у ньому суттєво більший. Таким чином,

розроблений підхід є доцільним для використання в умовах обмежених обсягів пам’яті або каналів передачі даних.

4.1.3 Оцінка метрики SSIM для оригінального та декодованого зображення

Для об'єктивного порівняння якості стиснення зображень було проведено дослідження впливу наповненості бази фрагментів на відповідність між закодованим та розкодованим зображенням. У межах експерименту використовувалися бази різних розмірів із фрагментами фіксованого розміру 16×16 пікселів. Для кожного варіанту виконувалося кодування зображення з подальшим обчисленням метрики структурної подібності SSIM. Окрім цього, до аналізу було включено час кодування, оскільки він безпосередньо залежить від кількості фрагментів у базі, що дозволяє оцінити компроміс між швидкістю та якістю стиснення. Результати порівняння наведено у таблиці 4.3.

Таблиця 4.3 – Вплив кількості фрагментів у базі на якість та час кодування

Кількість фрагментів	Час кодування, с	Подібність SSIM
7203	13,54	0,84
12005	14,12	0,89
21609	14,86	0,92
50421	18,43	0,96
100842	26,81	0,98
151263	34,31	0,99
201684	37,9	0,99

Зі збільшенням кількості фрагментів у базі спостерігається суттєве зростання якості кодування, проте це супроводжується відповідним зростанням часу обробки, що слід враховувати при виборі оптимальної конфігурації системи.

4.1.4 Кількість пам’яті необхідної для збереження фрагменту в базі

У процесі реалізації було використано СКБД Google BigQuery, яка підтримує зберігання двійкових даних у форматі BLOB. Для оптимізації обсягів збереженої інформації було впроваджено додатковий етап стиснення фрагментів зображень із використанням безвтратного формату PNG. Це дозволило зберегти повну візуальну інформацію про фрагменти без втрат якості, при цьому суттєво зменшивши обсяг даних у порівнянні із прямим збереженням нестиснутого масиву пікселів у форматі RGB.

Рівень ефективності стиснення оцінювався шляхом порівняння розміру нестиснутого масиву (розрахованого як кількість пікселів, помножена на розмір одного пікселя) з розміром відповідного PNG-файлу. Візуалізація результатів порівняння подана на рисунку 4.1.

```
/home/user/python-project/venv/bin/python
Розмір нестиснутого фрагменту: 1024 байт
Розмір PNG після компресії: 599 байт
Коефіцієнт компресії: 0.58
```

Рисунок 4.1 – Коефіцієнт стиснення при збереженні фрагмента зображення

Згідно з розрахунками, при використанні бази, що містить приблизно 100 000 фрагментів, обсяг стиснутої пам’яті становив (формула 4.2):

$$100\,000 \times (1024 - 599) = 40{,}53 \text{ МБ} \qquad (4.2)$$

Таким чином, використання PNG-стиснення у поєднанні з BigQuery дозволило істотно знизити обсяг даних, необхідних для збереження, без втрати візуальної якості.

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.4 – 4.10.

Таблиця 4.4 – Тест 1. Завантаження зображення в систему

Початковий стан системи	Користувач знаходиться на сторінці «Додати фрагменти у базу» або «Кодувати зображення»
Вхідні дані	Зображення формату PNG, JPEG
Опис проведення тесту	Користувач натискає поле завантажити файл й обирає необхідний файл у новому вікні
Очікуваний результат	Зображення коректно відображається в інтерфейсі.
Фактичний результат	Зображення коректно відображається в інтерфейсі.

Таблиця 4.5 – Тест 2. Виконання кодування зображення

Початковий стан системи	Користувач знаходиться на сторінці «Кодувати зображення»
Вхідні дані	Зображення формату PNG, JPEG
Опис проведення тесту	Користувач натискає поле завантажити файл й обирає необхідний файл для кодування новому вікні. Після того як його завантажений файл відобразився у інтерфейсі користувач натискає кнопку «Стиснути зображення»
Очікуваний результат	Отримання статистики по кодуванню, стає активна кнопка для завантаження бінарного файлу стиснутого зображення.
Фактичний результат	Отримання статистики по кодуванню, стає активна кнопка для завантаження бінарного файлу стиснутого зображення

Таблиця 4.6 – Тест 3. Завантаження некоректного файлу для кодування

Початковий стан системи	Користувач знаходиться на сторінці «Кодувати зображення»

Продовження таблиці 4.6

Вхідні дані	Нетиповий файл для системи
Опис проведення тесту	Користувач натискає поле завантажити файл й обирає необхідний файл для кодування новому вікні. Після того як його завантажений файл відобразився у інтерфейсі користувач натискає кнопку «Стиснути зображення»
Очікуваний результат	Попередження від системи про відповідність обраного файлу
Фактичний результат	Попередження від системи про відповідність обраного файлу

Таблиця 4.7 – Тест 4. Декодування зображення

Початковий стан системи	Користувач знаходиться на сторінці «Декодувати зображення»
Вхідні дані	Бінарний файл попередньо закодованого зображення
Опис проведення тесту	Користувач натискає поле вибрати файл й обирає бінарний файл для декодування новому вікні. Після того користувач вводить розміри декодованого зображення й натискає кнопку «Декодувати»
Очікуваний результат	Користувач отримає інформацію про час декодування й декодоване зображення у форматі PNG. Також стає активна кнопка завантаження зображення.
Фактичний результат	Користувач отримає інформацію про час декодування й декодоване зображення у форматі PNG. Також стає активна кнопка завантаження зображення.

Таблиця 4.8 – Тест 5 Відновлення втрачених фрагментів

Початковий стан системи	Користувач знаходиться на сторінці «Декодувати зображення»
Вхідні дані	Бінарний файл попередньо закодованого зображення

Продовження таблиці 4.8

Опис проведення тесту	Користувач натискає поле вибрати файл й обирає бінарний файл для декодування новому вікні. Далі в полі вміст файлу користувач видаляє декілька записів. Після того користувач вводить розміри декодованого зображення, натискає перемикач «Увімкнути відновлення» й натискає кнопку «Декодувати»
Очікуваний результат	Користувач отримає інформацію про час декодування й декодоване зображення з відновленими фрагментами у форматі PNG. Також стає активна кнопка завантаження зображення.
Фактичний результат	Користувач отримає інформацію про час декодування й декодоване зображення з відновленими фрагментами у форматі PNG. Також стає активна кнопка завантаження зображення.

Таблиця 4.9 – Тест 6. Визначення подібності між 2 зображеннями

Початковий стан системи	Користувач знаходиться на сторінці «Визначити подібність зображень»
Вхідні дані	Два зображення формату PNG, JPEG, закодоване й розкодоване
Опис проведення тесту	Користувач натискає поле завантажити файл й обирає оригнальне зображення для завантаження новому вікні. Далі користувач натискає інше поле завантажити файл й обирає декодоване зображення для завантаження новому вікні. Після цього натискає кнопку «Порівняти»
Очікуваний результат	Після отримання відповіді від сервера користувач отримує метрику подібності між 2 зображеннями
Фактичний результат	Після отримання відповіді від сервера користувач отримує метрику подібності між 2 зображеннями

Таблиця 4.10 – Тест 7. Зміна порогу подібності фрагментів

Початковий стан системи	Користувач знаходиться на сторінці «Управління базою фрагментів»
Вхідні дані	-
Опис проведення тесту	Вводить у поле для «зміни порогу подібності» нове значення у межах від 0 до 1 й натискає кнопку «Змінити поріг»
Очікуваний результат	Поле з відображенням поточного порогу подібності змінюється на застосоване значення
Фактичний результат	Поле з відображенням поточного порогу подібності змінюється на застосоване значення

4.3 Опис контрольного прикладу

Опишемо контрольний приклад, що покриває усі розгалуження та аспекти функціоналу.

Спочатку користувач відкриває вебсайт програмного забезпечення та бачить головну сторінку (рисунок 4.2).

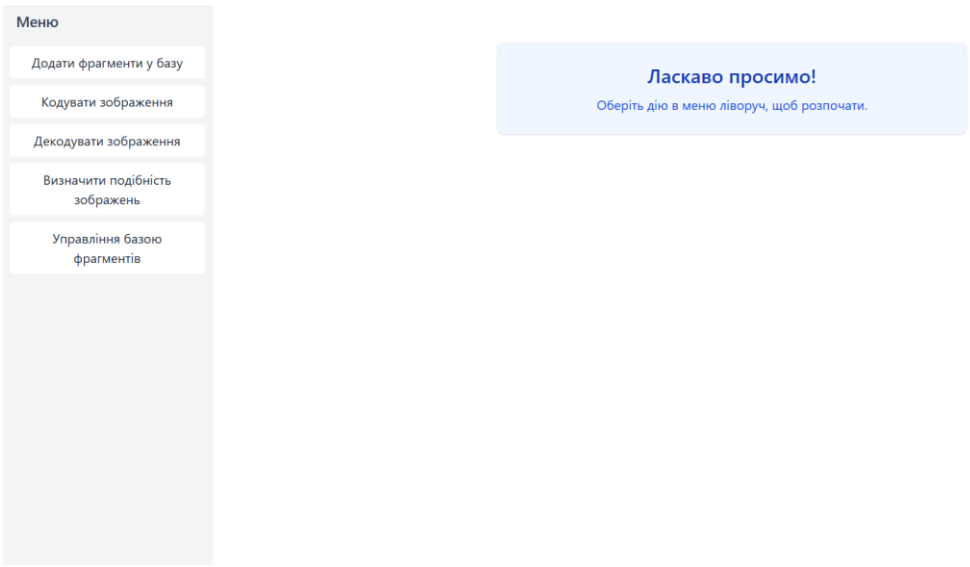


Рисунок 4.2 – Головна сторінка

Для здійснення кодування зображення необхідно перейти на сторінку «Кодувати зображення» (рисунок 4.3).

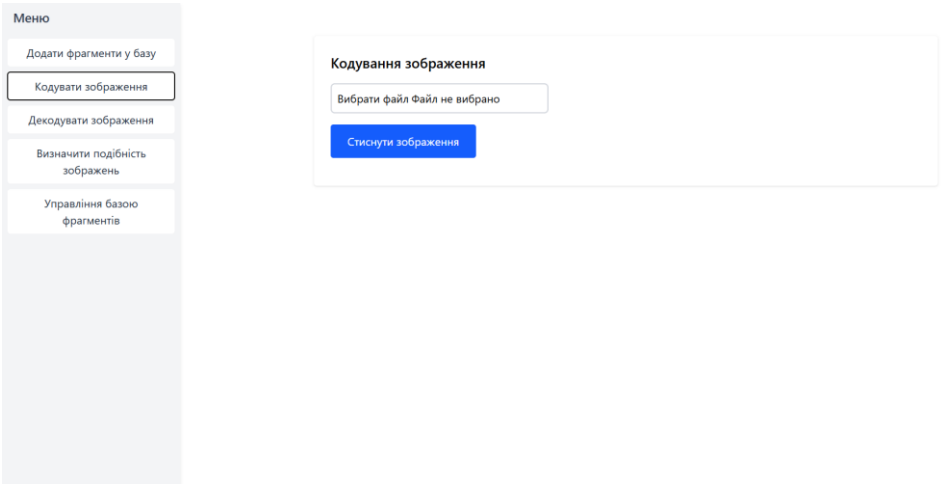


Рисунок 4.3 – Сторінка кодування зображення

Далі потрібно натиснути на поле «Вибрати файл» і завантажити необхідне зображення для кодування (рисунок 4.4). Після завантаження система відобразить поточний обраний файл користувача (рисунок 4.5).

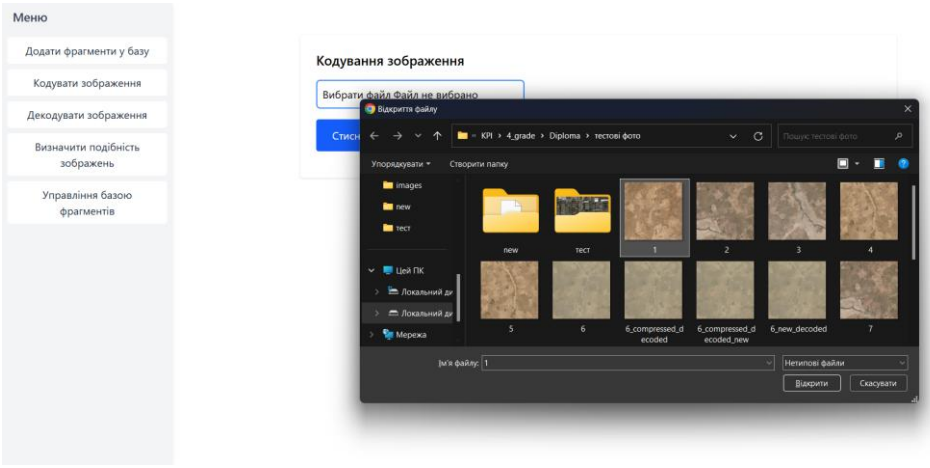


Рисунок 4.4 – Завантаження для кодування

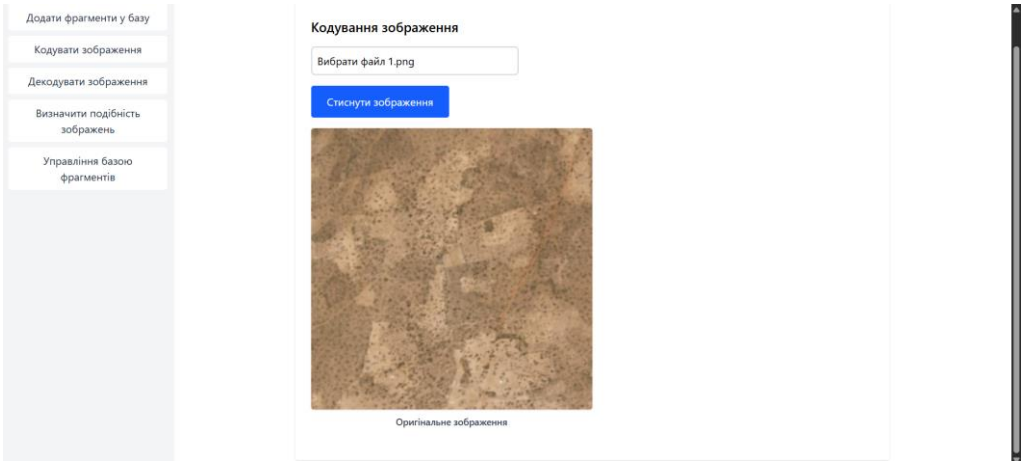


Рисунок 4.5 – Відображення поточного зображення

Для запуску процесу кодування зображення слід натиснути кнопку «Стиснути зображення», після чого зображення буде надіслано на сервер.

Після отримання відповіді від сервера праворуч з’явиться декодоване зображення, а внизу — статистика кодування: час обробки та відсоток стиснення. Також стане доступною кнопка для завантаження стиснутого зображення (рисунок 4.6).

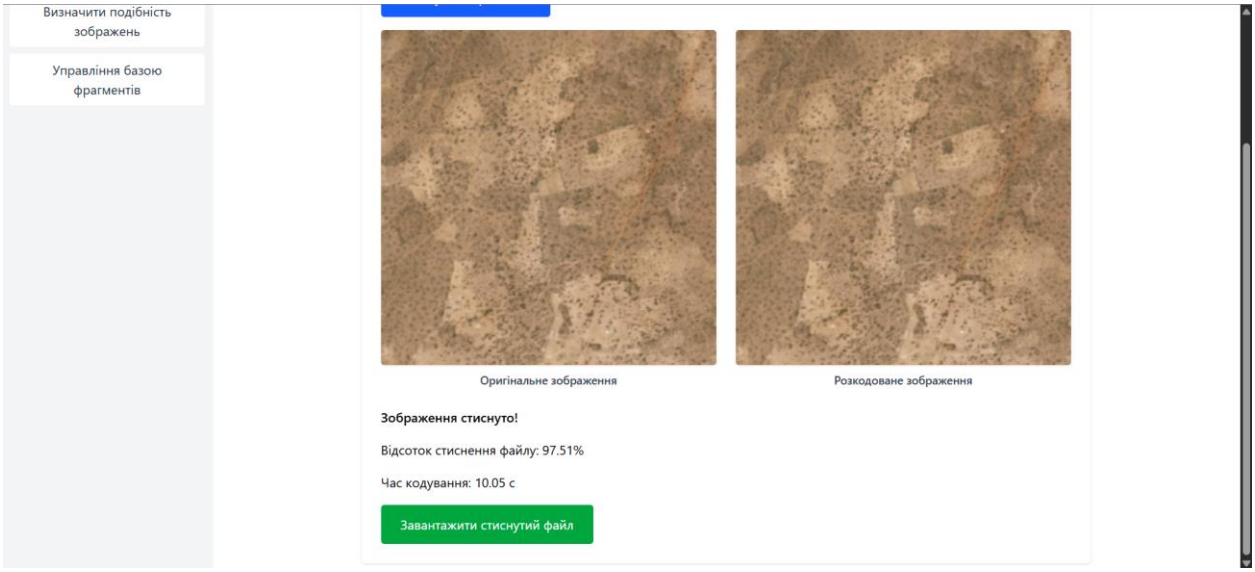


Рисунок 4.6 – Результати кодування

При натисненні на кнопку завантажити стиснутий файл відкриється додаткове вікно для завантаження файлу (рисунок 4.7).

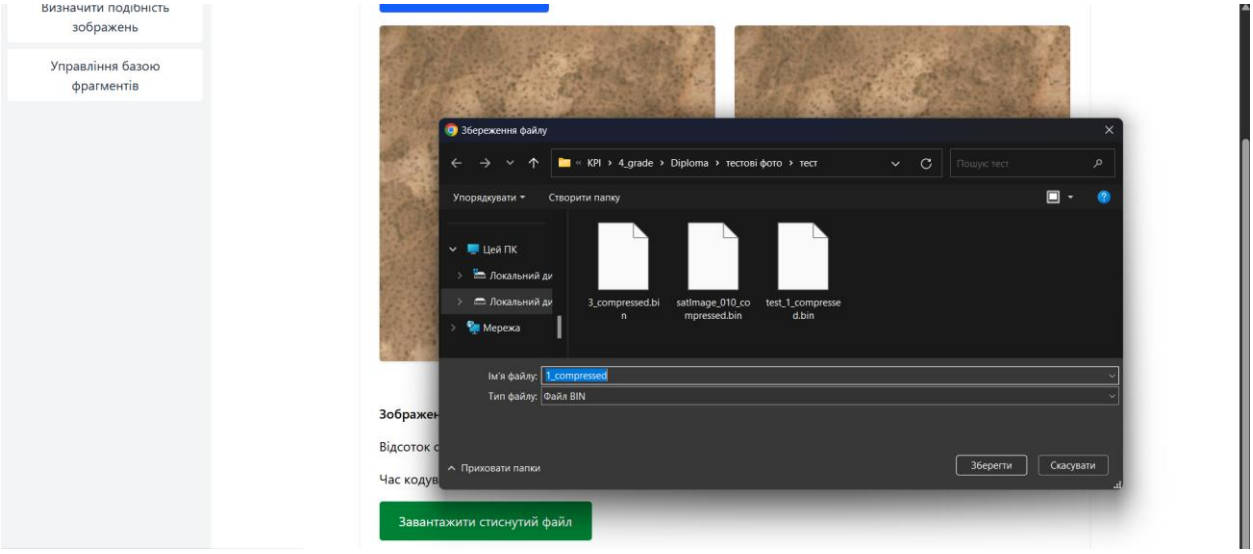


Рисунок 4.7 – Збереження стиснутого зображення

Після цього користувач може перейти на сторінку «Декодувати зображення» та завантажити файл зі стисненим зображенням (рисунок 4.8).

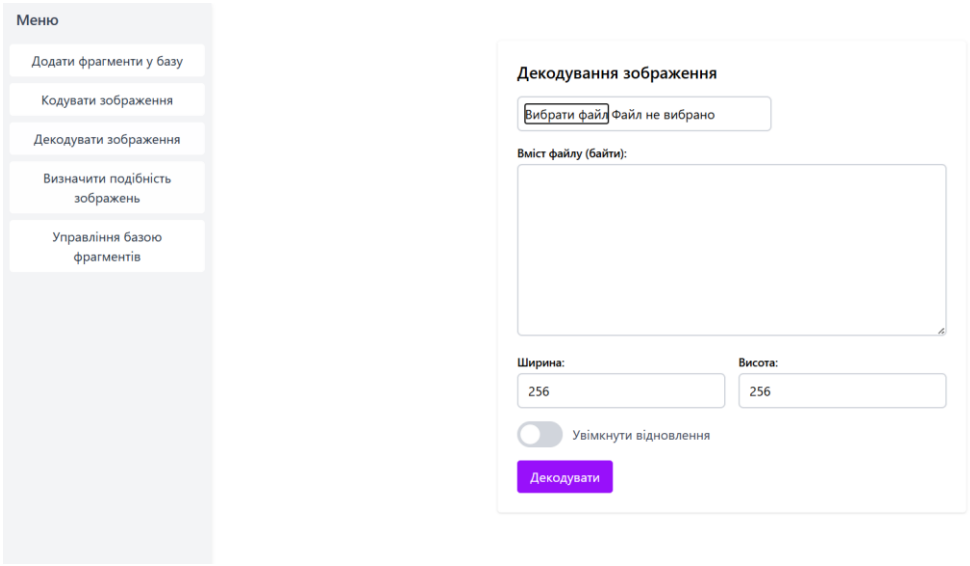


Рисунок 4.8 – Сторінка декодування зображення

Після завантаження файлу поле «Вміст файлу» відобразить закодовану інформацію з файлу, яка буде надіслана на сервер для обробки (рисунок 4.9). Наступним кроком є натискання кнопки «Декодувати», після чого користувач отримає декодоване зображення та зможе його завантажити (рисунок 4.10).

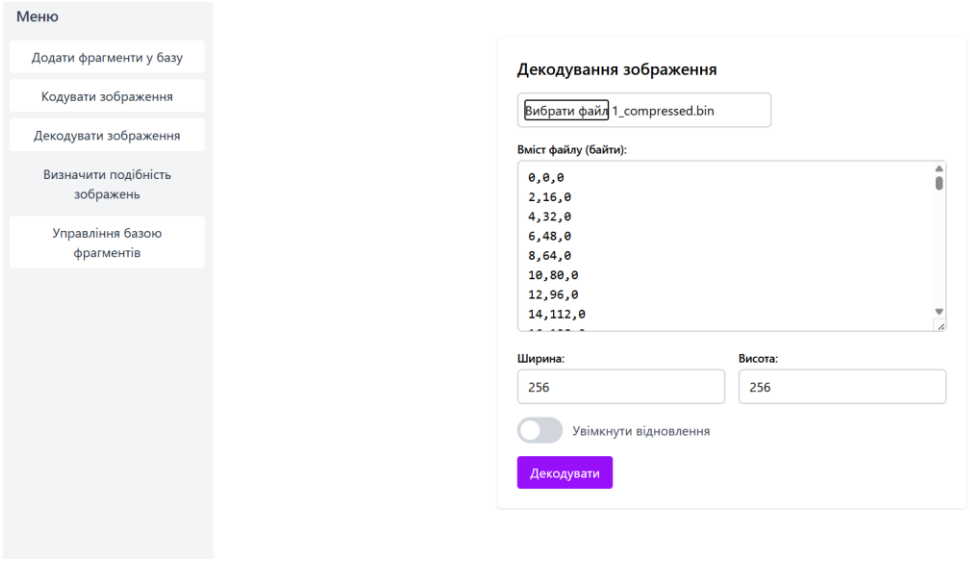


Рисунок 4.9 – Відображення вмісту файлу

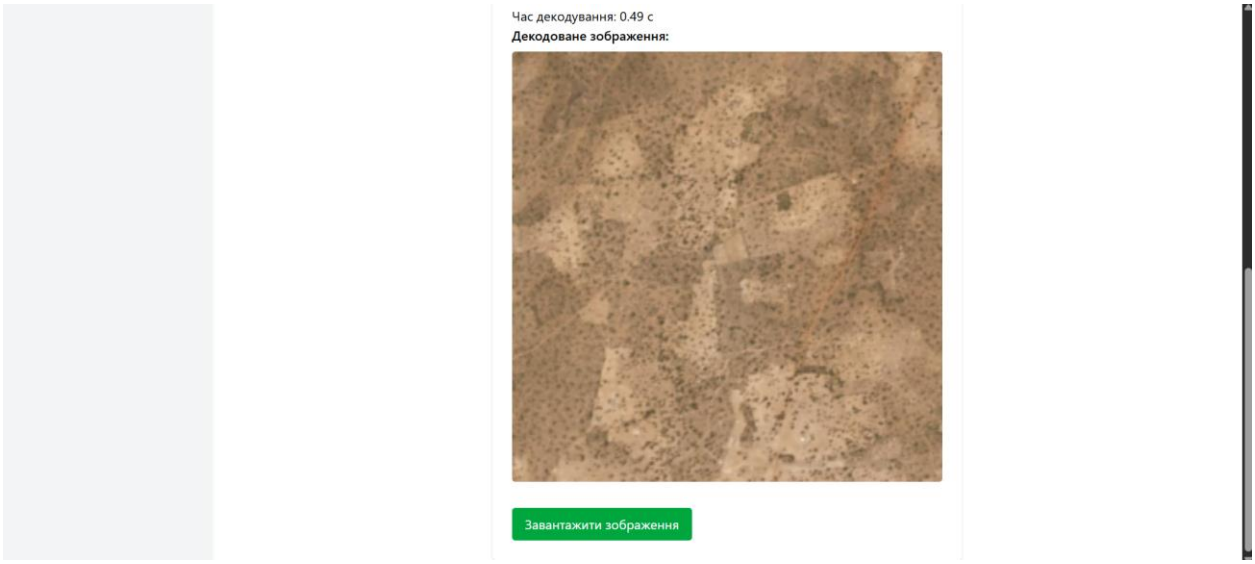


Рисунок 4.10 – Декодоване зображення

Для імітації втрати фрагментів користувач може вручну видалити кілька записів із поля «Вміст файлу» та повторно натиснути «Декодувати». У результаті система поверне зображення з відсутніми фрагментами (рисунок 4.11).

Рисунок 4.11 – Декодоване зображення з втраченими фрагментами

Для відновлення втрачених фрагментів користувачу потрібно активувати перемикач «Увімкнути відновлення» і знову натиснути кнопку «Декодувати». Після цього сервер надішле зображення з відновленими фрагментами (рисунок 4.12).

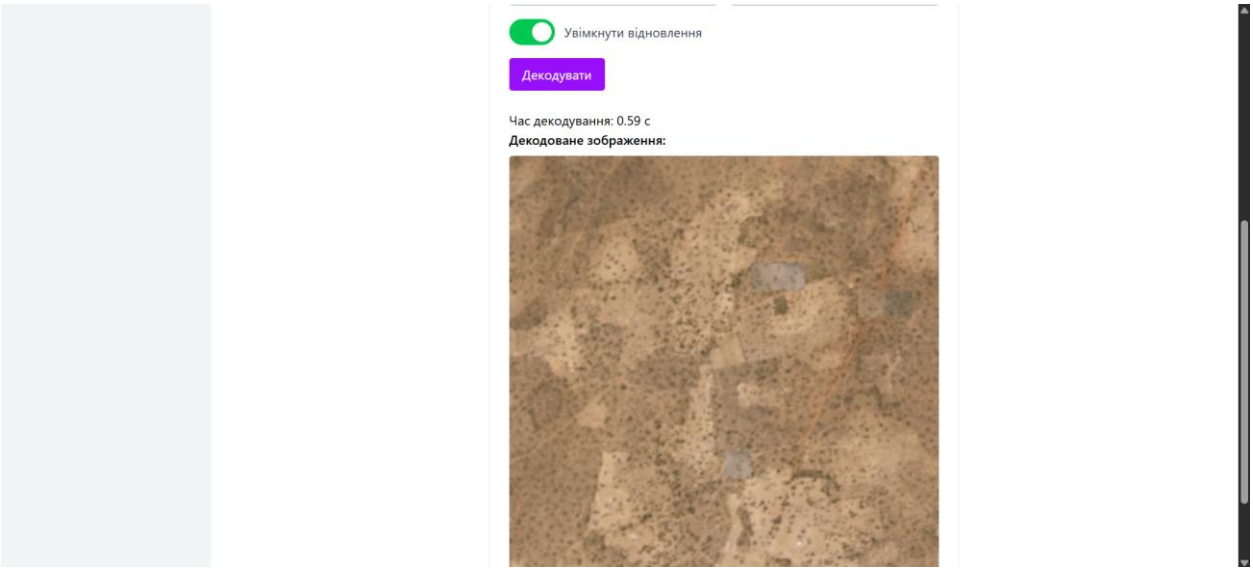


Рисунок 4.12 – Зображення з відновленими фрагментами

Щоб оцінити ефективність стиснення, слід перейти на вкладку «Визначення подібності зображень» у головному меню та завантажити в відповідні поля оригінальне та декодоване зображення. Після натискання кнопки «Порівняти» сервер поверне результат порівняння, який відобразиться у відповідному полі (рисунок 4.13).

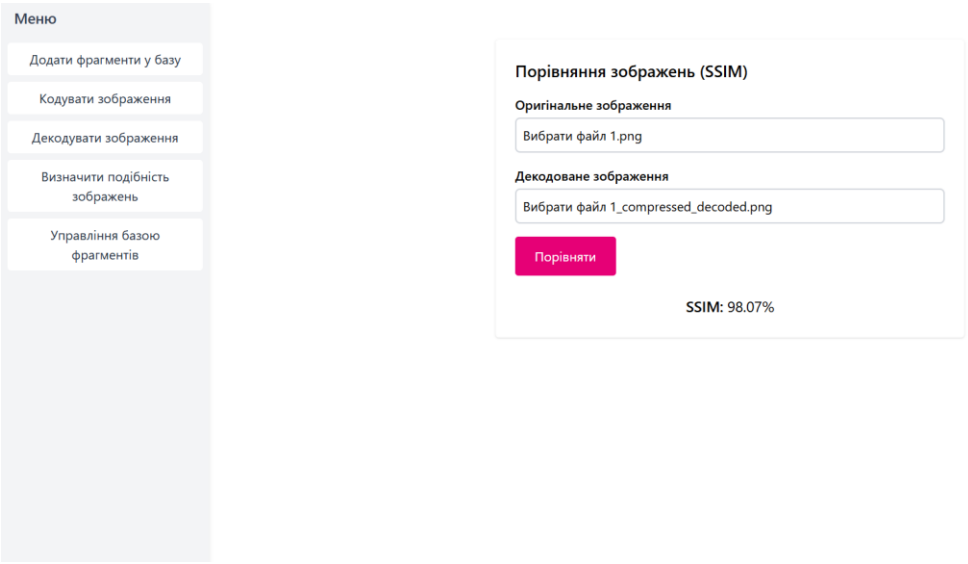


Рисунок 4.13 –Сторінка для порівняння подібності двох зображень

Користувач також має доступ до функціоналу управління базою фрагментів, де можна змінювати поріг подібності для кодування зображень. Цей параметр безпосередньо впливає на якість закодованих зображень і темп зростання бази фрагментів. На відповідній сторінці відображається поточне

значення порогу, яке можна змінити, ввівши число в межах від 0 до 1 (рисунок 4.14).

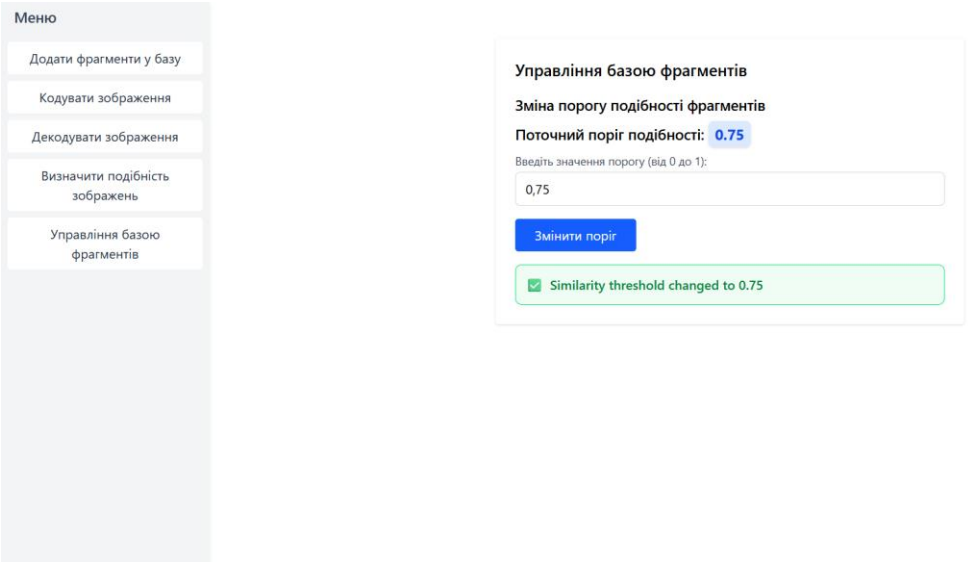


Рисунок 4.14 – Зміна порогу подібності

У разі введення некоректного значення система повідомить користувача про помилку (рисунок 4.15).

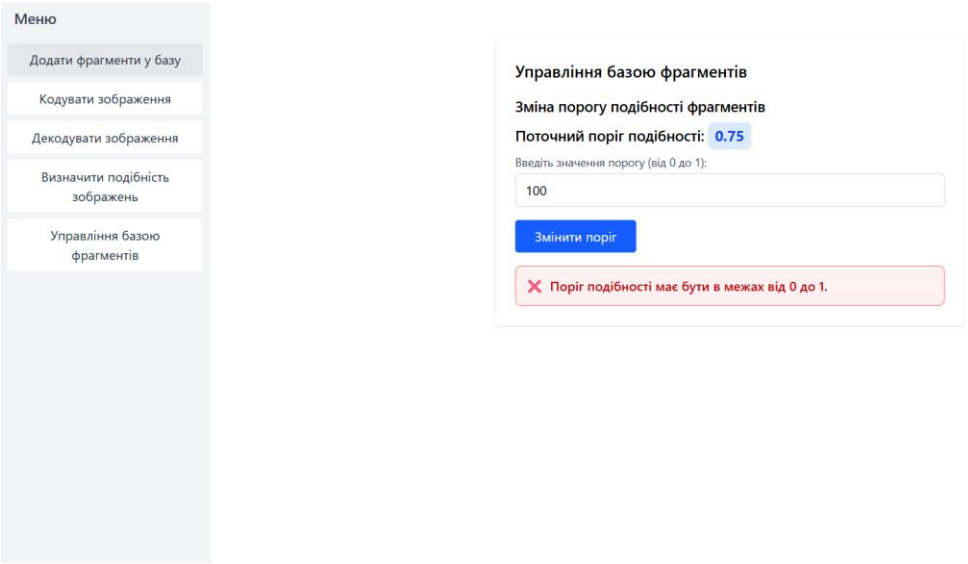


Рисунок 4.15 – Помилка при неправильному порогу подібності

Висновки до розділу

У межах цього розділу було виконано аналіз якості ПЗ з використанням ключових метрики, серед яких: час кодування зображення, відсоток стиснення, відповідність закодованого зображення до розкодованого (SSIM), а також обсяг пам’яті, необхідної для зберігання фрагментів у базі. Результати

дослідження показали, що зі збільшенням кількості фрагментів у базі спостерігається покращення якості декодування та зростання ефективності стиснення при помірному збільшенні часу обробки, що свідчить про загальне підвищення якості ПЗ.

Окрім цього, було проведено мануальне тестування програмного забезпечення відповідно до сформованих тест-кейсів, що дозволило переконатися в стабільності та функціональній повноті реалізованої системи. Також було створено детальне керівництво користувача, яке охоплює усі основні сценарії взаємодії з інтерфейсом програми, включно з випадками кодування, декодування, імітації втрат даних та відновлення зображення. У сукупності ці заходи підтверджують працездатність та високу якість розробленого рішення.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Програмне забезпечення для кодування зображень, розроблене в межах дипломного проєкту, є відкритим і доступним для розгортання будь-якими користувачами відповідно до власних потреб. Вихідний код поділений на дві частини: бекенд та фронтенд, кожна з яких зберігається в окремому репозиторії. Застосунок спроєктовано з урахуванням роздільного розгортання цих компонентів, а також із необхідністю попереднього налаштування бази даних та сховища даних.

Код програмного забезпечення оптимізований для використання хмарної інфраструктури Google Cloud Platform (GCP), зокрема таких сервісів як BigQuery та Cloud Storage (Bucket). У зв'язку з цим, для розгортання застосунку на інших платформах хмарних сервісів (наприклад, AWS або Azure) необхідно адаптувати модуль GCP Utils, змінюючи відповідні методи для взаємодії з іншими середовищами.

У випадку використання GCP, першим кроком є створення нового проєкту, у межах якого здійснюватиметься розгортання. На рисунку 5.1 представлено інтерфейс зі списком створених проєктів у Google Cloud.

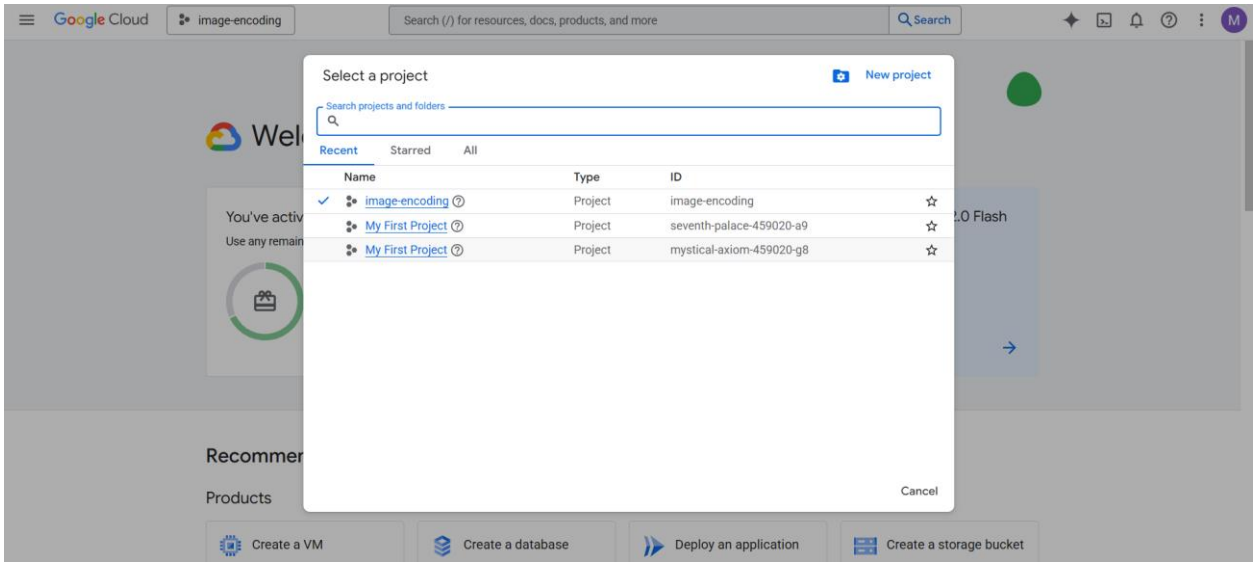


Рисунок 5.1 – Створений проєкт на платформі Google Cloud

Наступним кроком буде створення сервісного акаунта з відповідними правами доступу читання та запис у BigQuery, а також розгортання контейнерів у Artifact Registry (рисунок 5.2).

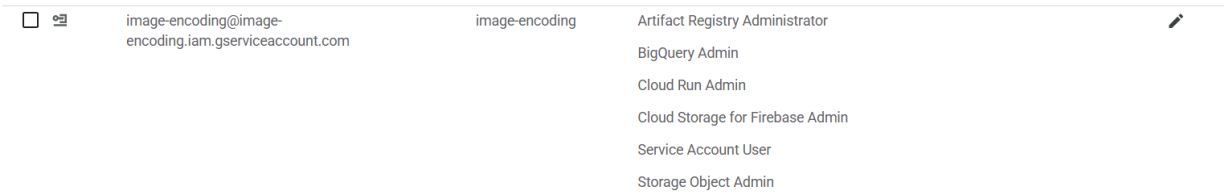


Рисунок 5.2 – Створений сервісний акаунт для застосунку

Далі йде Створення таблиці у BigQuery для зберігання фрагментів зображень зі схемою, що відповідає структурі даних, які використовуються у застосунку (рисунок 5.3).

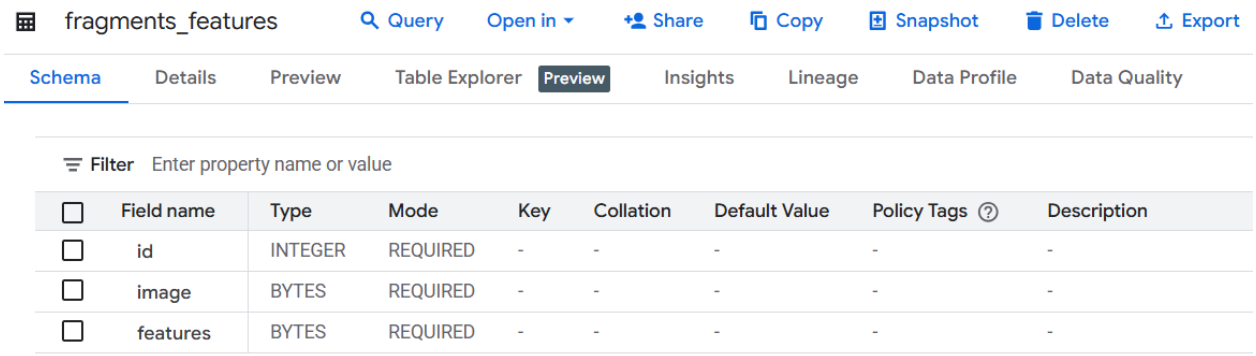


Рисунок 5.3 – Схема таблиці у BigQuery для збереження фрагментів

Після цього необхідно контейнеризувати бекенд-частину за допомогою наданого Dockerfile та створення відповідного Docker-образу. Цей контейнер можна розгорнути через Cloud Run або ж Google Kubernetes Engine. Розгорнута віртуальна машина повинна обов’язково включати в собі компонент GPU для роботи. Для забезпечення доступу до GCP-сервісів слід передати контейнеру ключ сервісного акаунта через змінну середовища (ENV).

Далі потрібно провести контейнеризацію фронтенд-частини, розробленої на React, із вказанням у файлі середовищних змінних (.env) URL-адреси розгорнутого бекенду. Це дозволяє забезпечити коректну взаємодію між клієнтською та серверною частинами застосунку.

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі Керівництво користувача.

Користувачі повинні мати можливість застосувати нові зміни до коду в своїх системах. Для спрощення розгортання та оновлення програмного забезпечення, розробленого в межах дипломного проєкту, було реалізовано процес неперервної інтеграції та розгортання (CI/CD) із використанням сервісу GitHub Actions. Це дозволяє автоматично будувати, зберігати та розгортати нові версії застосунку в хмарній інфраструктурі Google Cloud.

Процес CI запускається вручну або при потребі, після чого код збирається, контейнеризується за допомогою Docker і завантажується до приватного реєстру контейнерів. Далі створений контейнер автоматично розгортається у Cloud Run, де він стає доступним для користувачів. Всі необхідні налаштування для роботи із сервісами Google Cloud, зокрема доступ до бази даних та сховища, передаються через змінні середовища.

Таким чином, будь-які зміни до коду швидко інтегруються та публікуються без потреби ручного втручання, що значно підвищує ефективність підтримки й розвитку застосунку.

Висновки до розділу

У межах розділу було детально розглянуто процес розгортання та супроводу програмного забезпечення для кодування зображень. Застосунок має відкритий вихідний код та передбачає окреме розгортання фронтенд- і бекенд-частин, а також налаштування бази даних і сховища даних. Програмне забезпечення орієнтоване на використання хмарної інфраструктури Google Cloud, зокрема сервісів BigQuery і Cloud Storage, що забезпечує масштабованість і високу доступність системи.

Для автоматизації процесів збирання, контейнеризації та розгортання застосовано підхід безперервної інтеграції (CI) із використанням GitHub Actions. Це дозволяє мінімізувати ручну роботу при оновленні застосунку та

забезпечити швидке впровадження змін. Завдяки використанню Cloud Run, забезпечується гнучке керування ресурсами та масштабування відповідно до навантаження, включно з підтримкою GPU для обчислювально складних операцій.

Описані підходи до розгортання і супроводу гарантують зручне розширення функціональності, стабільну роботу програмного забезпечення та спрощення подальшого обслуговування системи як у середовищі Google Cloud, так і при адаптації під інші платформи.

ВИСНОВКИ

У ході виконання дипломного проєкту було спроєктовано та розроблено програмне забезпечення для кодування зображень з використанням глибинних нейронних мереж. На початковому етапі було проведено детальний аналіз предметної області, зокрема проблематики стиснення растрових зображень. У результаті аналізу було виявлено потенційні напрями для вдосконалення існуючих підходів. Як основу для реалізації було обрано алгоритм, що ґрунтується на подібності фрагментів, з метою його покращення та розширення функціональних можливостей.

На наступному етапі було сформульовано функціональні та нефункціональні вимоги до програмного забезпечення. Зокрема, передбачено використання обчислювальних ресурсів GPU для підвищення швидкодії алгоритму, оптимізацію збереження фрагментів у сховищі, а також можливість часткового відновлення інформації з пошкодженого або неповного файлу. Було проведено економічну оцінку проєкту із застосуванням методу Use Case Points, що дозволило визначити обсяг робіт, трудомісткість, вартість та ресурсні потреби розробки.

Виконано розробку алгоритму для відновлення втрачених елементів зображення та створення програмного забезпечення для демонстрації роботи алгоритму. Архітектура програмного забезпечення описана за допомогою діаграм компонентів, послідовностей і моделі C4. Завершений застосунок було контейнеризовано та розгорнуто в хмарному середовищі Google Cloud Console. Це надало можливість провести експериментальні дослідження, зокрема — оцінку функціональності алгоритму та порівняльний аналіз основних метрик якості, таких як ступінь стиснення, швидкодія та візуальна якість зображень. В результаті проведених експериментів встановлено, що розроблений алгоритм має високий показник стиснення при високому показнику подібності SSIM більше 98%.

Таким чином, усі поставлені в межах дипломного проєкту задачі, включаючи як дослідницьку, так і прикладну частини, були повністю виконані.

Запропоноване програмне забезпечення показало високу ефективність у контексті потенційної інтеграції в системи передачі зображень для безпілотних літальних апаратів. Алгоритм забезпечує високий рівень стиснення, стійкість до часткових втрат даних та абсолютну конфіденційність, адже без бази фрагментів зображення не підлягає відновленню. Це робить розроблене програмне забезпечення перспективним для використання в умовах обмежених каналів зв’язку та високих вимог до безпеки переданих даних.

Подальший розвиток проєкту доцільно здійснювати у напрямі підвищення продуктивності алгоритму для можливості обробки відеопотоків у реальному часі, а також інтеграції розробленої системи в embedded-платформи, що дозволить її ефективне використання в апаратному забезпеченні безпілотників та інших пристроїв з обмеженими обчислювальними ресурсами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Колесніков Д. Растрова та векторна графіка: особливості та переваги [Електронний ресурс]. – Режим доступу: https://brainlab.com.ua/blog/rastrova-ta-vektorna-grafikaosoblivosti-ta-perevagi (дата звернення: 17.04.2025).

2. Mishra D., Singh S. K., Singh R. K. Deep architectures for image compression: a critical review // Signal Processing. – 2022. – Vol. 191. – P. 108346. – DOI: https://doi.org/10.1016/j.sigpro.2021.108346 (дата звернення: 17.04.2025).

3. Kaur R., Choudhary P. A review of image compression techniques // International Journal of Computer Applications. – 2016. – Vol. 142, No. 1.

4. AI Image Compressor [Електронний ресурс]. – Режим доступу: https://ai.nero.com/image-compressor (дата звернення: 16.04.2025).

5. VanceAI Image Compressor [Електронний ресурс]. – Режим доступу: https://vanceai.com/image-compressor/ (дата звернення: 16.04.2025).

6. Portianyi I., Pospielova K., Oliinyk Y. Encoding raster images based on fragment similarity // Herald of Khmelnytskyi National University. – 2021. – T. 303, № 6. – С. 73–80. – DOI: https://doi.org/10.31891/2307-5732-2021-303-6-73-80 (дата звернення: 16.04.2025).

7. JPG, PNG and BMP image compression using discrete cosine transform / R. A. Hamzah та ін. // TELKOMNIKA (Telecommunication Computing Electronics and Control). – 2021. – Vol. 19, No. 3. – P. 1010. – DOI: https://doi.org/10.12928/telkomnika.v19i3.14758 (дата звернення: 17.04.2025).

8. Maayan G. D. AI-Based image compression: the state of the art [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/data-science/ai-based-image-compression-the-state-of-the-art-fb5aa6042bfa (дата звернення: 16.04.2025).

9. Ungureanu V.-I., Negirla P., Korodi A. Image-Compression techniques: classical and “region-of-interest-based” approaches presented in recent papers // Sensors. – 2024. – Vol. 24, No. 3. – P. 791. – DOI: https://doi.org/10.3390/s24030791 (дата звернення: 17.04.2025).

10. So-In C. Understanding JPEG and Applications [Електронний ресурс] // Department of Computer Science and Engineering. – 2004. С. 5–12 – Режим доступу:https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=949f102f84bdecb6bb6fe5a9111bc9b56e6d6a85 (дата звернення: 27.04.2025).

11. Clemmons R. K. Project estimation with use case points // CROSSTALK – The Journal of Defense Software Engineering. – 2006. – P. 18–22.

12. Wang Z., Bovik A., Sheikh H. Image quality assessment: from error visibility to structural similarity // IEEE Transactions on Image Processing. – 2004. – Vol. 13, No. 4. – P. 600–612.

13. Approximate nearest neighbor in Python [Електронний ресурс]. – Режим доступу: https://github.com/spotify/annoy (дата звернення: 03.06.2025).

14. Hristov H. Introduction to K-D trees [Електронний ресурс] // Baeldung CS. – Режим доступу: https://www.baeldung.com/cs/k-d-trees (дата звернення: 03.06.2025).

15. Satellite images for road segmentation [Електронний ресурс] // Kaggle. – Режим доступу: https://www.kaggle.com/datasets/sanadalali/satellite-images-for-road-segmentation (дата звернення: 05.06.2025).

ДОДАТОК А. ЗВІТ ПОДІБНОСТІ

StrikePlagiarism.com

Дата звіту 6/5/2025
Дата редагування 6/5/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
ІП_13_Музичук_ПЗ
Автор Науковий керівник / Експерт
ІП_13_МузичукОлійник Ю.О.
підрозділ
ФІОТ, К-а інформатики та програмної інженерії

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

8.13%
8.13%
КП 1

10
Довжина фрази для коефіцієнта подібності 2

11273
Кількість слів

86231
Кількість символів

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ КОДУВАННЯ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ГЛИБИННИХ НЕЙРОННИХ МЕРЕЖ

Текст програми

КПІ.ІП-1322.045490.02.81

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юрій ОЛІЙНИК

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Віталій МУЗИЧУК

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду
Бекенд частина застосунку
https://github.com/muzy4ukV/image-encode-api.git

Фронтенд частина застосунку
https://github.com/muzy4ukV/image-encoding-web.git

Файл encoder.py
Реалізація функціональної задачі кодування, декодування й відновлення зображення

```
import cv2
import os
from typing import List

import numpy as np
import tensorflow as tf
from numpy.lib.stride_tricks import as_strided
from skimage.metrics import structural_similarity as ssim_metric
from scipy.spatial import KDTree

from gcp_utils.bigquery_db import BigQueryDB
from fragment import Fragment
from time import time
from decorators import timing


class Encoder:
    def __init__(self):
        self.ssim_threshold = 0.8
        self.model = tf.keras.applications.ResNet50(weights='imagenet',
input_shape=(224, 224, 3))
        self.kernel_size = 16
        self.step_size = 8
        self.db = BigQueryDB()

    def fill_db(self, dir_path: str):
        checkpoints = [20000, 50000, 100000, 150000, 200000]
        check_index = 0
        for file_path in os.listdir(dir_path):
            if not file_path.endswith('.png') and not
file_path.endswith('.jpeg') and not file_path.endswith('.jpg'):
                continue

            print(f'Processing image {file_path}')
            img = cv2.imread(os.path.join(dir_path, file_path))
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            fragments = self.extract_fragments(img, self.kernel_size,
self.step_size)

            print(f'Fragments count: {len(fragments)}')

            prep_fragments = self.prepare_fragments(fragments)

            start_time = time()

            for fragment in prep_fragments:
```

```
                self.db.add_fragment(fragment)

            print(f'Fragments adding time: {time() - start_time}')

            if len(self.db.fragments) >= checkpoints[check_index] and
self.db_type == 'file':

self.db.save_results(path_to_save=f"fragments_base/frag_count_{checkpoints[ch
eck_index]}.npy")
                check_index += 1

    def add_fragments_from_img(self, img: np.array):
        fragments = self.extract_fragments(img, self.kernel_size,
self.step_size)
        prep_fragments = self.prepare_fragments(fragments)

        new_fragments = []
        if not self.db.is_empty():
            for fragment in prep_fragments:
                similar_fragment_id =
self.db.find_similar_fragment_id(fragment.features)
                similar_fragment_img =
self.db.get_fragment_by_id(similar_fragment_id)['image']
                similarity = self.get_ssim(fragment.image,
similar_fragment_img)

                if similarity < self.ssim_threshold:
                    new_fragments.append(fragment)
        else:
            new_fragments = prep_fragments

        print(f'All fragments from image: {len(fragments)}')
        print(f'New unique fragments count: {len(new_fragments)}')

        start_time = time()
        status = self.db.add_fragments(new_fragments)
        print(f"Image fragments adding to bq time: {time() - start_time}")

        return status, len(new_fragments)

    @timing("Encoding time")
    def encode(self, img: np.array) -> bytes:
        fragments = self.extract_fragments(img, self.kernel_size,
self.step_size)
        prepared_fragments = self.prepare_fragments(fragments)

        print(f'Fragments count: {len(fragments)}')

        encoded_fragment_info = []  # List of tuples (fragment_id, x, y)
        fragment_matches = []  # List of tuples (fragment, fragment_id,
similarity_score)
        reused_fragments = 0

        for fragment in prepared_fragments:
            if self.db.is_empty():
                new_fragment_id = self.db.add_fragment(fragment, flag=True)
                fragment_matches.append((fragment, new_fragment_id, 1))
                continue

            matched_fragment_id =
self.db.find_similar_fragment_id(fragment.features)
            matched_fragment_image =
self.db.get_fragment_by_id(matched_fragment_id)['image']
```

```
            similarity_score = self.get_ssim(fragment.image,
matched_fragment_image)

            if similarity_score > self.ssim_threshold:
                reused_fragments += 1
                fragment_matches.append((fragment, matched_fragment_id,
similarity_score))
            else:
                new_fragment_id = self.db.add_fragment(fragment, flag=True)
                fragment_matches.append((fragment, new_fragment_id, 1))

        # Filter and sort similarity data
        fragment_matches.sort(key=lambda x: x[2], reverse=True)

        # Create a mask to track occupied areas
        mask = np.zeros(img.shape[:2], dtype=bool)

        # Add fragments to encoded_fragment_info
        for fragment, fragment_id, _ in fragment_matches:
            x, y = fragment.x, fragment.y
            h, w = self.kernel_size, self.kernel_size

            # Clip fragment coordinates to fit image boundaries
            x_start, y_start = max(0, x), max(0, y)
            x_end, y_end = min(img.shape[1], x + w), min(img.shape[0], y + h)

            # Check if the fragment overlaps with the occupied area
            if np.any(mask[y_start:y_end, x_start:x_end]):
                continue

            # Update the mask
            mask[y_start:y_end, x_start:x_end] = True

            # Add the fragment to the encoded data
            encoded_fragment_info.extend([fragment_id, x, y])

        # Convert encoded_fragment_info to bytes and return
        encoded_bytes = np.array(encoded_fragment_info,
dtype=np.uint32).tobytes()

        # Update fragment tree
        self.db.build_tree()

        print(f'Encoded data size: {len(encoded_bytes)}')
        print(f'Reused fragments count: {reused_fragments}')
        print(f'Reuse ratio: {reused_fragments / len(fragment_matches) *
100:.2f}%')

        return encoded_bytes

    @timing("Decoding time")
    def decode(self, compressed_data: bytes, image_shape: tuple,
restore_image: bool = False) -> np.array:
        # Decompress the encoded fragment info
        decoded_array = np.frombuffer(compressed_data, dtype=np.uint32)

        # Parse fragment info into (fragment_id, x, y) tuples
        decoded_fragments = [
            tuple(decoded_array[i:i + 3])
            for i in range(0, len(decoded_array), 3)
        ]

        fragments = []
        for fragment_id, x, y in decoded_fragments:
```

```
            fragment = self.db.get_fragment_by_id(fragment_id)
            fragment_image = cv2.resize(fragment['image'], (self.kernel_size,
self.kernel_size))
            fragments.append(Fragment(img=fragment_image,
feature=fragment['feature'], x=x, y=y))

        # Reconstruct the original image from fragments
        reconstructed_image = self.reconstruct_image(fragments, image_shape,
restore_image)
        return reconstructed_image

    @tf.function
    def resize_and_predict(self, images: tf.Tensor) -> tf.Tensor:
        with tf.device('/GPU:0'):
            resized_images = tf.image.resize(images, [224, 224])
            return self.model(resized_images)

    @timing("Feature extraction from fragments")
    def prepare_fragments(self, fragments: list) -> list:
        batch_size = 64

        # Transform fragments into a tensor
        images_np = np.array([fragment.image for fragment in fragments])
        images_tensor = tf.convert_to_tensor(images_np, dtype=tf.float32)

        # Prepare fragments in batches
        num_fragments = len(fragments)
        prepared_fragments = []

        for start in range(0, num_fragments, batch_size):
            end = min(start + batch_size, num_fragments)
            batch_images = images_tensor[start:end]

            # Get features from CNN model
            batch_features = self.resize_and_predict(batch_images)
            batch_features =
batch_features.numpy().reshape(batch_features.shape[0], -1)

            for i, features in enumerate(batch_features):
                original_fragment = fragments[start + i]
                prepared_fragments.append(
                    Fragment(
                        img=original_fragment.image,
                        feature=features,
                        x=original_fragment.x,
                        y=original_fragment.y
                    )
                )

        return prepared_fragments

    @staticmethod
    def extract_fragments(image: np.ndarray, kernel_size: int, step_size:
int) -> List[Fragment]:
        height, width = image.shape[:2]

        # Check for valid parameters
        if not (2 <= kernel_size <= min(height, width)):
            raise ValueError(f"Invalid kernel size: {kernel_size}. Must be
between 2 and min(height, width).")
        if not (1 <= step_size <= kernel_size):
            raise ValueError(f"Invalid step size: {step_size}. Must be
between 1 and kernel_size.")
```

```
        # Create a view of the image with the desired fragment shape and
strides
        fragment_h, fragment_w, channels = kernel_size, kernel_size, 3
        num_fragments_y = (height - fragment_h) // step_size + 1
        num_fragments_x = (width - fragment_w) // step_size + 1

        fragment_shape = (num_fragments_y, num_fragments_x, fragment_h,
fragment_w, channels)
        strides = (
            image.strides[0] * step_size,
            image.strides[1] * step_size,
            image.strides[0],
            image.strides[1],
            image.strides[2]
        )
        fragments_view = as_strided(image, shape=fragment_shape,
strides=strides)

        fragments = [
            Fragment(img=fragments_view[y, x], x=x * step_size, y=y *
step_size)
            for y in range(num_fragments_y)
            for x in range(num_fragments_x)
        ]

        return fragments

    def reconstruct_image(self, fragments: List[Fragment], image_shape:
tuple,
                          should_restore_image: bool) -> np.ndarray:
        """Reconstructs an image from fragments."""
        height, width = self._adjust_image_dimensions(image_shape)
        reconstructed_image = np.zeros((height, width, 3), dtype=np.uint8)
        mask = np.zeros((height, width), dtype=bool)

        # Prepare fragment dictionary and blend initial fragments into the
image
        fragment_dict = self._populate_fragment_dict(fragments,
reconstructed_image, mask)

        # Calculate coordinates of empty fragments
        empty_fragments = self._calculate_empty_fragments(mask, height,
width)

        if not empty_fragments:
            return reconstructed_image

        if should_restore_image:
            fragments_coords, fragments_features, fragments_images =
self._prepare_kdtree_data(fragment_dict)
            tree = KDTree(fragments_coords)

            for y, x in empty_fragments:
                self._process_empty_fragment(
                    y, x, height, width, fragments_coords,
fragments_features, fragments_images, tree,
                    reconstructed_image
                )

        return reconstructed_image

    @staticmethod
    def get_ssim_for_fragments(img1: np.ndarray, img2: np.ndarray) -> float:
        img1 = img1.astype(np.float32) / 255.0
```

```
        img2 = img2.astype(np.float32) / 255.0

        return ssim_metric(img1, img2, channel_axis=2, win_size=3,
data_range=1.0)

    # Helper Functions
    def _adjust_image_dimensions(self, image_shape: tuple) -> tuple:
        height = image_shape[0] - (image_shape[0] % self.kernel_size)
        width = image_shape[1] - (image_shape[1] % self.kernel_size)
        return height, width

    def _populate_fragment_dict(self, fragments: List[Fragment],
reconstructed_image: np.ndarray,
                                mask: np.ndarray) -> dict:
        fragment_dict = {}
        for fragment in fragments:
            x, y = int(fragment.x), int(fragment.y)
            frag_height, frag_width, _ = fragment.image.shape
            reconstructed_image[y:y + frag_height, x:x + frag_width] =
fragment.image
            mask[y:y + frag_height, x:x + frag_width] = True
            fragment_dict[(y, x)] = (fragment.features, fragment.image)
        return fragment_dict

    def _calculate_empty_fragments(self, mask: np.ndarray, height: int,
width: int) -> list:
        return [
            (y, x)
            for y in range(0, height - self.kernel_size + 1,
self.kernel_size)
            for x in range(0, width - self.kernel_size + 1, self.kernel_size)
            if not mask[y:y + self.kernel_size, x:x + self.kernel_size].any()
        ]

    def _prepare_kdtree_data(self, fragment_dict: dict) -> tuple:
        fragments_coords = np.array(list(fragment_dict.keys()))
        fragments_features = np.array([v[0] for v in fragment_dict.values()])
        fragments_images = np.array([v[1] for v in fragment_dict.values()])
        return fragments_coords, fragments_features, fragments_images

    def _process_empty_fragment(
            self, y: int, x: int, height: int, width: int, fragments_coords:
np.ndarray,
            fragments_features: np.ndarray, fragments_images: np.ndarray,
tree: KDTree, reconstructed_image: np.ndarray
    ):
        NUM_NEIGHBORS_EDGE = 5
        NUM_NEIGHBORS_INTERNAL = 8

        is_edge = (x == 0 or y == 0 or x + self.kernel_size >= width or y +
self.kernel_size >= height)
        num_neighbors = NUM_NEIGHBORS_EDGE if is_edge else
NUM_NEIGHBORS_INTERNAL

        # Find nearest neighbors
        dists, idxs = tree.query((y, x), k=num_neighbors)
        neighbor_features = fragments_features[idxs]
        neighbor_images = fragments_images[idxs]

        # Predict features for the missing fragment
        predicted_feature = np.mean(neighbor_features, axis=0)
        candidates = self.db.find_k_similar_fragments(predicted_feature,
k=10)
```

```
        best_candidate, best_score = self._find_best_candidate(candidates,
neighbor_images)

        # Fill the fragment
        reconstructed_image[y:y + self.kernel_size, x:x + self.kernel_size] =
best_candidate['image']

    def _find_best_candidate(self, candidates: list, neighbor_images:
np.ndarray) -> tuple:
        best_candidate = candidates[0]
        best_score = 0
        for candidate in candidates:
            scores = [
                self.get_ssim_for_fragments(candidate['image'],
neighbor_image)
                for neighbor_image in neighbor_images
            ]
            if scores:
                avg_ssim = np.mean(scores)
                if avg_ssim > best_score:
                    best_score = avg_ssim
                    best_candidate = candidate
        return best_candidate, best_score

    def blend_fragments(self, fragments: List[Fragment], image: np.ndarray,
img_shape: tuple) -> np.ndarray:
        height, width = img_shape
        kernel_size = max(self.kernel_size // 2, 5)
        kernel = np.ones((kernel_size, kernel_size), np.uint8)

        for fragment in fragments:
            x, y = int(fragment.x), int(fragment.y)
            h, w, _ = fragment.image.shape

            x_start, y_start = max(0, x), max(0, y)
            x_end, y_end = min(width, x + w), min(height, y + h)

            region_img = image[y_start:y_end, x_start:x_end]
            region_frag = fragment.image[y_start - y:y_end - y, x_start -
x:x_end - x]

            edges = cv2.Canny(region_img, 100, 200)
            edges = cv2.morphologyEx(edges, cv2.MORPH_CLOSE, kernel)

            if np.mean(edges) > 0:
                blended = cv2.addWeighted(region_img, 0.5, region_frag, 0.5,
0)
                image[y_start:y_end, x_start:x_end] = blended
            else:
                image[y_start:y_end, x_start:x_end] = region_frag

        return image

    def set_ssim_threshold(self, threshold: float):
        self.ssim_threshold = threshold

    def get_ssim(self, original_img: np.array, decoded_img: np.array) ->
float:
        """
        Computes the Structural Similarity Index (SSIM) between two images.
        """
        return ssim_metric(original_img, decoded_img, multichannel=True,
win_size=7, channel_axis=2)
```

```
    def is_overlapping(self, fragment1, fragment2):
        x1, y1, w1, h1 = fragment1[1], fragment1[2], self.kernel_size,
self.kernel_size
        x2, y2, w2, h2 = fragment2[1], fragment2[2], self.kernel_size,
self.kernel_size

        if x1 >= x2 + w2 or x2 >= x1 + w1 or y1 >= y2 + h2 or y2 >= y1 + h1:
            return False

        return True
```

Файл server.py

Реалізація функціональної задачі створення API

```
import os

import uvicorn
from fastapi import UploadFile, File, Depends, HTTPException, Form, Request,
BackgroundTasks
from fastapi.responses import StreamingResponse
from fastapi import FastAPI
import io
import numpy as np
import cv2
from pydantic import BaseModel
from dotenv import load_dotenv

load_dotenv()
from contextlib import asynccontextmanager
import tensorflow as tf

from fastapi.middleware.cors import CORSMiddleware
from encoder import Encoder


@asynccontextmanager
async def lifespan(app: FastAPI):
    # □ Частина ДО yield — тут ініціалізуються ресурси
    if tf.test.gpu_device_name():
        print("Default GPU Device: {}".format(tf.test.gpu_device_name()))
    else:
        print("Please install GPU version of TF")
    encoder = Encoder()
    app.state.encoder = encoder

    yield  # ⬅ Після цього FastAPI починає обробляти HTTP-запити


app = FastAPI(lifespan=lifespan)

app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:5173"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)


def get_encoder(request: Request) -> Encoder:
    return request.app.state.encoder
```

```
def validate_image_file(file: UploadFile = File(...)) -> UploadFile:
    if not file.filename.endswith((".jpg", ".jpeg", ".png")):
        raise HTTPException(status_code=400, detail="Invalid file type")
    valid_mime_types = {"image/jpeg", "image/png"}

    if file.content_type not in valid_mime_types:
        raise HTTPException(status_code=400, detail="Invalid MIME type")
    return file


def validate_and_convert_to_nparray(file_bytes: bytes) -> np.ndarray:
    image_array = np.frombuffer(file_bytes, dtype=np.uint8)

    image = cv2.imdecode(image_array, cv2.IMREAD_COLOR)

    if image is None:
        raise HTTPException(status_code=400, detail="Corrupted or invalid
image file")

    return cv2.cvtColor(image, cv2.COLOR_BGR2RGB)


@app.post("/add-fragments/")
async def add_fragments(file: UploadFile = Depends(validate_image_file),
                        encoder: Encoder = Depends(get_encoder),
                        background_tasks: BackgroundTasks = BackgroundTasks):
    file_bytes = await file.read()

    # Validate and load image
    np_image = validate_and_convert_to_nparray(file_bytes)

    print("Image received successfully")

    adding_status, fragments_count = encoder.add_fragments_from_img(np_image)
    background_tasks.add_task(encoder.db.build_tree)
    if not fragments_count is None:
        return {
            "status": adding_status,
            "added_fragments_count": fragments_count,
            "db_fragments_count": encoder.db.get_db_size()
        }
    else:
        raise HTTPException(status_code=500, detail="Failed to add fragments
into base")


@app.post("/encode-image/")
async def encode_image(file: UploadFile = Depends(validate_image_file),
                       encoder: Encoder = Depends(get_encoder),
                       background_tasks: BackgroundTasks = BackgroundTasks):
    file_bytes = await file.read()

    image = validate_and_convert_to_nparray(file_bytes)

    print("Image received successfully")

    # Обробка
    try:
        encoded_image = encoder.encode(image)
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Encoding failed:
{str(e)}")
    # Фоново оновлюємо дерево й додаємо нові фрагменти в БД
    background_tasks.add_task(encoder.db.update_fragments)
```

```
    filename_no_ext = file.filename.split(".")[0]
    return StreamingResponse(
        io.BytesIO(encoded_image),
        media_type="application/octet-stream",
        headers={"Content-Disposition": f'attachment;
filename="{filename_no_ext}.bin"'},
    )


class ImageSize(BaseModel):
    width: int = Form()
    height: int = Form()


def get_image_size(
        width: int = Form(...),
        height: int = Form(...)) -> ImageSize:
    return ImageSize(width=width, height=height)


@app.post("/decode-image/")
async def decode_image(
        compressed_img: UploadFile = File(...),
        img_size: ImageSize = Depends(get_image_size),
        restore_image: bool = Form(False),
        encoder: Encoder = Depends(get_encoder)
):
    try:
        # width, height = image_shape.width, image_shape.height
        compressed_img = await compressed_img.read()

        # Декодуємо отримані дані
        decoded_image = encoder.decode(compressed_img, (img_size.height,
img_size.width),
                                       restore_image=restore_image)

        # Конвертуємо отримане зображення в формат, який можна відправити як
відповідь
        decoded_image = cv2.cvtColor(decoded_image, cv2.COLOR_RGB2BGR)
        _, buffer = cv2.imencode('.png', decoded_image)

        image_bytes = buffer.tobytes()

        image_io = io.BytesIO(image_bytes)
        image_io.seek(0)

        # Повертаємо як стрімінгову відповідь
        return StreamingResponse(
            image_io,
            media_type="image/png",
            headers={"Content-Disposition": "inline;
filename=decoded_image.png"}
        )

    except ValueError as ve:
        if "could not broadcast input array from shape" in str(ve):
            raise HTTPException(status_code=400, detail="Image decoding
failed: Invalid image dimensions")
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Decoding failed:
{str(e)}")


@app.post("/change-similarity-threshold/{threshold}")
```

```
def change_similarity_threshold(threshold: float, encoder: Encoder =
Depends(get_encoder)):
    # Validate the threshold range
    if not (0 <= threshold <= 1):
        raise HTTPException(status_code=400, detail="Threshold must be
between 0 and 1.")

    # Set the similarity threshold in the encoder
    encoder.set_ssim_threshold(threshold)

    # Return a JSON response
    return {"message": f"Similarity threshold changed to {threshold}",
            "similarity_threshold": encoder.ssim_threshold}


@app.get("/get-similarity-threshold/")
def get_similarity_threshold(encoder: Encoder = Depends(get_encoder)):
    return {"similarity_threshold": encoder.ssim_threshold}


@app.post("/get-ssim/")
async def get_ssim_metric(
        original_img: UploadFile = File(...),
        decoded_img: UploadFile = File(...),
        encoder: Encoder = Depends(get_encoder)
):
    # Валідація типів
    validate_image_file(original_img)
    validate_image_file(decoded_img)

    # Читання байтів
    original_bytes = await original_img.read()
    decoded_bytes = await decoded_img.read()

    # Перетворення у NumPy масив
    original_np = validate_and_convert_to_nparray(original_bytes)
    decoded_np = validate_and_convert_to_nparray(decoded_bytes)

    # Обчислення SSIM
    try:
        ssim_value = encoder.get_ssim(original_np, decoded_np)
    except ValueError as ve:
        if "Input images must have the same dimensions" in str(ve):
            raise HTTPException(status_code=400, detail="SSIM calculation
failed: Images have different dimensions")

    return {
        "status": "success",
        "ssim": ssim_value
    }

@app.get("/health/")
def health():
    return {"status": "healthy"}


@app.get("/")
async def root():
    return {"message": "Hello World"}


if __name__ == "__main__":
    port = int(os.environ.get("PORT", 8080))
    uvicorn.run("server:app", host="0.0.0.0", port=port)
```

Файл bigquery_db.py
Реалізація функціональної задачі збереження фрагментів до БД

```
from decorators import timing
import numpy as np
from annoy import AnnoyIndex
from google.cloud import bigquery
import os
import cv2
import pandas as pd
from typing import Optional
from fragment import Fragment
from .credentials import CREDENTIALS
from .bucket_utils import GCSBucketUtils
from .label_generator import LabelGenerator


class BigQueryDB:
    TABLE_CONFIG = bigquery.LoadJobConfig(
        schema=[
            bigquery.SchemaField("id", "INTEGER", "REQUIRED"),
            bigquery.SchemaField("image", "BYTES", "REQUIRED"),  # <==
ключове
            bigquery.SchemaField("features", "BYTES", "REQUIRED")
        ]
    )

    def __init__(self):
        super().__init__()
        # Завантажуємо облікові дані
        # Створюємо клієнта BigQuery
        self.client = bigquery.Client(credentials=CREDENTIALS,
project=CREDENTIALS.project_id)
        self.tree = None
        self.project_id = CREDENTIALS.project_id
        self.dataset_id = os.environ['GCP_DATASET_ID']
        self.table_id = os.environ['GCP_TABLE_ID']
        self.target_table =
self.client.get_table(f"{self.project_id}.{self.dataset_id}.{self.table_id}")
        self.fragments = {}
        self.label_generator = None
        self.prepare_fragments()
        self.buffer_fragments_ids = []

    def is_empty(self):
        return len(self.fragments) == 0

    @timing("Time preparing fragments")
    def prepare_fragments(self):
        query = f"SELECT * FROM
`{self.project_id}.{self.dataset_id}.{self.table_id}`"
        features_df = self.client.query(query).to_dataframe()

        if features_df.empty:
            print("No features found in DB.")
            return

        for i, row in features_df.iterrows():
            fragment_id = int(row['id'])

            # Декодуємо PNG байти (row['image'] — це bytes)
            image = cv2.imdecode(np.frombuffer(row['image'], dtype=np.uint8),
cv2.IMREAD_COLOR)
            # Приводимо до RGB (якщо потрібно)
```

```
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            fragment_doc = {
                'image': image,
                'feature': np.frombuffer(row['features'], dtype=np.float32)
            }
            self.fragments[fragment_id] = fragment_doc

        self.build_tree()

    def build_tree(self):
        features_dims = self.fragments[0]['feature'].shape[0]
        self.tree = AnnoyIndex(features_dims, 'euclidean')
        for i, fragment in self.fragments.items():
            self.tree.add_item(i, fragment['feature'])

        n_trees = min(100, max(10, int(np.log2(len(self.fragments))) * 5))
        self.tree.build(n_trees, n_jobs=-1)
        print("Tree was built")

    def add_fragments(self, fragments: list[Fragment]) -> Optional[str]:
        if len(fragments) == 0:
            return "No fragments to add into base"
        rows = []
        for fragment in fragments:
            rows.append({
                "id": len(self.fragments),
                "image": BigQueryDB.compress_nparr_to_bytes(fragment.image),
                "features": fragment.features.tobytes()
            })
            self.add_fragment(fragment)

        fragments_df = pd.DataFrame(rows)
        fragments_df['id'] = fragments_df['id'].astype(int)

        try:
            job = self.client.load_table_from_dataframe(fragments_df,
self.target_table,

job_config=BigQueryDB.TABLE_CONFIG)
            job.result()
            return "Successfully added fragments into base"

        except Exception as e:
            print(f"Error occurred while adding fragments to BigQuery: {e}")
            raise e

    def add_fragment(self, fragment: Fragment, flag: bool = False):
        fragment_elem = {
            'image': fragment.image,
            'feature': fragment.features
        }
        fragment_id = len(self.fragments)
        self.fragments[fragment_id] = fragment_elem
        if flag:
            self.buffer_fragments_ids.append(fragment_id)
        return fragment_id

    @timing("Time updating fragments")
    def update_fragments(self):
        if len(self.buffer_fragments_ids) == 0:
            return
        else:
            fragment_to_add = []
            for fragment_id in self.buffer_fragments_ids:
```

```
                fragment = self.fragments[fragment_id]
                fragment_to_add.append({
                    "id": fragment_id,
                    "image":
BigQueryDB.compress_nparr_to_bytes(fragment['image']),
                    "features": fragment['feature'].tobytes()
                })
            fragments_df = pd.DataFrame(fragment_to_add)
            try:
                job = self.client.load_table_from_dataframe(fragments_df,
self.target_table,

job_config=BigQueryDB.TABLE_CONFIG)
                job.result()
                print("Updated fragments loaded successfully")
                self.buffer_fragments_ids = []
            except Exception as e:
                print(f"Error occurred while adding fragments to BigQuery:
{e}")

    def get_db_size(self):
        return len(self.fragments)

    def get_fragment_by_id(self, fragment_id: int):
        return self.fragments[fragment_id]

    def find_similar_fragment_id(self, fragment_feature):
        similar_fragment_id = self.tree.get_nns_by_vector(fragment_feature,
1)[0]
        return similar_fragment_id

    def find_k_similar_fragments(self, fragment_feature, k):
        similar_fragment_ids = self.tree.get_nns_by_vector(fragment_feature,
k)
        return [self.get_fragment_by_id(fr_id) for fr_id in
similar_fragment_ids]

    @staticmethod
    def compress_nparr_to_bytes(nparr):
        # PNG-стиснення
        _, encoded_png = cv2.imencode('.png', cv2.cvtColor(nparr,
cv2.COLOR_RGB2BGR))
        return encoded_png.tobytes()
```

Файл bucket_utils.cpp

Реалізація функціональної задачі управління сховищем даних

```
import numpy as np
from google.cloud import storage
from .credentials import CREDENTIALS
import cv2
import tempfile
import os

class GCSBucketUtils:

    def __init__(self):
        self.bucket_name = os.environ.get('GCP_BUCKET_NAME')
        self.client = storage.Client(credentials=CREDENTIALS,
project=CREDENTIALS.project_id)
        self.bucket = self.client.get_bucket(self.bucket_name)

    def add_fragment(self, array: np.array, label: int):
        """
```

```
        Converts a numpy array to a PNG image, saves it with a unique name,
and uploads it to the GCS bucket.
        :param label:
        :param array: Numpy array to be converted to a PNG image.
        """

        # Create a temporary file to save the PNG
        with tempfile.NamedTemporaryFile(suffix=".png") as temp_file:
            # Convert the numpy array to a PNG file
            cv2.imwrite(temp_file.name, array)

            # Upload the file to the GCS bucket
            blob = self.bucket.blob(f"{label}.png")
            blob.upload_from_filename(temp_file.name)

    def get_fragment(self, fragment_id):
        blob = self.bucket.blob(f"{fragment_id}.png")
        content = blob.download_as_bytes()
        return content  # content — це bytes
```

Файл label_generator.py

```
class LabelGenerator:
    _instance = None  # Статичне поле для збереження єдиного екземпляру

    def __new__(cls, max_db_label=0):
        if cls._instance is None:
            cls._instance = super(LabelGenerator, cls).__new__(cls)
            cls._instance.__initialized = False
        return cls._instance

    def __init__(self, max_db_label):
        if self.__initialized:
            return
        self.__current_value = max_db_label
        self.__initialized = True

    def generate(self):
        self.__current_value += 1
        return self.__current_value
```

Файл App.jsx

Реалізація функціональної задачі відображення головної сторінки

```
import { useState } from 'react'
import MenuNavigation from './components/MenuNavigation'
import AddFragment from './components/AddFragment'
import EncodeImage from './components/EncodeImage'
import DecodeImage from './components/DecodeImage'
import ImageSimilarity from './components/ImageSimilarity'
import BaseManagement from './components/BaseManagement'

function App() {
  const [view, setView] = useState('home')

  const renderComponent = () => {
    switch (view) {
      case 'add': return <AddFragment />
      case 'encode': return <EncodeImage />
      case 'decode': return <DecodeImage />
      case 'ssim': return <ImageSimilarity />
      case 'base': return <BaseManagement />
      default: return (
```

```
        <div className="max-w-xl mx-auto mt-10 p-6 bg-blue-50 rounded-lg
shadow text-center">
          <h2 className="text-2xl font-semibold text-blue-800 mb-2">Ласкаво
просимо!</h2>
          <p className="text-blue-700">Оберіть дію в меню ліворуч, щоб
розпочати.</p>
        </div>
      )
    }
  }

  return (
    <div className="flex min-h-screen">
      {/* Sidebar Navigation */}
      <div className="w-64 bg-gray-100 p-2 shadow-md">
        <h2 className="text-lg font-semibold text-gray-700 mb-4 px-
2">Меню</h2>
        <MenuNavigation onNavigate={setView} />
      </div>

      {/* Main Content */}
      <div className="flex-1 p-2">
        {renderComponent()}
      </div>
    </div>
  )
}

export default App
```

Файл MenuNavigation.jsx

Реалізація функціональної задачі відображення меню

```
function MenuNavigation({ onNavigate }) {
  return (
    <nav className="flex flex-col gap-2">
      <button
        onClick={() => onNavigate('add')}
        className="px-4 py-2 bg-white text-gray-800 rounded hover:bg-gray-200
transition"
      >
        Додати фрагменти у базу
      </button>
      <button
        onClick={() => onNavigate('encode')}
        className="px-4 py-2 bg-white text-gray-800 rounded hover:bg-gray-200
transition"
      >
        Кодувати зображення
      </button>
      <button
        onClick={() => onNavigate('decode')}
        className="px-4 py-2 bg-white text-gray-800 rounded hover:bg-gray-200
transition"
      >
        Декодувати зображення
      </button>
      <button
        onClick={() => onNavigate('ssim')}
        className="px-4 py-2 bg-white text-gray-800 rounded hover:bg-gray-200
transition"
      >
        Визначити подібність зображень
      </button>
```

```
      <button
        onClick={() => onNavigate('base')}
        className="px-4 py-2 bg-white text-gray-800 rounded hover:bg-gray-200
transition"
      >
        Управління базою фрагментів
      </button>
    </nav>
  )
}

export default MenuNavigation
```

Файл EcnodeImage.py

Реалізація функціональної задачі інтерфейсу для кодування зображення

```
import { useState } from 'react'
import { encodeImage, decodeImage } from '../api/apiClient'

function EncodeImage(props) {
  const [selectedFile, setSelectedFile] = useState(null)
  const [previewUrl, setPreviewUrl] = useState(null)
  const [compressedData, setCompressedData] = useState(null)
  const [isLoading, setIsLoading] = useState(false)
  const [encodingTime, setEncodingTime] = useState(null)
  const [decodedImageUrl, setDecodedImageUrl] = useState(null)


  const handleFileChange = (e) => {
    const file = e.target.files[0]
    if (file && (file.type === 'image/png' || file.type === 'image/jpeg')) {
      setSelectedFile(file);
      setCompressedData(null);
      setDecodedImageUrl(null);
      setPreviewUrl(URL.createObjectURL(file));
      // Збереження ширини та висоти у localStorage
      const img = new Image()
      img.onload = () => {
        localStorage.setItem('imageWidth', img.naturalWidth.toString())
        localStorage.setItem('imageHeight', img.naturalHeight.toString())
      }
      img.src = URL.createObjectURL(file)
    } else {
      alert('Будь ласка, виберіть PNG або JPG файл')
    }
  }

  const handleSubmit = async () => {
    if (!selectedFile) {
      alert('Оберіть зображення перед кодуванням');
      return;
    }

    setIsLoading(true);
    setEncodingTime(null);
    setCompressedData(null);
    setDecodedImageUrl(null);


    const startTime = performance.now();

    try {
      const encoded_image_blob = await encodeImage(selectedFile);
      setCompressedData(encoded_image_blob);
```

```
      const decoded_image_blob = await decodeImage(encoded_image_blob,
localStorage.getItem('imageWidth'), localStorage.getItem('imageHeight'))
      setDecodedImageUrl(URL.createObjectURL(decoded_image_blob))
    } catch (error) {
      alert(error.message);
    } finally {
      setEncodingTime(performance.now() - startTime)
      setIsLoading(false);
    }
  };

  return (
    <div className="w-full max-w-4xl mx-auto mt-10 p-6 bg-white rounded
shadow">
      <h3 className="text-xl font-semibold mb-4">Кодування зображення</h3>

      <input
        type="file"
        accept="image/png, image/jpeg"
        onChange={handleFileChange}
        className="mb-4 p-2 border-2 border-gray-300 rounded-md cursor-
pointer hover:border-blue-500 focus:ring-2 focus:ring-blue-500"
      />
      <div className="mb-4">
        <button
          onClick={handleSubmit}
          disabled={isLoading}
          className="bg-blue-600 text-white px-6 py-3 rounded hover:bg-blue-
700 disabled:opacity-50"
        >
          {isLoading ? 'Обробка...' : 'Стиснути зображення'}
        </button>
      </div>

      {previewUrl && (
        <div className="mb-6">
          <div className="flex flex-nowrap gap-6">
            {/* Оригінальне зображення */}
            <div className="w-1/2 text-center">
              <img
                src={previewUrl}
                alt="Оригінал"
                className="w-full rounded shadow"
              />
              <p className="mt-2 text-sm text-gray-700 font-
medium">Оригінальне зображення</p>
            </div>

            {/* Розкодоване зображення */}
            {decodedImageUrl && (
              <div className="w-1/2 text-center">
                <img
                  src={decodedImageUrl}
                  alt="Розкодоване"
                  className="w-full rounded shadow"
                />
                <p className="mt-2 text-sm text-gray-700 font-
medium">Розкодоване зображення</p>
              </div>
            )}
          </div>
        </div>
      )}
```

```
      {compressedData && (
        <div className="mt-6">
          <h4 className="font-medium mb-2">Зображення стиснуто!</h4>

          {/* Відсоток стиснення */}
          {compressedData && selectedFile && (
            <p className="mt-4">
              Відсоток стиснення файлу:{' '}
              {((1 - compressedData.size / selectedFile.size) *
100).toFixed(2)}%
            </p>
          )}

          {/* Час кодування */}
          {encodingTime !== null && (
            <p className="mt-4">
              Час кодування: {(encodingTime / 1000).toFixed(2)} с
            </p>
          )}

          {/* Кнопка для завантаження — з відступом зверху */}
          <div className="mt-4">
            <a
              href={URL.createObjectURL(compressedData)}
              download={`${selectedFile.name.split('.')[0]}_compressed.bin`}
            >
              <button className="bg-green-600 text-white px-6 py-3 rounded
hover:bg-green-700">
                Завантажити стиснутий файл
              </button>
            </a>
          </div>
        </div>
      )}
    </div>
  );

}

export default EncodeImage
```

Файл DecodeImage.jsx

Реалізація функціональної задачі інтерфейсу для декодування зображення

```
import { useState } from 'react'
import { decodeImage } from '../api/apiClient'



function DecodeImage() {
  const [selectedFile, setSelectedFile] = useState(null)
  const [fileText, setFileText] = useState('')
  const [decodedImageUrl, setDecodedImageUrl] = useState(null)
  const [isLoading, setIsLoading] = useState(false)
  const [width, setWidth] = useState(localStorage.getItem('imageWidth'))
  const [height, setHeight] = useState(localStorage.getItem('imageHeight'))
  const [restoringTime, setDecodingTime] = useState(null)
  const [enableRestoration, setEnableRestoration] = useState(false)



  const handleFileChange = async (e) => {
    const file = e.target.files[0]
```

```
    if (!file) return

    setSelectedFile(file);
    const arrayBuffer = await file.arrayBuffer()
    // Просто читаємо байти без розпакування
    const uint32Array = new Uint32Array(arrayBuffer)
    const decodedFragments = []
    for (let i = 0; i < uint32Array.length; i += 3) {
      decodedFragments.push([
        uint32Array[i],
        uint32Array[i + 1],
        uint32Array[i + 2],
      ])
    }

    // Перетворюємо байти в текст (очікується, що файл містить коми та цифри
у вигляді рядка)
    const text = decodedFragments.map(triplet =>
triplet.join(',')).join('\n')
    setFileText(text)
  }

  const handleSubmit = async () => {
    if (!fileText) return alert('Вміст файлу порожній')
    if (!width || !height) return alert('Вкажіть ширину та висоту')

    setIsLoading(true)
    setDecodingTime(null)

    localStorage.setItem('imageWidth', width.toString())
    localStorage.setItem('imageHeight', height.toString())

    const startTime = performance.now()

    try {
      // Перетворюємо текст у масив BigInt
      const lines = fileText.trim().split('\n')
      const flatValues = lines.flatMap(line =>
        line.split(',').map((val) => parseInt(val.trim()))
      )
      const uint32Array = new Uint32Array(flatValues)
      // Відправляємо байти на бекенд
      const blob = new Blob([uint32Array], { type: 'application/octet-stream'
})
      const responseBlob = await decodeImage(blob, width, height,
enableRestoration)

      setDecodedImageUrl(URL.createObjectURL(responseBlob))
      setDecodingTime(performance.now() - startTime)
    } catch (error) {
      alert(error.message)
    } finally {
      setIsLoading(false)
    }
  }


  return (
    <div className="max-w-xl mx-auto mt-10 p-6 bg-white rounded shadow">
      <h3 className="text-xl font-semibold mb-4">Декодування зображення</h3>

      <input
        type="file"
        accept="application/octet-stream"
```

```
        onChange={handleFileChange}
        className="mb-4 p-2 border-2 border-gray-300 rounded-md cursor-
pointer"
      />

      <div className="mb-4">
        <label className="block text-sm font-medium mb-1">Вміст файлу
(байти):</label>
        <textarea
          rows={8}
          value={fileText}
          onChange={(e) => setFileText(e.target.value)}
          className="w-full px-3 py-2 border-2 border-gray-300 rounded-md
font-mono"
        />
      </div>

      <div className="mb-4 flex gap-4">
        <div className="flex-1">
          <label className="block text-sm font-medium">Ширина:</label>
          <input
            type="number"
            value={width}
            onChange={(e) => setWidth(e.target.value)}
            className="mt-1 px-3 py-2 border-2 border-gray-300 rounded-md w-
full"
          />
        </div>
        <div className="flex-1">
          <label className="block text-sm font-medium">Висота:</label>
          <input
            type="number"
            value={height}
            onChange={(e) => setHeight(e.target.value)}
            className="mt-1 px-3 py-2 border-2 border-gray-300 rounded-md w-
full"
          />
        </div>
      </div>

      <div className="mb-4">
        <label className="flex items-center cursor-pointer">
          <div className="relative">
            <input
              type="checkbox"
              checked={enableRestoration}
              onChange={(e) => setEnableRestoration(e.target.checked)}
              className="sr-only"
            />
            <div
              className={`block w-14 h-8 rounded-full transition-colors
duration-300 ${
                enableRestoration ? 'bg-green-500' : 'bg-gray-300'
              }`}
            ></div>
            <div
              className={`dot absolute left-1 top-1 bg-white w-6 h-6 rounded-
full transition-transform duration-300 ${
                enableRestoration ? 'translate-x-6' : ''
              }`}
            ></div>
          </div>
          <span className="ml-3 text-gray-700">Увімкнути відновлення</span>
        </label>
```

```
      </div>

      <button
        onClick={handleSubmit}
        disabled={isLoading}
        className="bg-purple-600 text-white px-4 py-2 rounded hover:bg-
purple-700"
      >
        {isLoading ? 'Обробка...' : 'Декодувати'}
      </button>

      {decodedImageUrl && (
        <div className="mt-6">
          {restoringTime !== null && (
            <p className="mt-4">Час декодування: {(restoringTime /
1000).toFixed(2)} c</p>
          )}
          <h4 className="font-medium mb-2">Декодоване зображення:</h4>
          <img src={decodedImageUrl} alt="Decoded" className="w-[150%] max-w-
full rounded" />
          <br />
          <a href={decodedImageUrl}
download={`${selectedFile.name.split('.')[0]}_decoded.png`}>
            <button className="bg-green-600 text-white px-4 py-2 mt-2 rounded
hover:bg-green-700">
              Завантажити зображення
            </button>
          </a>
        </div>
      )}
    </div>
  )
}

export default DecodeImage
```

Файл ImageSimilarity.jsx

Реалізація функціональної задачі інтерфейсу для порівняння подібності зображення

```
import { useState } from 'react';
import { get_ssim } from '../api/apiClient';

function ImageSimilarity() {
  const [originalFile, setOriginalFile] = useState(null);
  const [decodedFile, setDecodedFile] = useState(null);
  const [ssim, setSsim] = useState(null);
  const [isLoading, setIsLoading] = useState(false);

  const handleOriginalChange = (e) => {
    const file = e.target.files[0];
    if (file && (file.type === 'image/png' || file.type === 'image/jpeg')) {
        setOriginalFile(file);
    } else {
        alert('Будь ласка, виберіть PNG або JPG файл')
    }
  };

  const handleDecodedChange = (e) => {
    const file = e.target.files[0];
    if (file && (file.type === 'image/png' || file.type === 'image/jpeg')) {
        setDecodedFile(file);
    } else {
```

```
        alert('Будь ласка, виберіть PNG або JPG файл')
    }  };

  const handleCompare = async () => {
    if (!originalFile || !decodedFile) {
      alert('Будь ласка, оберіть обидва зображення');
      return;
    }

    setIsLoading(true);
    setSsim(null);

    try {
      const result = await get_ssim(originalFile, decodedFile);
      if (result?.ssim !== undefined) {
        setSsim((result.ssim * 100).toFixed(2)); // у відсотках
      } else {
        alert('Некоректна відповідь від сервера');
      }
    } catch (error) {
      alert(error.message || 'Сталася помилка при обчисленні SSIM');
    } finally {
      setIsLoading(false);
    }
  };

  return (
    <div className="max-w-xl mx-auto mt-10 p-6 bg-white rounded shadow">
      <h3 className="text-xl font-semibold mb-4">Порівняння зображень
(SSIM)</h3>

      <div className="mb-4">
        <label className="block font-medium mb-1">Оригінальне
зображення</label>
        <input
          type="file"
          accept="image/png, image/jpeg"
          onChange={handleOriginalChange}
          className="w-full p-2 border-2 border-gray-300 rounded-md cursor-
pointer hover:border-blue-500"
        />
      </div>

      <div className="mb-4">
        <label className="block font-medium mb-1">Декодоване
зображення</label>
        <input
          type="file"
          accept="image/png, image/jpeg"
          onChange={handleDecodedChange}
          className="w-full p-2 border-2 border-gray-300 rounded-md cursor-
pointer hover:border-blue-500"
        />
      </div>

      <button
        onClick={handleCompare}
        disabled={isLoading}
        className="bg-pink-600 text-white px-6 py-3 rounded hover:bg-pink-700
disabled:opacity-50"
      >
        {isLoading ? 'Обробка...' : 'Порівняти'}
      </button>
```

```
      {ssim && (
        <div className="mt-6 text-center text-lg">
          <span className="font-semibold">SSIM:</span> {ssim}%
        </div>
      )}
    </div>
  );
}

export default ImageSimilarity;
```

Файл BaseManagement.jsx

Реалізація функціональної задачі управління базою фрагментів

```
import { useState, useEffect } from 'react';
import { changeSimilarityThreshold, fetchCurrentThreshold } from
'../api/apiClient';

function BaseManagement() {
  const [threshold, setThreshold] = useState(null);
  const [isLoading, setIsLoading] = useState(false);
  const [inputValue, setInputValue] = useState('');
  const [message, setMessage] = useState(null);
  const [error, setError] = useState(null);

  // Завантажуємо початковий поріг один раз при монтуванні
  useEffect(() => {
    fetchCurrentThreshold()
      .then(value => {
        setThreshold(value);
        setInputValue(value.toString());
      })
      .catch(err => setError(err.message));
  }, []);

  const handleInputChange = (e) => {
    setInputValue(e.target.value);
  };

  const handleSubmit = async () => {
    setIsLoading(true);
    setError(null);
    setMessage(null);


    try {
      const newThreshold = parseFloat(inputValue);
      const response = await changeSimilarityThreshold(newThreshold);
      setMessage(response.message || 'Поріг успішно змінено');
      setThreshold(response.similarity_threshold);
      setInputValue(response.similarity_threshold.toString());
    } catch (err) {
      setError(err.message || 'Сталася помилка');
    } finally {
      setIsLoading(false);
    }
  };

  return (
    <div className="max-w-xl mx-auto mt-10 p-6 bg-white rounded shadow">
      <h3 className="text-xl font-semibold mb-4">Управління базою
фрагментів</h3>
      <p className="text-lg font-semibold mb-2">Зміна порогу подібності
фрагментів</p>
```

```
      <p className="text-lg font-semibold mb-2">
        Поточний поріг подібності:{" "}
        <span className="font-bold text-blue-700 bg-blue-100 px-2 py-1
rounded-md shadow-sm">
          {threshold !== null ? threshold : 'Завантаження...'}
        </span>
      </p>


      <label className="block text-sm text-gray-700 mb-1"
htmlFor="thresholdInput">
        Введіть значення порогу (від 0 до 1):
      </label>
      <input
        id="thresholdInput"
        type="number"
        step="0.05"
        min="0"
        max="1"
        value={inputValue}
        onChange={handleInputChange}
        className="w-full border border-gray-300 rounded-md px-3 py-2 mb-4
focus:outline-none focus:ring-2 focus:ring-blue-500"
        placeholder="Наприклад: 0.75"
      />

      <button
        onClick={handleSubmit}
        disabled={isLoading}
        className="bg-blue-600 text-white px-6 py-2 rounded hover:bg-blue-700
disabled:opacity-50"
      >
        {isLoading ? 'Застосування...' : 'Змінити поріг'}
      </button>

      {message && (
        <div className="mt-4 w-full flex items-start gap-2 border border-
green-400 bg-green-50 rounded-lg p-3">
          <span>✅</span>
          <p className="text-green-700 font-medium">{message}</p>
        </div>
      )}

      {error && (
        <div className="mt-4 w-full flex items-start gap-2 border border-red-
400 bg-red-50 rounded-lg p-3">
          <span>❌</span>
          <p className="text-red-700 font-medium">{error}</p>
        </div>
      )}
    </div>
  );
}

export default BaseManagement;
```

Файл apiClient.js

Реалізація функціональної задачі для звернннення до API кодування

```
const API_BASE_URL = import.meta.env.VITE_API_BASE_URL;

/**
 * Обробляє помилки відповіді від сервера
```

```
 */
async function handleApiError(response, defaultMessage) {
  let errorMessage = defaultMessage;

  try {
    const errorData = await response.json();
    errorMessage = errorData.detail || errorData.message || errorMessage;
  } catch {
    try {
      const text = await response.text();
      if (text) errorMessage = text;
    } catch (_) {
      // лишаємо дефолтне повідомлення
    }
  }

  throw new Error(errorMessage);
}

export async function addFragment(imageFile) {
  const formData = new FormData();
  formData.append("file", imageFile);

  const response = await fetch(`${API_BASE_URL}/add-fragments/`, {
    method: "POST",
    body: formData,
  });

  if (!response.ok) {
    await handleApiError(response, 'Помилка при додаванні нового файлу до
бази фрагментів');
  }

  return await response.json();
}

export async function get_ssim(originalImage, decodedImage) {
  const formData = new FormData();
  formData.append('original_img', originalImage);
  formData.append('decoded_img', decodedImage);

  const response = await fetch(`${API_BASE_URL}/get-ssim/`, {
    method: 'POST',
    body: formData,
  });

  if (!response.ok) {
    await handleApiError(response, 'Помилка при визначенні SSIM');
  }

  return await response.json();
}

export async function encodeImage(imageFile) {
  const formData = new FormData();
  formData.append('file', imageFile);

  const response = await fetch(`${API_BASE_URL}/encode-image/`, {
    method: 'POST',
    body: formData,
  });

  if (!response.ok) {
    await handleApiError(response, 'Помилка при кодуванні зображення');
```

```
  }

  return await response.blob();
}


export async function decodeImage(imageFile, width, height, enableRestoration
= false) {
  const formData = new FormData();
  formData.append('compressed_img', imageFile);
  formData.append('width', width);
  formData.append('height', height);
  formData.append('restore_image', enableRestoration ? 'true' : 'false')

  const response = await fetch(`${API_BASE_URL}/decode-image/`, {
    method: 'POST',
    body: formData,
  });

  if (!response.ok) {
    await handleApiError(response, 'Помилка при декодуванні зображення');
  }

  return await response.blob();
}

export async function changeSimilarityThreshold(threshold) {
  if (isNaN(threshold) || threshold < 0 || threshold > 1) {
    throw new Error("Поріг подібності має бути в межах від 0 до 1.");
  }

  const response = await fetch(`${API_BASE_URL}/change-similarity-
threshold/${threshold}`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
  });

  if (!response.ok) {
    await handleApiError(response, "Помилка при зміні порогу подібності");
  }

  return await response.json();
}

export async function fetchCurrentThreshold() {
  const response = await fetch(`${API_BASE_URL}/get-similarity-threshold/`);
  if (!response.ok) {
    throw new Error('Не вдалося завантажити поточний поріг');
  }
  const data = await response.json();
  return data.similarity_threshold;
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ КОДУВАННЯ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ГЛИБИННИХ НЕЙРОННИХ МЕРЕЖ

Програма та методика тестування

КПІ.ІП-1322.045490.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юрій ОЛІЙНИК

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Віталій МУЗИЧУК

Київ – 2025

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є розроблене програмне забезпечення кодування зображень з використанням глибинних нейронних мереж.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка роботи нейронної мережі;
- перевірка збереження фрагментів у базі;
- перевірка правильності визначення подібності зображень;
- перевірка сумісності вебзастосунка з останніми версіями сучасних браузерів (Chrome, Opera, Firefox тощо);
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;
- тестування «чорної скриньки» – об’єктом тестування тут є функції присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Перевірка програмного забезпечення здійснюється вручну шляхом наскрізного тестування (End-to-End, E2E, або ланцюгового тестування) з метою виявлення помилок і недоліків як у функціональній складовій, так і в аспектах зручності використання. Для оцінки стабільності роботи та працездатності застосунку необхідно виконати такі види тестування:

- динамічне тестування на відповідність функціональним вимогам;
- мануальне тестування для перевірки коректності обробки неправильних даних;
- тестування «чорної скриньки» для перевірки якості кодування й декодування, відновлення зображення й визначення рівня подібності між зображеннями;
- мануальне тестування на визначення часу обробки фотографій;
- мануальне тестування на визначення відсотку стиснення зображення;
- тестування програмного застосунку у різних браузерах та їх версіях.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ КОДУВАННЯ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ГЛИБИННИХ НЕЙРОННИХ МЕРЕЖ

Керівництво користувача

КПІ.ІП-1322.045490.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юрій ОЛІЙНИК

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Віталій МУЗИЧУК

Київ – 2025

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Програмне забезпечення кодування зображень з використанням глибинних нейронних мереж призначене для ефективного стиснення й відновлення зображень з використанням методу подібності фрагментів.

Основним призначенням програми є зменшення обсягу пам'яті, необхідної для зберігання зображень, а також підвищення ефективності їх передачі через мережі зв'язку.

Кінцева збірка програмного забезпечення містить веб інтерфейс для демонстрування роботи розробленого алгоритму й дозволяє користувачеві – кодувати растрові зображення й отримувати їх у вигляді бінарних файлів, декодувати стисненні зображення, відновлювати втрачені фрагменти зображень, визначати подібність між стиснутим й оригінальним зображенням.

Репозиторії для встановлення містить:

– вихідний код для фронтенд й бекенд частини застосунку;

– сценарії для налаштування середовища та запуску програми, включаючи Dockerfile та docker-compose.yml;

– файл для налаштування CI/CD процесу деплойменту до Google Cloud Platform.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесора: архітектура x86-64 або ARM64, щонайменше 32 потоки обробки;

– об’єм ОЗП: 8 ГБ;

– графічний процесор (GPU): сумісний з CUDA, рекомендовано NVIDIA Tesla T4;

– швидкість підключення до мережі Інтернет: від 20 Мбіт/с;

– програмне забезпечення: встановлений CUDA Toolkit, сумісний із відповідною версією TensorFlow.

Рекомендована конфігурація технічних:

– тип процесора: архітектура x86-64 або ARM64, щонайменше 64 потоки обробки;

– об’єм ОЗП: 32 ГБ;

– графічний процесор (GPU): сумісний з CUDA, рекомендовано NVIDIA L4 або вище;

– швидкість підключення до мережі Інтернет: від 100 Мбіт/с;

– програмне забезпечення: встановлений CUDA Toolkit, сумісний із відповідною версією TensorFlow.

2.2 Завантаження застосунку

На даний момент застосунок можна встановити власноруч, використовуючи вихідний код розміщений у репозиторіях GitHub. Для цього спершу необхідно склонувати його до свого профілю, виконати необхідну підготовку Google Cloud та запустити процес CI/CD через GitHub Actions. Програма буде доступна через браузер.

2.3 Перевірка коректної роботи

Після завершення процесу розгортання програмного забезпечення необхідно відкрити веб-браузер і перейти за адресою, яка вказана у конфігурації. У вікні інтерфейсу треба переконатися, що всі елементи користувацького інтерфейсу завантажилися коректно і відображаються правильно.

3 ВИКОНАННЯ ПРОГРАМИ

Після коректного встановлення застосунку для користувача буде доступно головний екран (рисунок 3.1), де користувач може обрати одну з сторінок для початку роботи з програмою.

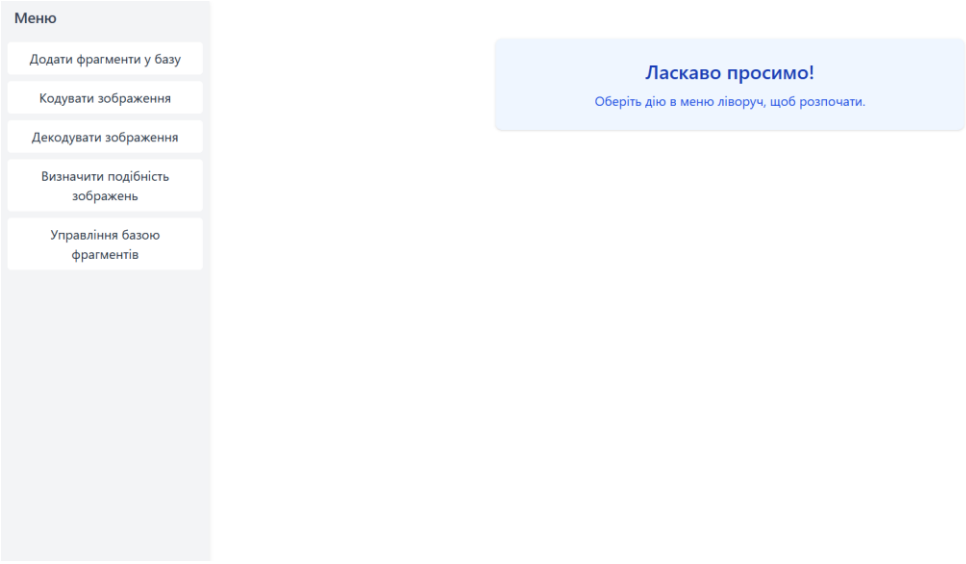


Рисунок 3.1 – Головна сторінка

Надалі користувачу необхідно заповнити базу фрагментами із зображень, якщо він попередньо цього ще не зробив. Для цього треба натиснути на кнопку «Додати фрагменти у базу» в меню й відкриється сторінка для завантаження зображень (рисунок 3.2).

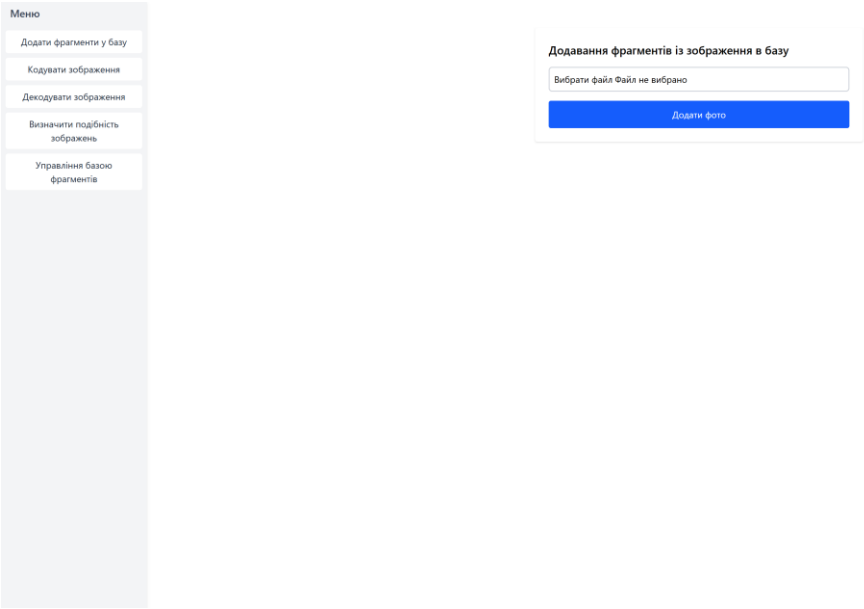


Рисунок 3.2 – Сторінка для завантаження фрагментів у базу

Наступним кроком буде обрання необхідного фото для завантаження до бази фрагментів. Для цього необхідно натиснути на поле «Вибрати файл» й додати бажане зображення (рисунок 3.3).

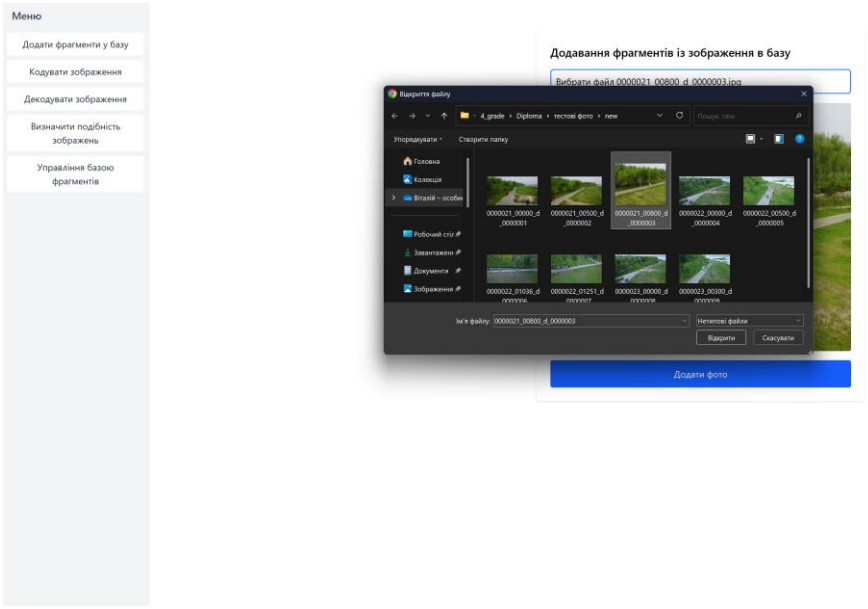


Рисунок 3.3 – Додання зображення в програму

Якщо користувач спробує обрати невідповідний тип файлу він отримає помилку й не зможе здійснити запит на сервер (рисунок 3.4).

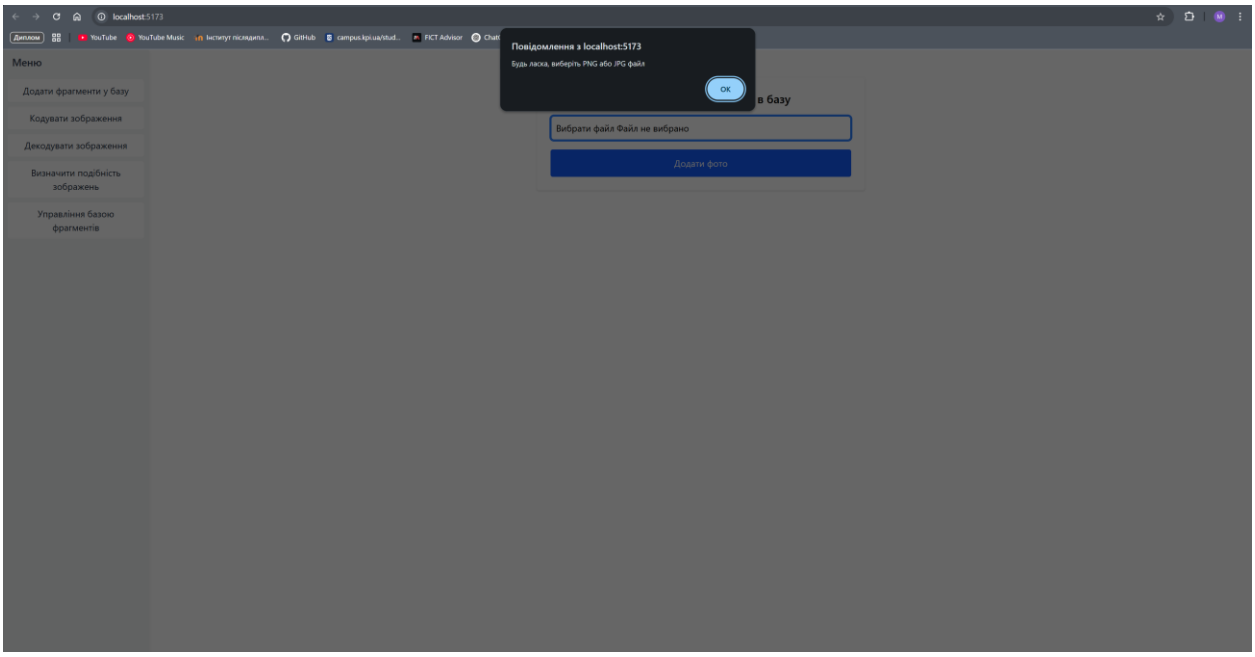


Рисунок 3.4 – Повідомлення про невідповідність типу файлу

Також якщо користувач нічого не завантажить й вирішить натиснути на кнопку «Додати фото» система сповістить його про те, що необхідно прикріпити зображення. Такі попередження про некоректні дії користувача

присутні у програмному забезпеченні для всіх полів й сторінок де необхідно завантажувати фото (рисунок 3.5).

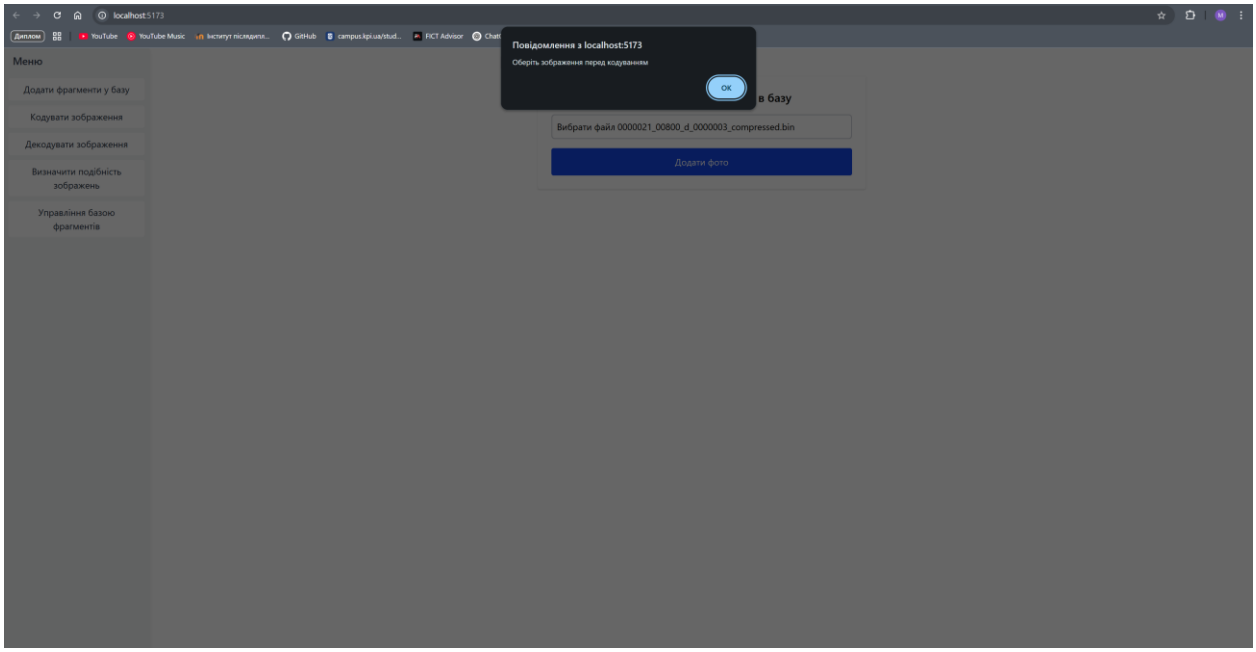


Рисунок 3.5 – Повідомлення про необхідність вибору зображення перед надсиланням

Для коректного завантаження користувач повинен обрати фото правильного формату й натиснути кнопку. Після цього відбудеться запит на сервер, який потім поверне у відповідь статистику щодо додавання нових фрагментів у базу (рисунок 3.6).

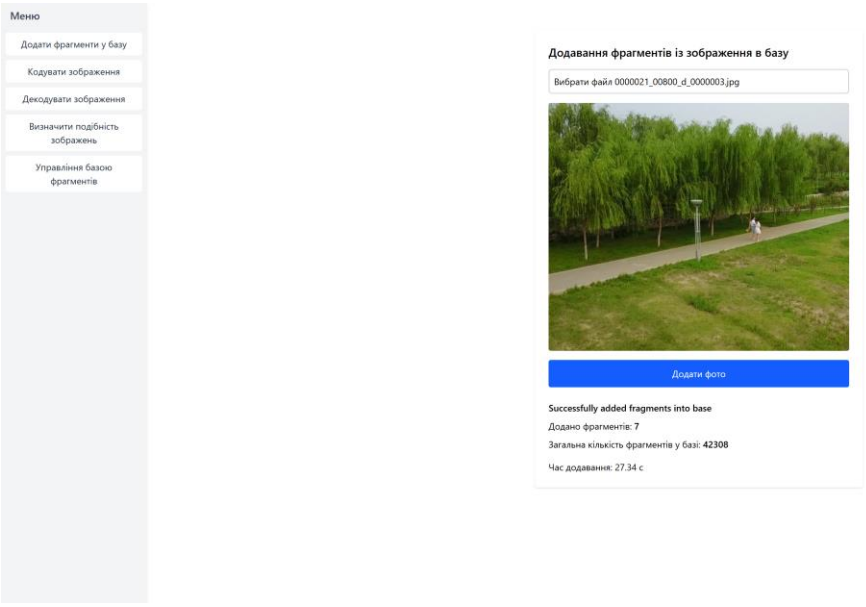


Рисунок 3.6 – Успішне завантаження зображення до бази фрагментів

Після того як користувач заповнив базу зображеннями з подібним змістом він може здійснити кодування нових зображень на основі фрагментів з бази. Для цього потрібно перейти на сторінку «Кодувати зображення», вибрати зображення для кодування й натиснути кнопку «Стиснути зображення». У результаті з’явиться інформація щодо статистики кодування й додаткова кнопка для завантаження стиснутого файлу (рисунок 3.7).

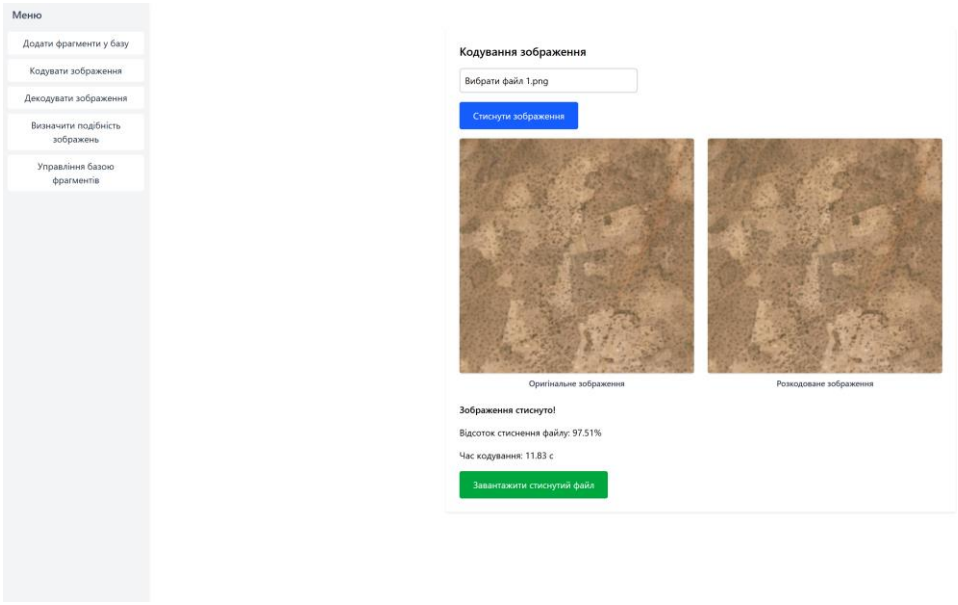


Рисунок 3.7 – Вигляд сторінка після кодування зображення

Після цього користувач може перейти на сторінку «Декодувати зображення» та завантажити попередньо стиснутий файл із зображенням (рисунок 3.8).

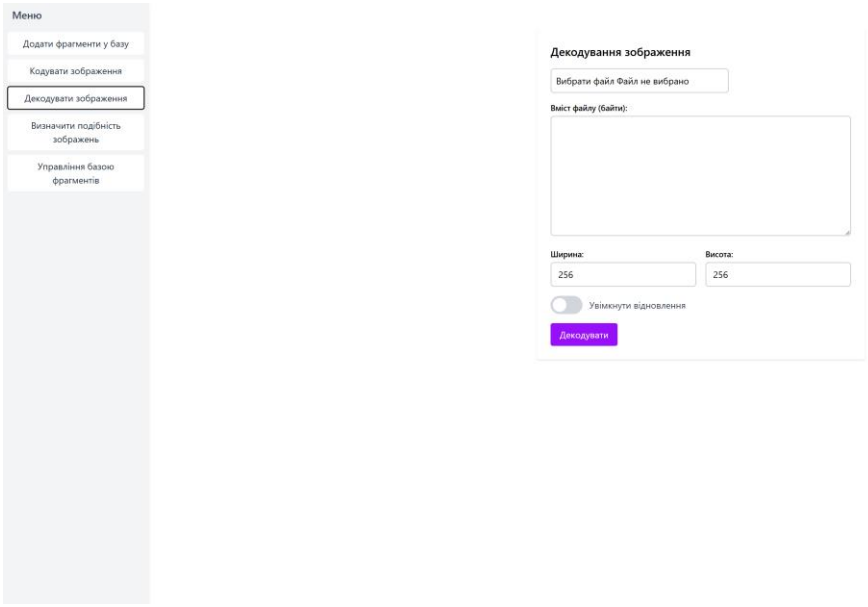


Рисунок 3.8 – Сторінка декодування зображення

Після того як файл буде завантажено, у полі «Вміст файлу» з’явиться закодована інформація, що автоматично передається на сервер для подальшої обробки. Далі користувач має натиснути кнопку «Декодувати», після чого отримає результат — декодоване зображення, яке можна зберегти на свій пристрій (рисунок 3.9).

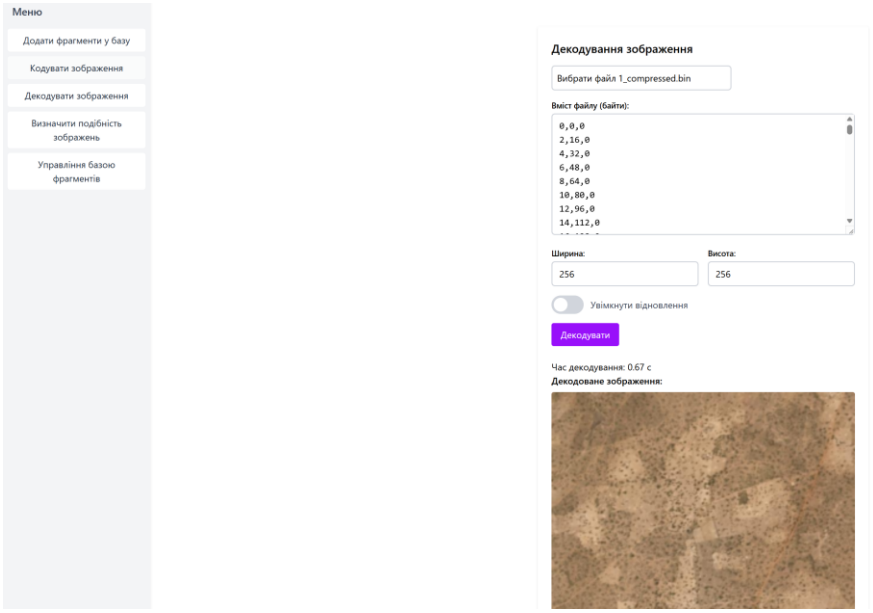


Рисунок 3.9 – Успішно декодоване зображення

Якщо користувач введе неправильні розміри вихідного зображення система сповістить його про це (рисунок 3.10).

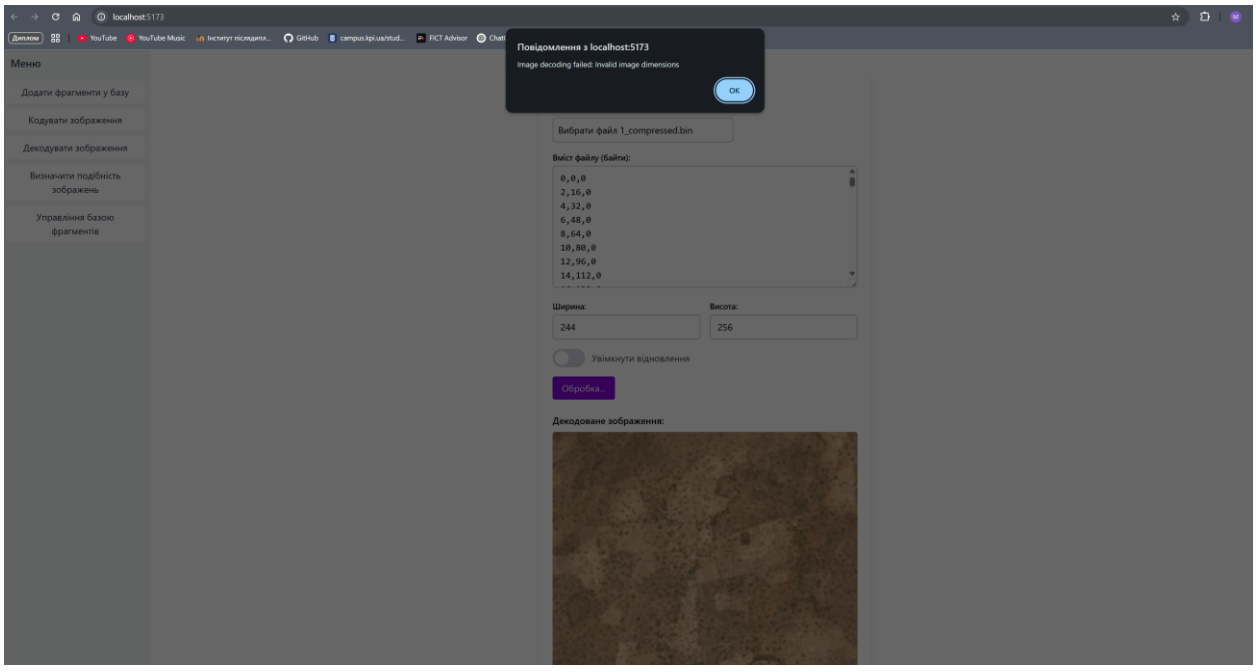


Рисунок 3.10 – Повідомлення про неправильну розмірність вихідного файлу

Для імітації втрати фрагментів користувач може вручну видалити кілька записів із поля «Вміст файлу» та повторно натиснути «Декодувати». У результаті система поверне зображення з відсутніми фрагментами (рисунок 3.11).

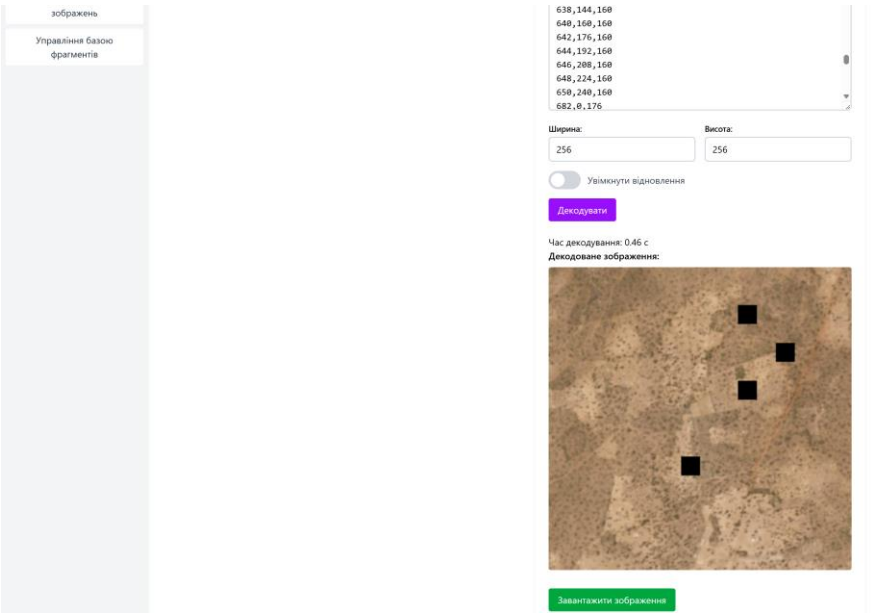


Рисунок 3.11 – Вигляд декодоване зображення з втраченими фрагментами

Для того щоб скористатися функціоналом з відновлення зображення необхідно активувати перемикач «Увімкнути відновлення» й відправити інформацію на декодування ще раз (рисунок 3.12).

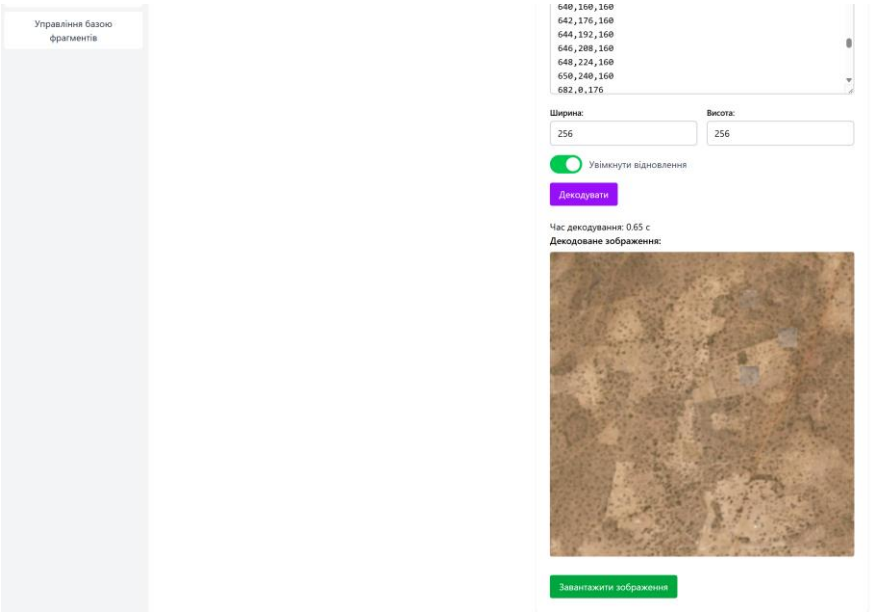


Рисунок 3.12 – Вигляд зображення з відновленими фрагментами

Щоб оцінити ефективність стиснення, слід перейти на сторінку «Визначення подібності зображень» у головному меню та завантажити в

відповідні поля оригінальне та декодоване зображення. Після натискання кнопки «Порівняти» сервер поверне результат порівняння, який відобразиться у відповідному полі (рисунок 3.13).

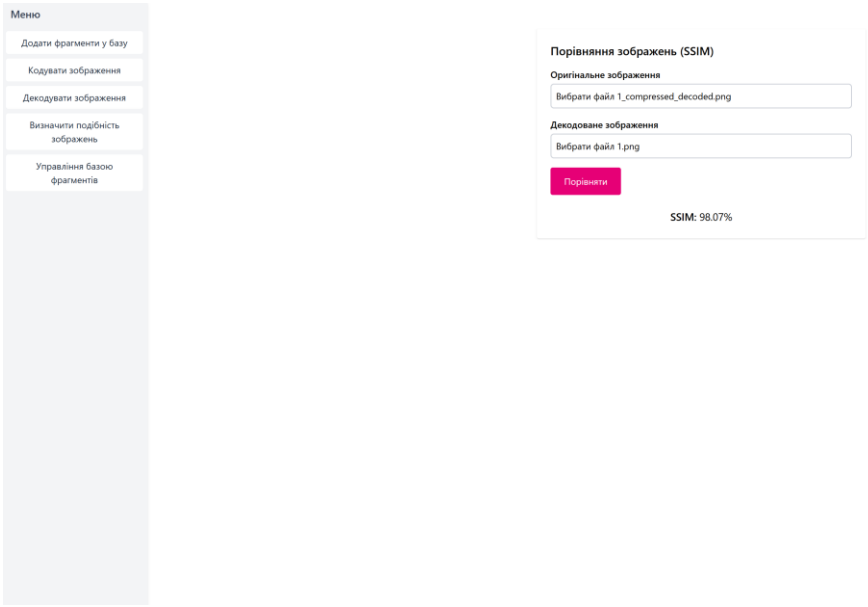


Рисунок 3.13 – Порівняння подібності двох зображень

Користувачеві доступна також функція керування базою фрагментів, де можна налаштувати поріг подібності, що використовується під час кодування зображень. Цей показник безпосередньо впливає як на якість отриманого зображення, так і на швидкість збільшення обсягу бази фрагментів. На відповідній сторінці відображається поточне значення порогу, яке користувач може змінити, ввівши значення в діапазоні від 0 до 1 (рисунок 3.14).

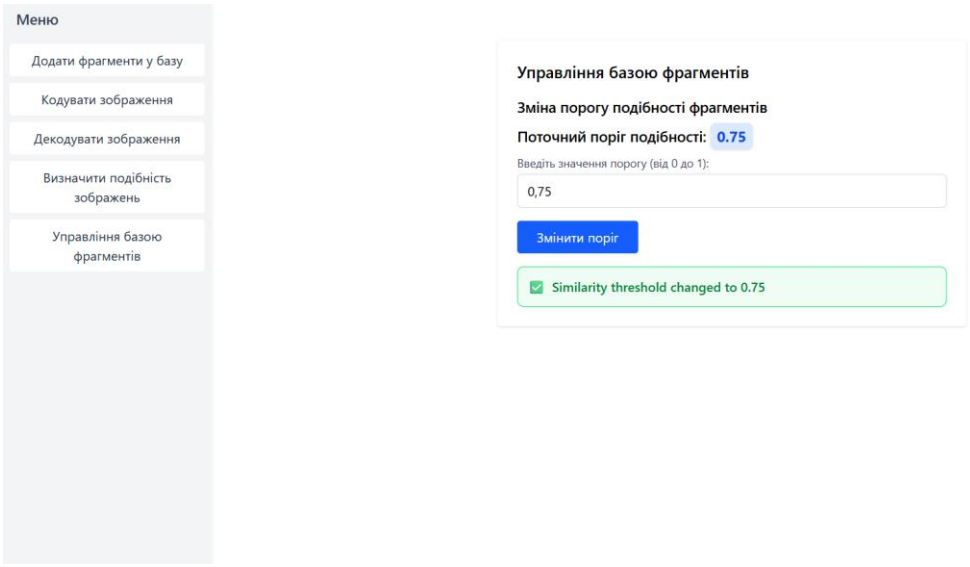


Рисунок 3.14 – Зміна порогу подібності

У разі введення некоректного значення система повідомить користувача про помилку (рисунок 3.15).

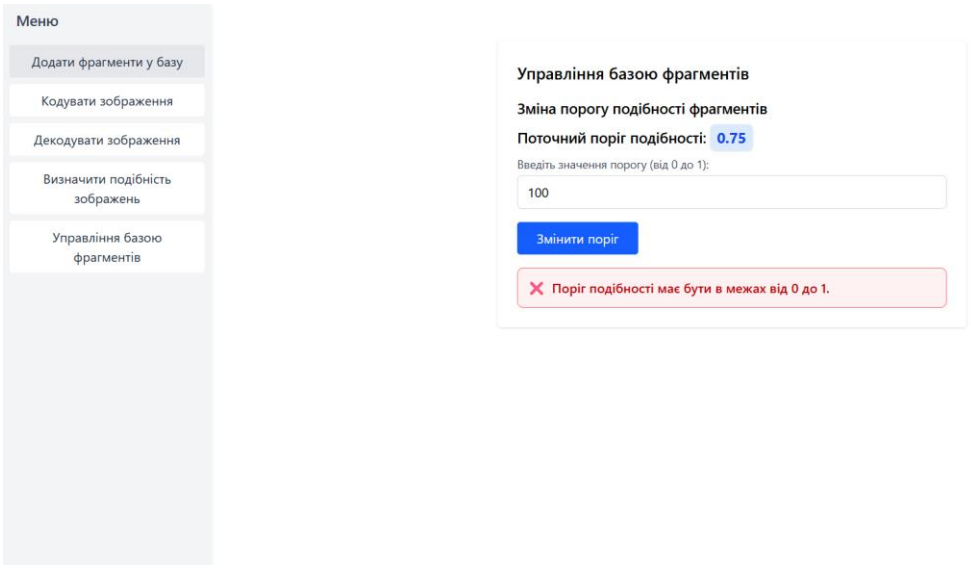


Рисунок 3.15 – Помилка при неправильному порогу подібності

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ КОДУВАННЯ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ГЛИБИННИХ НЕЙРОННИХ МЕРЕЖ

Графічний матеріал

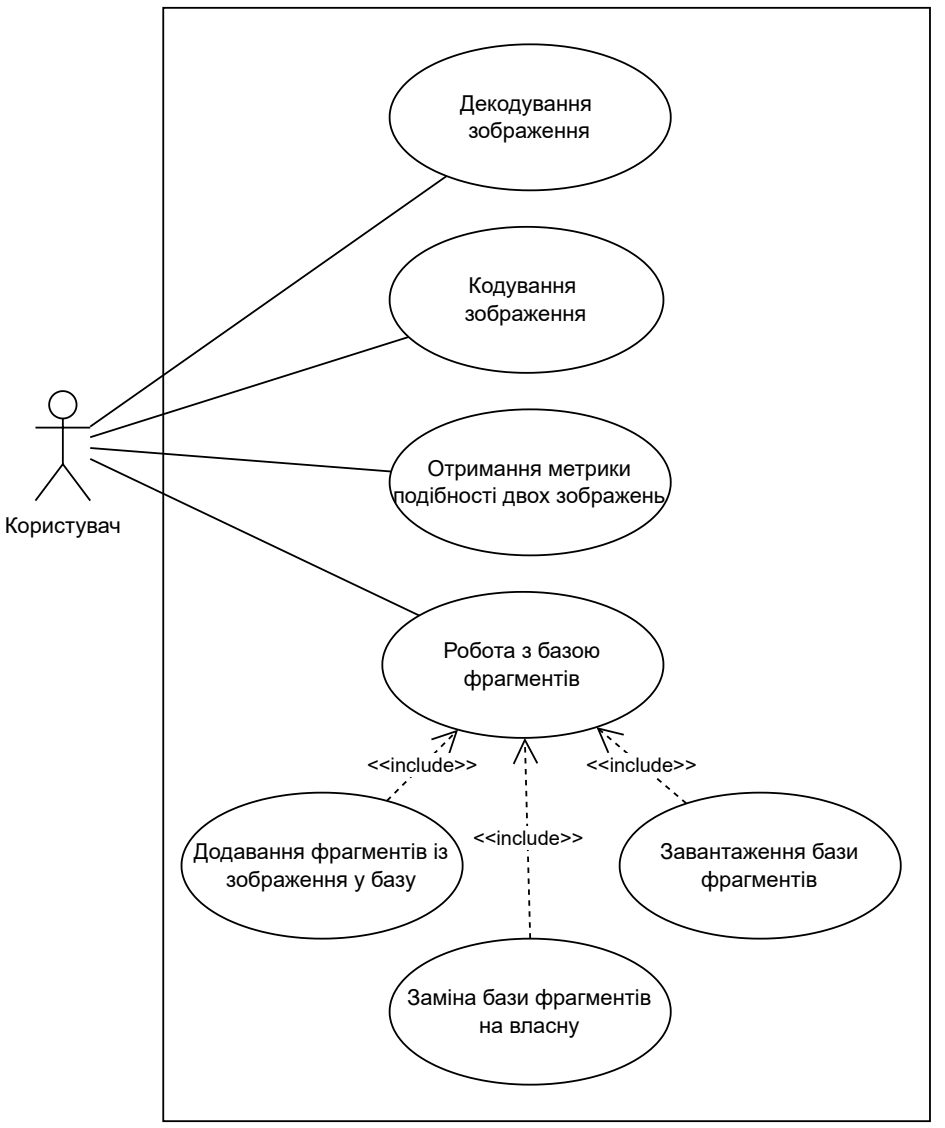
КПІ.ІП-1322.045490.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юрій ОЛІЙНИК

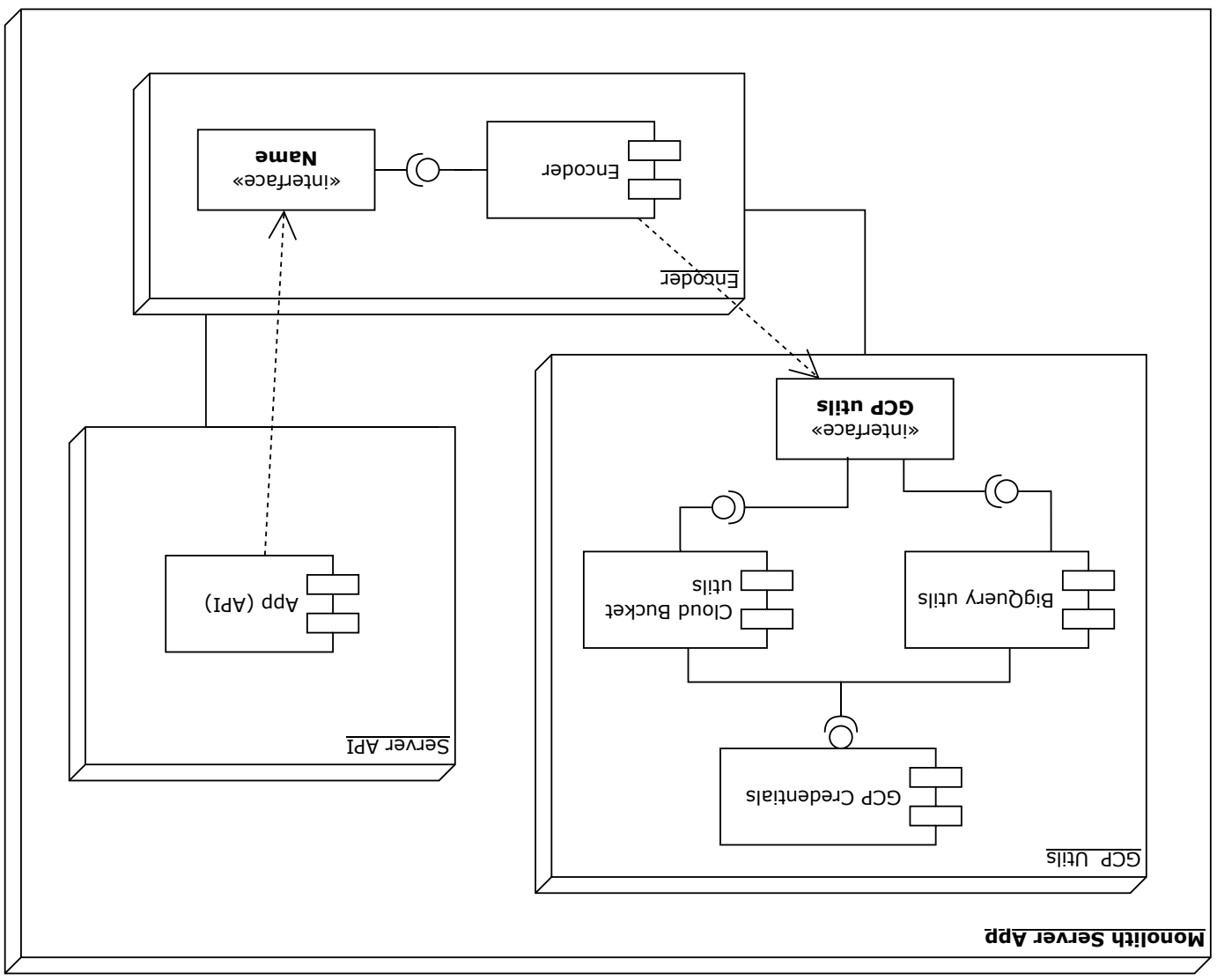
Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Віталій МУЗИЧУК

Київ – 2025



					КПІ.ІП-1322.045490.06.99.ССВ	
					Схема варіантів використань	Лит. / Маса / Масштаб
Зм.	Арк.	№ докум.	Підп.	Дата		Аркуш 1 / Аркушів 3
Розробив		Музичук В. А.			Програмне забезпечення для кодування зображень з використанням глибинних нейронних мереж	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-13
Перевірив		Олійник Ю. О.				
Т. контр.						
Н. контр.		Вітковська І. І.				
Затвердив		Жаріков Е.В.				



Змн.	Арк.	№ докум.	Підп.	Дата
Розробив		Музичук В. А.		
Перевірив		Олійник Ю. О.		
Т. контр.				
Н. контр.		Вітковська І. І.		
Затвердив		Жаріков Е.В.		

КПІ.ІП-1322.045490.06.99.ССМ

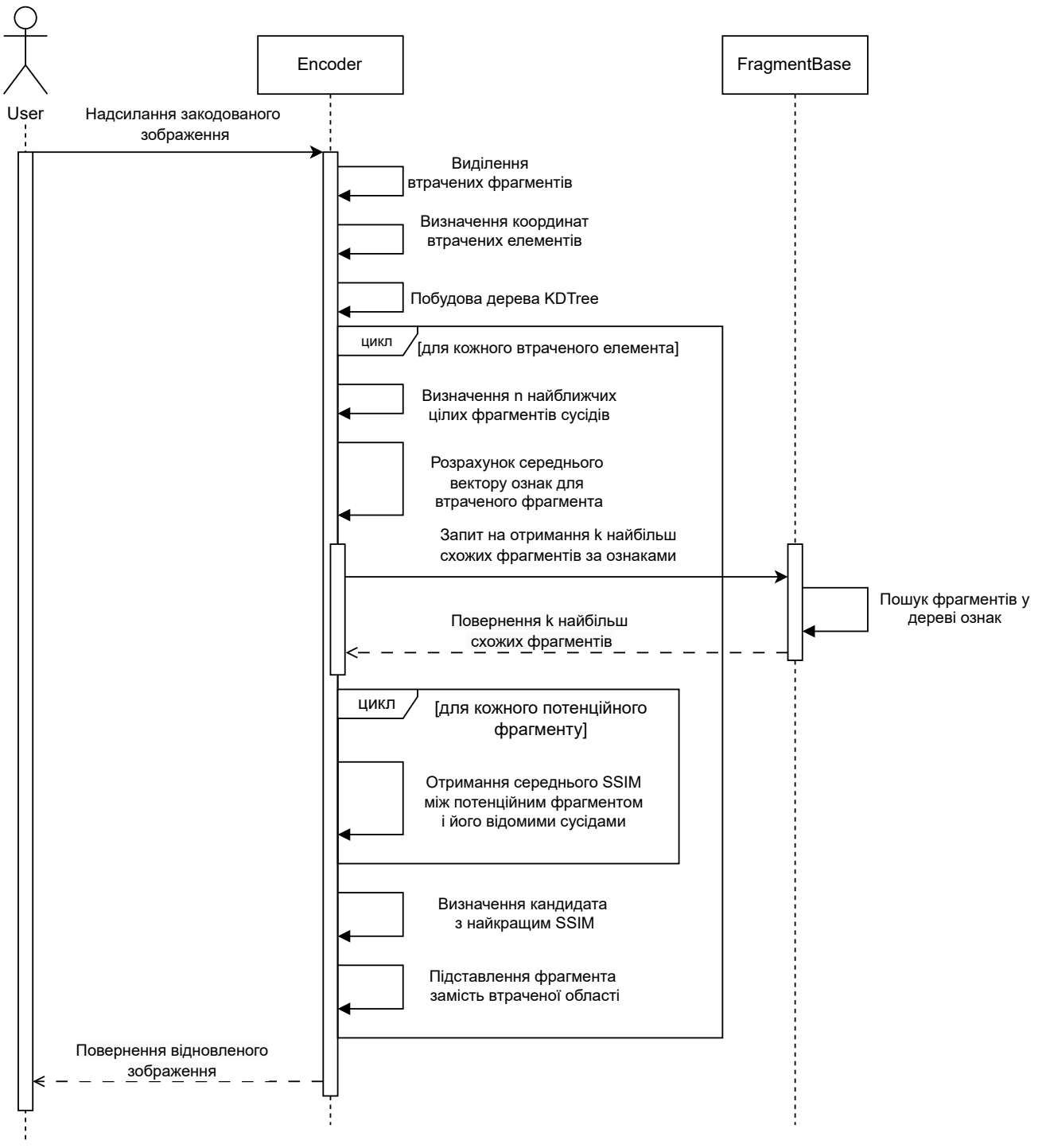
Літ.	Маса	Масштаб
Аркуш 2		Аркушів 3

Схема структура компонентів програмного забезпечення

Програмне забезпечення для кодування зображень з використанням глибинних нейронних мереж

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-13

КПІ.ІП-1322.045490.06.99.ССМ



					КПІ.ІП-1322.045490.06.99.ССП
Зм.	Арк.	№ докум.	Підп.	Дата	
Розробив		Музичук В. А.			Схема структурна послідовності
Перевірив		Олійник Ю. О.			
Т. контр.					
Н. контр.		Вітковська І. І.			Програмне забезпечення для кодування зображень з використанням глибинних нейронних мереж
Затвердив		Жаріков Е.В.			

Лит.	Маса	Масштаб
Аркуш 3	Аркушів 3	

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-13