МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет

імені Петра Могили

Факультет комп’ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри,

д-р техн. наук, проф.

_____Ірина ЖУРАВСЬКА

« __ » _____ 2025 р.

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Апаратно-програмний комплекс стиснення проміжних кадрів відеопотоку

Спеціальність 123 Комп’ютерна інженерія

123 – КБР.01 – 605.22450505

Студент _____ Дмитро. ДОЦЕНКО
підпис
«__» _____ 202__ р.

Керівник канд. техн. наук, доцент _____ Ярослав КРАЙНИК
підпис
«__» _____ 202__ р.

Миколаїв – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Зав. кафедри _____Ірина ЖУРАВСЬКА

«_____» _____ 2025 р.

ЗАВДАННЯ
на виконання кваліфікаційної магістерської роботи

Видано студенту групи 605 факультету комп'ютерних наук

Доценко Дмитро Володимирович
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи

Апаратно-програмний комплекс стиснення проміжних кадрів відеопотоку

Затверджена наказом по ЧНУ ім. Петра Могили від 23.06.2025 № 165/1.

2. Строк представлення кваліфікаційної роботи «_10_» ___грудня____ 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
Розроблений комбінований метод стиснення проміжних кадрів з використанням попередньої обробки, виділення вузлових точок, стиснення основної області методом QOI та ентропійного кодування вузлових точок алгоритмом Хаффмана. Розгорнуто апаратно-програмну реалізацію методу у двох варіантах: – вбудована система на STM32F746G; – клієнт-серверний вебзастосунок для обробки кадрів. Виконано тестування та оцінку ефективності розробленого комплексу.

4. Перелік питань, що підлягають розробці
Аналіз методів стиснення відео та зображень, визначення вимог до проміжних кадрів.Розробка комбінованого методу стиснення з декореляцією яскравісної та кольорових компонент. Реалізація попередньої обробки та виділення вузлових точок. Стиснення основної частини кадру методом QOI. Стиснення вузлових точок методом Хаффмана. Проєктування апаратно-програмного комплексу, реалізація на STM32 та у вебінтерфейсі. Проведення порівняльного тестування ефективності та якості.

5. Перелік графічних матеріалів

Структурна схема комплексу; Блок-схеми алгоритмів; Схеми інтерфейсу вебзастосунку; Результати тестування у вигляді графіків/таблиць._____

Керівник роботи _____ ___Ярослав КРАЙНИК

Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ ___Дмитро ДОЦЕНКО

Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Дата видачі завдання «____» _____ 20_____ р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: Апаратно-програмний комплекс стиснення проміжних кадрів відеопотоку

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КР	01.02.25	05.02.25	Виконав
2	Огляд літератури та аналіз існуючих методів	06.02.25	10.03.25	Виконав
3	Складання календарного плану БКР	11.03.25	05.04.25	Виконав
4	Аналіз предметної області та постановка задачі	06.04.25	18.04.25	Виконав
5	Розробка комбінованого методу стиснення	19.04.25	10.05.25	Виконав
6	Розробка математичної моделі та блок-схем	11.05.25	05.06.25	Виконав
7	Реалізація методу у вебзастосунку	06.06.25	21.08.25	Виконав
8	Реалізація методу на STM32F746G	22.08.25	15.10.25	Виконав
9	Тестування та аналіз результатів	16.10.25	01.11.25	Виконав
10	Оформлення роботи та презентації	02.11.25	10.11.25	Виконав
11	Попередній захист (I етап)	11.11.25	11.11.25	Виконав
12	Рецензування	20.11.25	24.11.25	Виконав
13	Попередній захист (II етап)	25.11.25	25.11.25	Виконав
14	Завершення оформлення	26.11.25	30.11.25	Виконав
15	Захист магістерської роботи	16.12.25	16.12.25	Виконав

Керівник роботи _____ Ярослав КРАЙНИК

Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ Дмитро ДОЦЕНКО

Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Дата видачі завдання «____» _____ 20_____ р.

АНОТАЦІЯ

до кваліфікаційної магістерської роботи
«Апаратно-програмний комплекс стиснення проміжних кадрів відеопотоку для вбудованих систем і вебзастосунку»
Студент 605 гр.: Доценко Дмитро Володимирович
Керівник: канд. техн. наук, доцент Крайник Ярослав Михайлович

Кваліфікаційна магістерська робота присвячена розробці апаратно-програмного комплексу для стиснення проміжних кадрів відеопотоку з використанням комбінованого методу, що поєднує попереднє перетворення кадру у колірний простір YCrCb, виділення вузлових точок, стиснення основної області алгоритмом QOI, а також ентропійне стиснення вузлових точок алгоритмом Хаффмана.

Актуальність роботи зумовлена потребою у методах зменшення обсягів відеоданих в умовах обмежених обчислювальних ресурсів вбудованих систем, а також необхідністю забезпечення швидкої обробки кадрів у мережевих та веб-середовищах.

Об’єкт дослідження – процес стиснення проміжних кадрів відеопотоку. Предмет дослідження – комбінований метод стиснення з розділенням обробки вузлових точок та основної області кадру.

У роботі розроблено архітектуру та виконано реалізацію комплексу у трьох варіантах: – на апаратній платформі STM32F746G для випадків автономної роботи вбудованих систем; – на одноплатному комп'ютері Raspberry Pi 4 для обробки потокового відео в реальному часі; – у вигляді вебзастосунку з клієнт-серверною взаємодією для обробки та аналізу кадрів у браузері.

Проведено порівняльне тестування, яке підтвердило ефективність комбінованого методу: досягнуто коефіцієнт стиснення до 48 разів та забезпечено роботу в режимі реального часу.

Робота пройшла апробацію на низці наукових конференцій, а саме: XXVI Всеукраїнській науково-практичній конференції «Могилянські читання – 2023» (Миколаїв, 2023 р.), XXVII Всеукраїнській науково-практичній конференції «Могилянські читання – 2024» (Миколаїв, 2024 р.), XXVIII Всеукраїнській науково-практичній конференції «Могилянські читання – 2025» (Миколаїв, 2025 р.), XXI Міжнародній науковій конференції «Ольвійський форум – 2023» (Миколаїв, 2023 р.), XXIII Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Стан, досягнення та перспективи інформаційних систем і технологій» (Одеса, 2023 р.).

Публікації. За результатами досліджень опубліковано статтю «Метод стиснення зображень з використанням попередньої обробки та алгоритмів Quite OK Image і Хаффмана» у науковому фаховому виданні «Електронне моделювання», випуск №2 (2024 р.), а також статтю «Побудова теплової карти Wi-Fi за допомогою IoT-модуля ESP8266» у науковому фаховому виданні «Комп'ютерно-інтегровані технології: освіта, наука, виробництво», випуск №60 (2025 р.).

Структура роботи. Пояснювальна записка магістерської роботи складається зі вступу, чотирьох розділів, висновків, переліку джерел посилання та додатків. У вступі визначається актуальність теми, сформульовано мету та завдання. У першому розділі проводиться аналіз існуючих технологій. У другому розділі розробляється математична модель та алгоритм комбінованого методу. Третій розділ присвячено апаратно-програмній реалізації комплексу. У четвертому розділі наведено результати експериментальних досліджень ефективності та швидкодії розробленого рішення.

Ключові слова: *зображення, стиснення, алгоритм Хаффмана, вузлові точки, ефективність, якість, порівняння, алгоритм QOI*

ABSTRACT

of the Master's Thesis

"Hardware-Software Complex for Compression of Intermediate Video Frames for Embedded Systems and Web Applications"

Student: Dotsenko Dmytro Volodymyrovych

Supervisor: Ph.D., Associate Professor Krainyk Yaroslav Mykhailovych

The master's qualification thesis is devoted to the development of a hardware-software complex for compressing intermediate video frames using a combined method that integrates preliminary conversion of frames into the YCrCb color space, extraction of nodal points, compression of the main region using the QOI algorithm, and entropy compression of nodal points using the Huffman algorithm.

The relevance of the work is determined by the need for methods that provide efficient video data reduction under limited computational resources of embedded systems, as well as the necessity to ensure rapid frame processing in network and web environments.

The object of the study is the process of compressing intermediate frames of a video stream. The subject of the study is a combined compression method with separate processing of nodal points and the main frame region.

The architecture was developed and the complex was implemented in three variants: – on the STM32F746G hardware platform for standalone embedded system operation; – on the Raspberry Pi 4 single-board computer for real-time video stream processing; – as a web application with client-server interaction for frame processing and analysis in a browser.

Comparative testing confirmed the efficiency of the combined method, achieving a compression ratio of up to 48x and ensuring real-time operation.

The work has been presented at the following conferences: XXVI All-Ukrainian Scientific and Practical Conference "Mohyla Readings – 2023" (Mykolaiv, 2023), XXVII All-Ukrainian Scientific and Practical Conference "Mohyla Readings – 2024" (Mykolaiv, 2024), XXVIII All-Ukrainian Scientific and Practical Conference "Mohyla Readings – 2025" (Mykolaiv, 2025), XXI International Scientific Conference "Olvian Forum – 2023" (Mykolaiv, 2023), XXIII All-Ukrainian Scientific and Technical Conference "State, Achievements and Prospects of Information Systems and Technologies" (Odesa, 2023).

Publications. Based on the research results, the article "Image Compression Method Using Pre-Processing and Quite OK Image and Huffman Algorithms" was published in the scientific journal Electronic Modeling, Issue No. 2 (2024), and the article "Building a Wi-Fi Heatmap Using the ESP8266 IoT Module" was published in the scientific journal Computer-Integrated Technologies: Education, Science, Production, Issue No. 60 (2025).

Structure of the thesis. The explanatory note consists of an introduction, four chapters, conclusions, a list of references, and appendices. The introduction defines the relevance, aim, and tasks. The first chapter analyzes existing technologies. The second chapter describes the development of the mathematical model and the combined compression algorithm. The third chapter is devoted to the hardware-software implementation of the complex. The fourth chapter presents the results of experimental research on the efficiency and performance of the developed solution.

Keywords: *image, compression, Huffman algorithm, node values, efficiency, quality, comparison, QOI algorithm.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

CABAC – Context-Adaptive Binary Arithmetic Coding

DCT – Discrete Cosine Transform

FLIF – Free Lossless Image Format

GIF – Graphics Interchange Format

JPEG – Joint Photographic Experts Group

LZW – Lempel-Ziv-Welch

MANIAC – Meta-Adaptive Near-zero Integer Arithmetic Coding

MOS – Mean Opinion Score

PNG – Portable Network Graphics

PSNR – Peak Signal-to-Noise Ratio

QOI – Quite OK Image format

RGB – Red Green Blue

SSI – Structural Similarity Index

WebP – Web Picture format

YCrCb – Luminance (Y) and Chrominance (Cr and Cb)

ВСТУП

За останні роки обсяг відеоданих зріс настільки, що вони стали звичайною частиною майже будь-якої цифрової системи. Відео використовують і у великих вебплатформах, і у відносно простих вбудованих рішеннях: системах відеоспостереження, IoT-пристроях, різних спеціалізованих комплексах. При цьому «сире» (raw) відео займає дуже багато місця. Великі файли перевантажують канали зв’язку, вимагають значного дискового простору і помітно впливають на швидкість роботи вебінтерфейсів, затримки під час перегляду, загальне враження користувача. На практиці це добре видно: без стиснення нормальна передача та зберігання відео майже нереальні. Стиснення відеоданих у таких умовах уже не виглядає окремою опцією, скоріше обов’язковим етапом. Без нього потоки просто «з’їдають» мережу й пам’ять, особливо якщо йдеться про онлайн-сервіси або вбудовані пристрої з вузькими ресурсами. Це добре видно на практиці: варто прибрати стиснення, і система починає помітно гальмувати.

На цьому фоні стає зрозуміло, що одного лише «максимального» стиснення замало. Потрібен метод, який одночасно тримає кілька параметрів: коефіцієнт стиснення, швидкість обробки й вимоги до заліза. Класичні кодеки H.264, H.265 дають дуже компактні файли, але ціною складної математики та великої кількості обчислень. На мікроконтролерах або простих одноплатниках реалізувати їх повноцінно майже нереально, особливо якщо треба працювати близько до реального часу.

З іншого боку, є зовсім інший полюс – прості алгоритми на кшталт QOI (The Quite OK Image Format) чи RLE (Run-length encoding). Вони швидкі, невибагливі до ресурсів, їх легко реалізувати навіть у досить простому середовищі. Але за це доводиться платити іншим: коефіцієнт стиснення у багатьох випадках помітно гірший, ніж хотілося б, особливо на великих відеопотоках.

У результаті маємо доволі банальну ситуацію: є важкі, але дуже ефективні кодеки, і є легкі, але слабкі за стисканням алгоритми, а потрібно щось посередині. Потрібен такий метод стиснення, який працює швидше за складні стандарти, але не падає за ефективністю до рівня найпростіших схем. Бажано, щоб його можна було без серйозних переробок використати і у вебзастосунку, і у вбудованій системі з жорсткими обмеженнями по ресурсах. Спроба запропонувати й перевірити саме такий «середній» варіант і є основною метою цієї роботи.

Мета роботи полягає у розробці та дослідженні комбінованого методу стиснення проміжних кадрів відеопотоку. Метод базується на поєднанні попередньої обробки даних з перетворенням у колірний простір YCrCb, стисненні основної області кадру за швидким алгоритмом QOI та стисненні вузлових точок за ентропійним алгоритмом Хаффмана. Робота також має на меті практичну реалізацію та демонстрацію ефективності цього методу на двох цільових платформах: у вебзастосунку та на вбудованій платформі STM32. Поставлена мета вимагає вирішення таких завдань:

1) провести комплексний аналіз предметної області та існуючих методів стиснення статичних зображень і відеоданих, виявити їхні переваги та недоліки;

2) розробити комбінований метод стиснення проміжних кадрів, що поєднує перетворення YCrCb, алгоритм QOI та кодування Хаффмана;

3) реалізувати розроблений метод у вигляді вебзастосунку з клієнт-серверною архітектурою (Next.js, Python) для демонстрації та тестування;

4) адаптувати та реалізувати метод для вбудованої системи на базі мікроконтролера STM32F746G Discovery в умовах жорстких апаратних обмежень;

5) провести тестування та порівняти ефективність (коефіцієнт стиснення) і продуктивність (швидкість обробки) розробленого методу з існуючими аналогами на обох платформах.

Об'єктом дослідження є процес стиснення візуальних даних та методи стиснення проміжних кадрів відеопотоку.

Предметом дослідження є комбінований метод стиснення проміжних кадрів відео, що поєднує перетворення у колірний простір YCrCb, стиснення основної області за алгоритмом QOI та ентропійне кодування вузлових точок за методом Хаффмана.

Запропонований у роботі метод стиснення, по суті, можна розглядати як окремий практичний результат. Він вийшов достатньо гнучким і таким, що його реально масштабувати під різні задачі обробки відео. У вбудованих системах цей підхід можна використовувати там, де потрібне стиснення майже в реальному часі й немає запасу за ресурсами: обмежена пам’ять, невисока тактова частота, жорсткі вимоги до затримок.

У вебзастосунках той самий метод дає змогу зменшити обсяг передаваних даних і, відповідно, навантаження на мережу та сховище. Тобто йдеться не тільки про «красивий алгоритм», а й про дуже прикладні речі: менше трафіку, менше витрат на зберігання. Окремо варто відзначити й розроблений вебінтерфейс. Він не лише демонструє роботу методу, але й може надалі використовуватися як невеликий полігон для подальших експериментів і досліджень у цій темі.

Робота пройшла апробацію на XXVIII Всеукраїнській науково-практичній конференції «Могилянські читання – 2025» з доповіддю «Реалізація комбінованого методу стиснення проміжних кадрів відео у вебзастосунку»; XXVII Всеукраїнській щорічній науково-практичній конференції «Могилянські читання – 2024» з доповіддю «Реалізація комбінованого методу стиснення проміжних кадрів відео на платформі STM32F746G DISCOVERY»; XVII Міжнародній науковій конференції «Ольвійський форум – 2023» з доповідями «Стиснення вузлових точок зображення за допомогою алгоритму Хаффмана» та «Продуктивність ORM для серверних фреймворків вебзастосунків»; XXIII Всеукраїнській науково-

технічній конференції молодих вчених, аспірантів і студентів «Стан, досягнення та перспективи інформаційних систем і технологій» з доповіддю «Алгоритм попередньої обробки зображень для алгоритму QOI»; XXVI Всеукраїнській науково-практичній конференції «Могилянські читання – 2023» з доповіддю «Метод стиснення проміжних кадрів відео»; XXXI Міжнародній науково-практичній конференції «Інформаційні технології: наука, техніка, технологія, освіта, здоров’я» (MicroCAD-2023) з доповіддю «Використання Model-based підходу для мультиагентного навчання з підкріпленням»

Результати дослідження також відображені у публікаціях «Метод стиснення зображень з використанням попередньої обробки та алгоритмів Quite OK Image і Хаффмана» у науковому фаховому виданні «Електронне моделювання», випуск №2 (2024 р.), та «Побудова теплової карти Wi-Fi за допомогою IoT-модуля ESP8266» у науковому фаховому виданні «Комп’ютерно-інтегровані технології: освіта, наука, виробництво», випуск №60 (2025 р.).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У цьому розділі послідовно розглядається предметна область, пов’язана зі стисненням проміжних кадрів відеопотоку. Спочатку обґрунтовується, чому ця задача взагалі важлива саме зараз, з огляду на зростання обсягів візуальних даних. Далі проводиться огляд основних підходів до стиснення, подається їх умовна класифікація, окремо відзначаються сильні та слабкі сторони кожного методу в контексті поставленої задачі. На основі цього аналізу формулюється вибір комбінованого підходу до стиснення. Після цього розглядаються програмно-апаратні засоби, які заплановано використати для реалізації та тестування методу на двох цільових платформах. Завершується розділ переліком функціональних і нефункціональних вимог до системи, що розробляється.

1.1 Огляд предметної сфери та постановка задачі

Останніми роками навіть без якоїсь глибокої аналітики видно, що візуальних даних стало просто надзвичайно багато. Відео і зображення з’являються зараз фактично в кожному сервісі. Є дані, що лише за 2021 рік (уявіть масштаби) з'явилося десь 1,4 трильйона цифрових фото. Причому ця кількість, звісно, продовжує постійно рости. Це, як видно, на практиці доводить одне: робота з великими масивами візуальної інформації це вже звичайна, рутинна технічна задача..

З необробленим відео (у форматі raw) виникають помітні інженерні труднощі, бо воно займає дуже великий обсяг. Зберігати таке складно. Передача подібних потоків сильно перевантажує канали зв’язку, через що мережеве обладнання і сховища працюють, можна сказати, «на межі». З подальшою обробкою теж не легше: що більше даних, то більше треба часу й обчислювальних ресурсів навіть якщо це просто базовий аналіз. Саме тому задача ефективного стиснення зокрема, коли йдеться про проміжні кадри

відеопотоку стає фактично однією з ключових. Без вирішення цього питання проєктувати сучасні системи передачі та обробки відео просто неможливо.

1.1.1 Принципи стиснення відео

Відеопотік можна уявити як послідовність окремих кадрів, тобто статичних зображень, які просто швидко змінюються одне за одним. Через це під час стиснення доводиться працювати не лише з тим, що видно всередині одного кадру, а й з тим, як кадри пов’язані між собою. Саме тут і з’являються два типи надлишковості даних, без яких відео давно було б у рази більшим.

Перший тип просторова або внутрішньокадрова надлишковість (рис. 1.1). Її легко помітити навіть без складних прикладів: великі ділянки одного кольору, як небо, трава чи стіна, майже не змінюються від пікселя до пікселя. Виходить багато повторів. Алгоритми стиснення зображень якраз і намагаються прибрати ці повторення, тобто зменшити те, що дублюється всередині одного кадру.

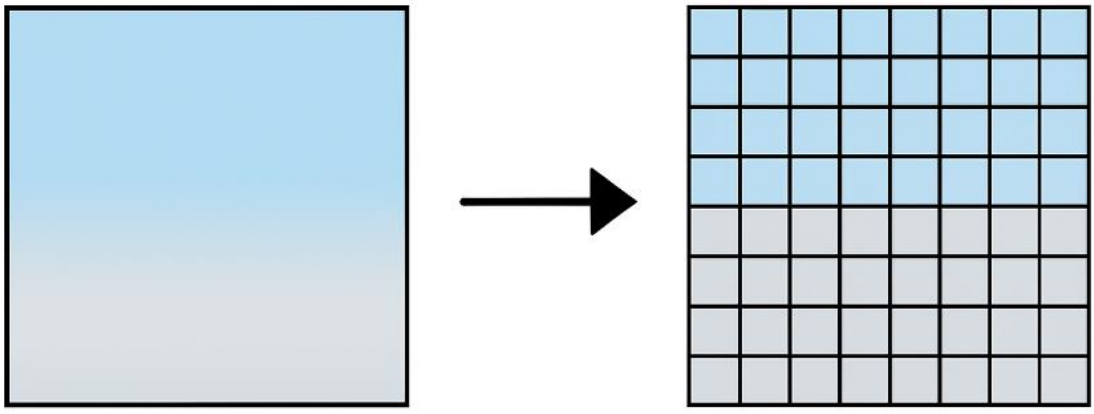


Рисунок 1.1 – Просторова (внутрішньокадрова) надлишковість

Другий тип часова надлишковість (рис. 1.2). Це вже про сам відеопотік. У більшості сцен фон рухається мало або взагалі стоїть на місці, а зміни між кадрами часто мінімальні. Людина теж сприймає це як плавний рух, але для алгоритмів це величезне джерело економії, бо кадри сильно схожі між собою.

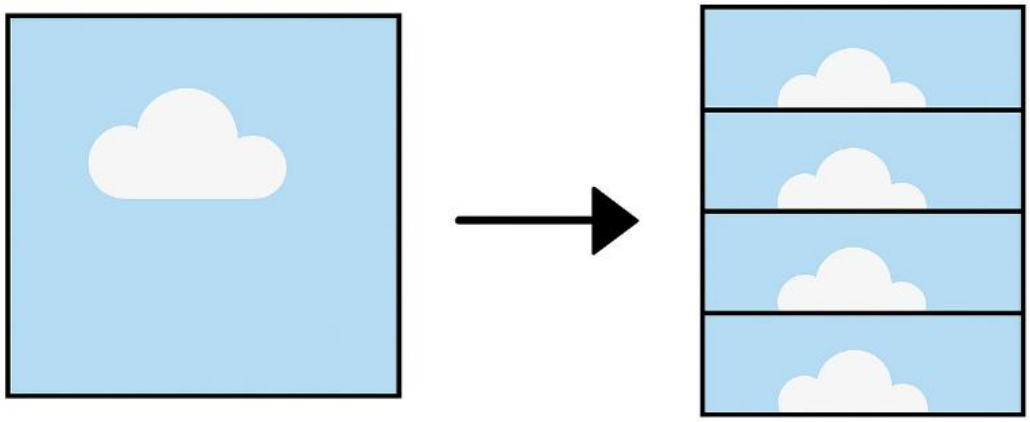


Рисунок 1.2 – Часова (міжкадрова) надлишковість

Завдяки цьому і досягаються високі коефіцієнти стиснення у відеокодеках зміни передаються компактніше, ніж увесь кадр повністю.

1.1.2 Класичний підхід до стиснення відео

Більшість популярних відеокодеків H.264, H.265, AV1 використовують складну комбінацію методів, щоб позбутися обох типів надлишковості. Особливо це стосується часової, бо саме вона дає найбільший виграш. Для цього кадри об’єднують у групи зображень, або GOP (Group of Pictures). Група майже завжди починається з I-кадру (Intra-coded frame). Він стискається як звичайне зображення і не залежить від інших кадрів так простіше відновлювати відео, якщо десь трапляється помилка. Далі йдуть проміжні кадри. Вони вже не зберігають повну інформацію про зображення, а лише те, що змінилося. Через це їх можна стискати значно сильніше, не втрачаючи загальної якості руху. Наступні кадри в групі є проміжними і кодуються значно ефективніше:

1) P-кадри (англ. Predicted frames) кодуються з використанням передбачення від попереднього I- або P-кадру (рис. 1.3);

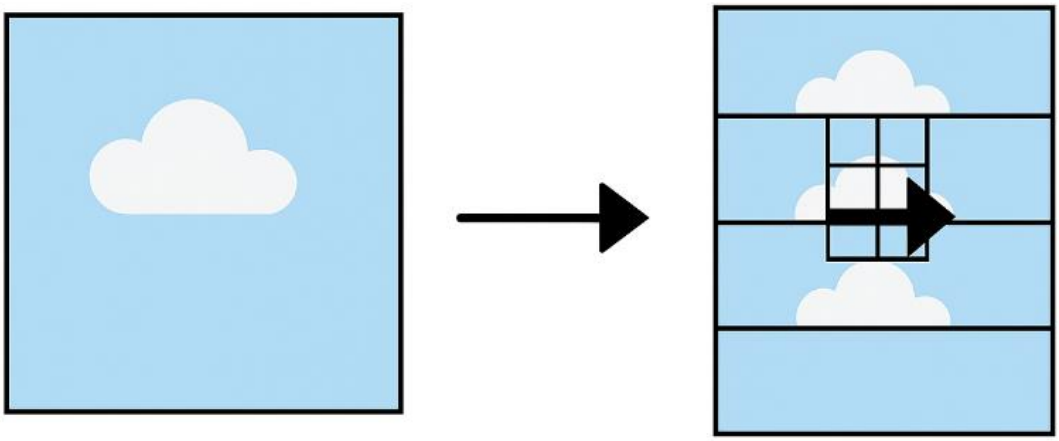


Рисунок 1.3 – Приклад P-кадрів

Замість повного зображення зберігаються лише вектори руху блоків пікселів та «залишкова» інформація – різниця між передбаченим та реальним блоком;

2) B-кадри (англ. Bi-directionally predicted frames) використовують для передбачення як попередні, так і наступні кадри, що дозволяє досягти ще вищого ступеня стиснення (рис. 1.4).

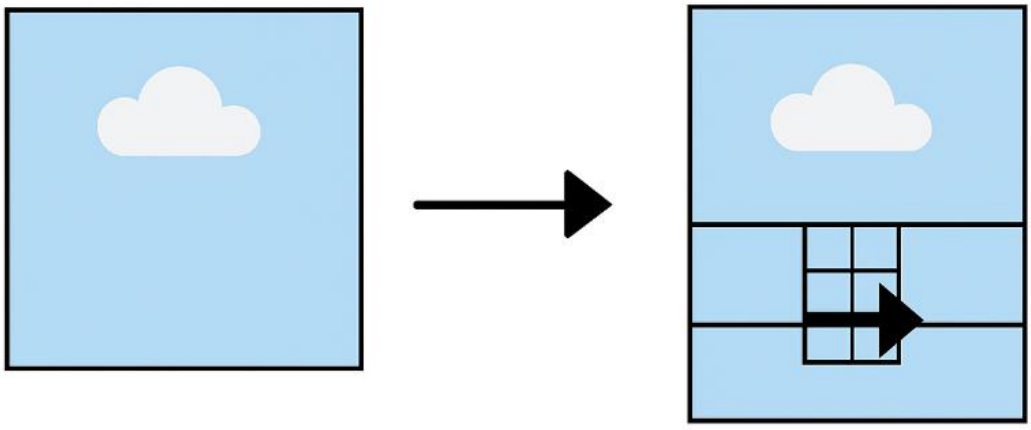


Рисунок 1.4 – Приклад B-кадрів

Цей підхід, що базується на оцінці та компенсації руху, є надзвичайно ефективним, але водночас і обчислювально складним, вимагаючи значних ресурсів процесора та пам'яті.

1.1.3 Постановка задачі

У цій кваліфікаційній роботі основний акцент зроблено саме на стисненні проміжних кадрів відеопотоку. Але замість того, щоб одразу переходити до класичних складних алгоритмів аналізу руху, пропонується інший підхід. Ідея у тому, щоб свідомо відмовитися від усунення часової надлишковості й натомість максимально спростити обробку. Кожен проміжний кадр розглядається як окреме статичне зображення, яке потрібно стиснути якомога швидше й з мінімальними витратами ресурсів.

Така постановка задачі не випадкова. Вона пов’язана з вимогами трьох дуже різних цільових платформ, під які розробляється рішення:

1) вебзастосунок. Тут важливий баланс. З одного боку, потрібен достатньо високий коефіцієнт стиснення, щоб зменшити мережевий трафік і витрати на зберігання даних. З іншого боку, час обробки не може бути надто великим, оскільки користувач не повинен довго чекати на результат. Тобто сервер має встигати обробляти кадри досить швидко, щоб інтерфейс лишався «живим» і зручним;

2) вбудована система на базі STM32. Тут пріоритети частково змінюються. Головне забезпечити обробку в режимі, наближеному до реального часу, та вкладатися в жорсткі обмеження за оперативною пам’яттю. На таких платформах кожен зайвий байт і кожна зайва операція мають значення. Тому деяке зниження ефективності стиснення допускається, якщо це дає приріст у швидкодії та зменшує використання RAM;

3) одноплатний комп’ютер Raspberry Pi 4. Ця платформа займає проміжне місце між класичним сервером і мікроконтролером. З одного боку, Raspberry Pi 4 має суттєво більші ресурси, ніж STM32 (операційна система, багатозадачність, підтримка камер, апаратне прискорення відео), що дозволяє обробляти потокове відео в режимі, близькому до реального часу. З іншого боку, це все ще компактний і відносно обмежений пристрій, який часто працює в автономних або наближених до польових умовах. Тому до алгоритму тут

висуваються дві групи вимог: стабільна робота з потоковим відео (мінімальні затримки, передбачуване навантаження на процесор) та помірні витрати пам’яті й дискового простору при зберіганні стиснених кадрів.

Як видно з цього, центральна проблема дослідження полягає не у пошуку «ідеального» алгоритму з максимально можливим коефіцієнтом стиснення. Наголос робиться на розробці гнучкого, адаптивного методу, який буде певною «золотою серединою». Він має працювати помітно швидше й простіше, ніж складні стандарти на кшталт H.264/H.265, але водночас давати кращий результат, ніж найпримітивніші підходи, наприклад кодування довжин серій (RLE). Важливо, щоб цей метод можна було налаштувати під усі три платформи – вебзастосунок, Raspberry Pi 4 та STM32 – з урахуванням їхніх різних та частково суперечливих вимог до швидкодії, якості відновлення та використання ресурсів.

1.2 Аналіз існуючих методів та технологій стиснення

Перш ніж обрати підхід до стиснення (або тим більше спробувати створити щось своє), треба обов'язково розібратися з базою тим, що вже активно використовується. Технологій для стиснення окремих кадрів є чимало, але по суті їх зручно звести до двох головних груп. Це методи без втрат та методи з втратами.

Різниця між ними не тільки в тому, як саме відбувається стиснення, а й у тому, де такий підхід доречний і на які компроміси можна піти. Зрештою, всюди є свої нюанси.

1.2.1 Методи стиснення без втрат (Lossless)

Методи без втрат це коли алгоритм гарантує, що після декодування зображення відновиться абсолютно повністю. Тобто, біт у біт. Іншими словами, результат буде ідентичний оригіналу без усіляких змін кольорів чи згладжування деталей.

Питання є справді критичним. Особливо у тих сферах, де навіть мінімальна втрата інформації просто неприпустима, бо це одразу створює серйозні проблеми. У таких сценаріях, як видно з практики, підхід «трохи гірше, але менше місця займає» просто не працює, бо ціна можливої помилки занадто висока.

Таким методом є RLE (англ. Run-Length Encoding). Даний метод є одним із найпростіших та найстаріших алгоритмів стиснення. Його ідея полягає у заміні послідовностей однакових значень (серій) парою, що складається зі значення та кількості його повторень. Наприклад, рядок пікселів AAAAABBB може бути закодований як 5A3B (рис. 1.5).

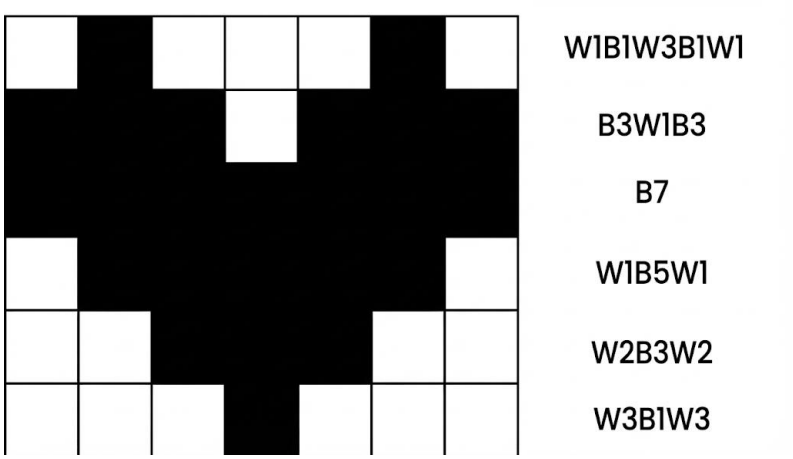


Рисунок 1.5 – Приклад кодування методом RLE

Головний плюс RLE (це, до речі, Run-Length Encoding) у його швидкості. Він майже не навантажує систему. Алгоритм просто проходить по даних лише один раз, ніяких складних обчислень там немає. Через це RLE легко реалізувати. Навіть на дуже простому чи, скажімо, старому обладнанні.

Проблема лише в тому, що ефективність тут як на практиці видно дуже сильно залежить від самих даних. Тобто, що саме ми намагаємося стиснути. Якщо повторень мало, то й сенсу від RLE, по суті, не буде.

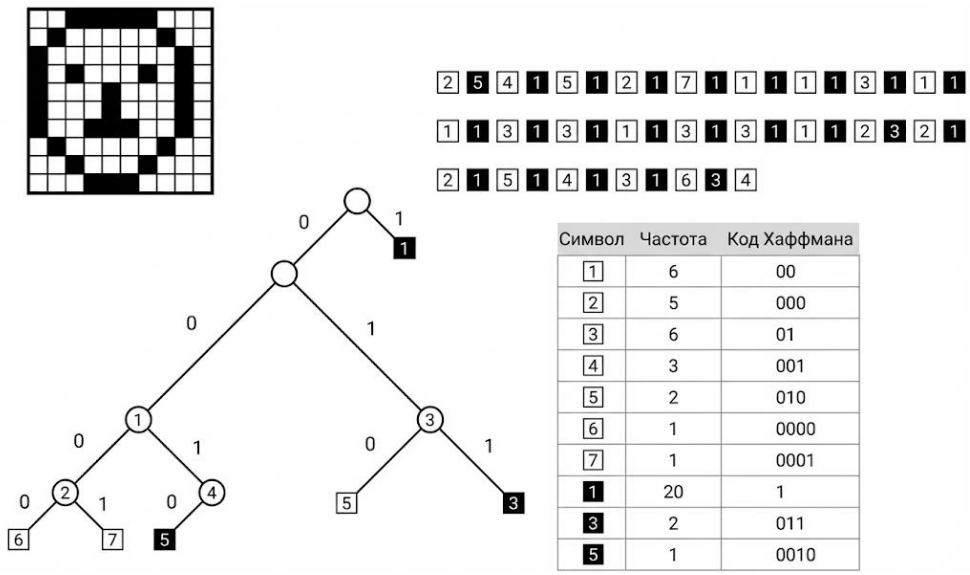
На зображеннях із великими однотонними ділянками RLE показує себе добре. Наприклад, це можуть бути діаграми, схеми, скріншоти інтерфейсів або

проста векторна графіка, де багато однакових пікселів і мало дрібних переходів. У таких випадках коефіцієнт стиснення виходить високим. А от для фотореалістичних зображень ситуація протилежна: сусідні пікселі там рідко мають точно однаковий колір, серії однакових значень короткі, і RLE просто «нема за що зачепитися». У результаті розмір файлу може навіть зрости, тобто замість стиснення отримується фактично негативний ефект.

Алгоритм Хаффмана використовують тоді, коли потрібно стиснути дані без втрат і при цьому не зіпсувати важливу інформацію. У випадку зображень це стосується вузлових точок, бо саме вони впливають на подальше відтворення кадру. Ідея алгоритму досить пряма: символи, які трапляються частіше, отримують коротші бітові коди, а ті, що зустрічаються рідше, навпаки довші. Через це загальний обсяг даних помітно зменшується, хоча сама структура інформації залишається незмінною.

Початок роботи виглядає так: рахується частота появи кожного символу у вхідному наборі. Після цього будується бінарне дерево. У листках розміщуються символи, а їх «вага» відповідає частоті. Побудова рухається знизу: обираються два вузли з найменшими значеннями й об’єднуються в один (в додатку А на рис. А.4). Потім знову два найменші, і так далі, поки не утвориться кореневий вузол.

Кодування відбувається під час обходу дерева. Ліве ребро зазвичай позначають нулем, праве одиницею. Рухаючись від кореня до певного листка, ми «збираємо» його бітовий код. Далі кожен символ у вхідних даних замінюється на свій код Хаффмана, і так формується закодована послідовність (рис. 1.6).



Символ	Частота	Код Хаффмана
1	6	00
2	5	000
3	6	01
4	3	001
5	2	010
6	1	0000
7	1	0001
1	20	1
3	2	011
5	1	0010

Рисунок 1.6 – Приклад кодування зображення деревом Хаффмана

Декодування працює у зворотний бік: біт за бітом алгоритм рухається деревом, поки не дійде до потрібного листка. У цей момент символ додається до вихідної послідовності, і весь процес починається знову вже від кореня. По суті, це звичайний обхід дерева, тільки замість самих символів напрям руху задають біти коду.

Попри те, що метод Хаффмана існує давно і вважається досить простим, питання стиснення вузлових точок досі залишається важливим. Тут недостатньо просто зменшити обсяг даних. Потрібно, щоб після стиснення система все ще могла коректно аналізувати кадри, розпізнавати об’єкти чи працювати з деталями зображення. У тих областях, де помилка навіть у кілька пікселів може вплинути на результат (комп’ютерний зір, системи аналізу відео), це особливо помітно.

Тому оптимізація обробки вузлових точок усе ще залишається актуальною. Тут багато що впирається саме у швидкість і точність. Якщо ці точки обробляються повільно або неточно, «просідає» вся система. Це особливо помітно в задачах, де важливий стабільний темп роботи і прогнозований результат. Через це поєднання класичних підходів, зокрема алгоритму Хаффмана, із більш новими рішеннями на кшталт QOI виглядає

логічним і перспективним. Така комбінація дозволяє зменшити обчислювальні витрати, але при цьому не втратити критичну інформацію. А інколи навіть виграти в точності на рівні практичного застосування алгоритмів обробки зображень.

Ще один дуже поширений формат стиснення зображень без втрат – це Portable Network Graphics (PNG). Його спочатку розробляли як вільну й більш гнучку заміну формату GIF, і зараз він фактично став стандартом для вебграфіки. PNG підтримує кілька режимів представлення зображень: індексовані палітри (до 8 біт на піксель), зображення у відтінках сірого (до 16 біт на піксель) і повноколірні варіанти без палітри (до 48 біт на піксель). Окремо варто згадати підтримку альфа-каналу: 8-бітова прозорість дозволяє будувати зображення з плавними переходами між непрозорими та прозорими областями, що важливо для інтерфейсів і вебдизайну.

Стиснення у PNG відбувається у два основні етапи (рис. 1.7). Спочатку до зображення застосовується попередня фільтрація. Її мета зменшити надмірність у даних, щоб зробити потік пікселів більш «регулярним» і, відповідно, придатним до подальшого стиснення. Після цього вже відфільтровані дані передаються на другий етап, де використовується алгоритм DEFLATE. Він поєднує в собі ідеї LZ77 та кодування Хаффмана: спочатку знаходяться повторювані послідовності, а потім результуючий потік кодується змінною довжиною бітів. У сукупності це дозволяє помітно зменшити розмір файлу, не втрачаючи жодного біта інформації про зображення..

Рисунок 1.7 – Етапи стиснення зображення за допомогою PNG

Формат PNG зазвичай обирають у тих випадках, коли важливо зберегти зображення «як є», без спотворень і артефактів. Його основна риса – стиснення без втрат, тобто після збереження всі дрібні деталі та кольори залишаються такими самими, як в оригіналі. Через це PNG часто застосовують для логотипів, іконок, схем, елементів інтерфейсу, тобто там, де будь-яке розмивання або зміна кольору небажані. Окремо варто згадати підтримку прозорості: формат працює з альфа-каналом, що дає змогу коректно накладати зображення на різні фони, що добре видно на прикладах вебдизайну. PNG є відкритим стандартом, тому без проблем відкривається у більшості сучасних браузерів і графічних редакторів.

Разом з перевагами у PNG є й свої мінуси. Якщо порівнювати з форматами зі стисненням із втратами (типовий приклад – JPEG), ступінь стиснення в PNG зазвичай нижчий. У результаті файли виходять більшими за розміром при такій самій, з точки зору користувача, якості. Це особливо помітно на великих зображеннях або фотографіях. Крім того, алгоритм PNG спочатку не розраховувався на фотографічний контент, де значно краще працюють саме втратні методи. На практиці для зберігання фото частіше використовують JPEG чи подібні формати, а PNG залишають для графіки, де важлива точність відтворення.

Розвиток алгоритмів обробки зображень призвів до того, що для безвтратного стиснення зараз доступний цілий набір рішень – від досить складних до дуже простих. PNG у цьому наборі є лише одним із варіантів, але все ще залишається «дефолтним» інструментом у багатьох прикладних задачах, особливо пов’язаних з вебом і інтерфейсами.

1.2.2 Методи стиснення з втратами (Lossy)

Методи стиснення з втратами працюють інакше. Частина візуальної інформації свідомо відкидається, передусім та, яку людське око помічає найменше. За рахунок цього вдається досягти значно вищих коефіцієнтів стиснення, ніж у безвтратних алгоритмів. Саме тому подібні методи стали стандартом для зберігання фотографій та передачі відео, де розмір файлів і швидкість передавання часто важать не менше, ніж абсолютна точність зображення.

Один із найвідоміших прикладів цього підходу – формат JPEG (Joint Photographic Experts Group). У ньому стиснення відбувається не за один крок, а через послідовність етапів, кожен із яких по-своєму «ужимає» зображення. Спочатку зображення переводиться в колірний простір YCrCb, де компонент Y відповідає за яскравість, а Cr та Cb описують кольорову складову. Це перетворення дозволяє окремо працювати з яскравістю та кольором і

враховувати те, що око чутливіше до змін яскравості, ніж до незначних відхилень кольору (рис. 1.8).

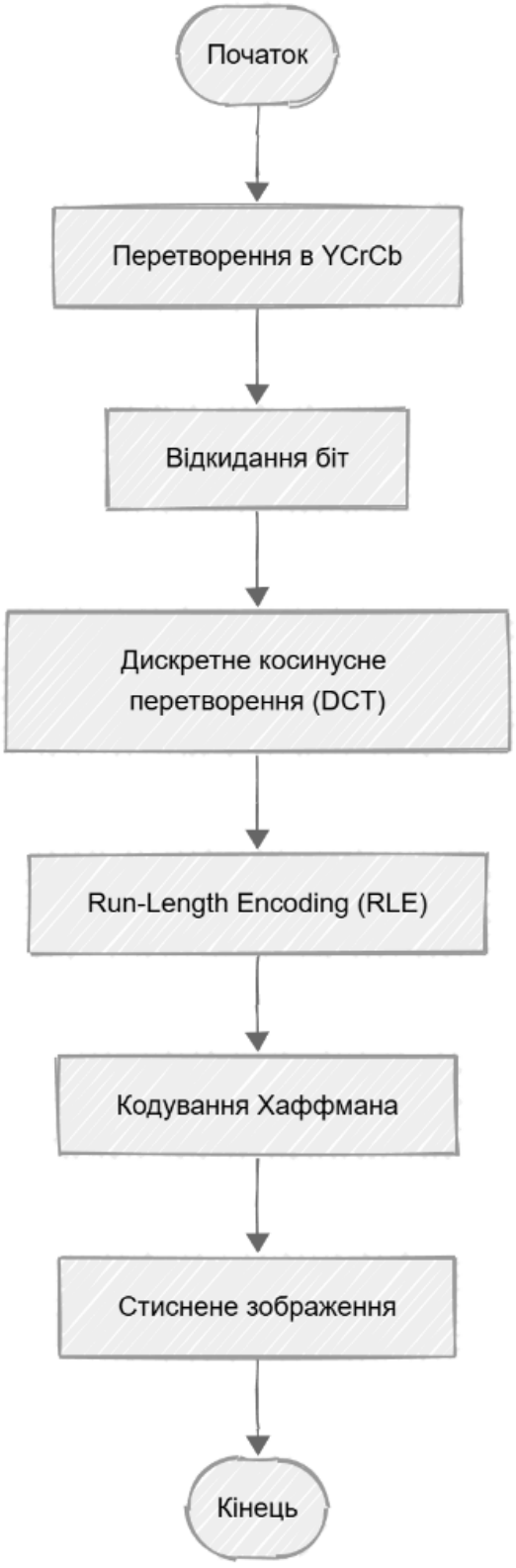


Рисунок 1.8 – Етапи стиснення зображення за допомогою JPEG

На наступному кроці відкидаються найменш значущі біти. Частина дрібних деталей при цьому втрачається, але загальне сприйняття зображення майже не змінюється, зате обсяг даних помітно зменшується. Далі застосовується дискретне косинусне перетворення, яке переводить зображення з просторової області в частотну. Енергія сигналу концентрується у невеликій кількості основних частот, що зручно для подальшого стиснення.

Після цього результати DCT додатково кодуються. Спершу використовується кодування Хаффмана: частіші значення отримують коротші коди, рідкісні – довші, завдяки чому економляться біти. На завершальному етапі застосовується кодування типу Run–Length Encoding, яке стискає послідовності повторюваних значень і ще більше скорочує довжину бітового потоку. У підсумку весь каскад етапів дає досить високий рівень стиснення при прийнятній, з погляду людини, якості зображення.

Розуміння того, як саме працює стиснення в JPEG, потрібне не лише для того, щоб розібрати сам формат. Це ще й нормальна база для ширших речей. Коли стоїть задача зробити новий алгоритм або підправити вже існуючий метод, знання таких класичних підходів реально допомагає. Особливо в тих випадках, де потрібно зменшити обсяг даних, але не «вбити» якість занадто сильно. І тут якраз постійно доводиться шукати цей баланс. Такі компроміси під час роботи з графікою трапляються дуже часто, фактично це вже звична частина задачі.

Також розглянемо відеокодеки (H.264, H.265, AV1). Як було зазначено раніше, сучасні відеокодеки є комплексними стандартами, що поєднують внутрішньокадрове стиснення з втратами (схоже на JPEG для I-кадрів) з надзвичайно ефективними алгоритмами міжкадрового передбачення та компенсації руху для P- та B-кадрів (рис. 1.9).

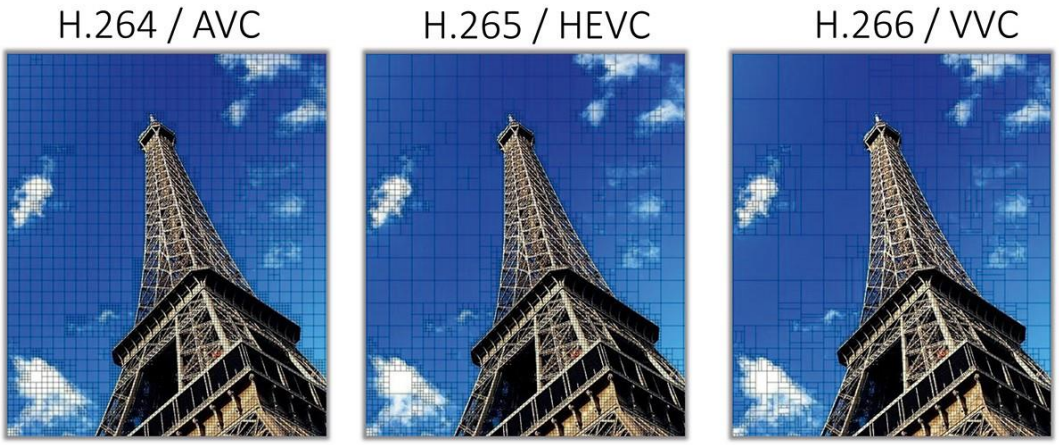


Рисунок 1.9 – Відеокодики H.264, H.265, AV1

Ці кодеки забезпечують найвищий з можливих коефіцієнтів стиснення для відеоданих, що робить їх незамінними для стрімінгу та мовлення. Проте їхня обчислювальна складність є на порядки вищою за складність будь-якого алгоритму стиснення статичних зображень. Ефективна робота цих кодеків у реальному часі, як правило, вимагає спеціалізованих апаратних прискорювачів (кодерів/декодерів), які інтегровані в сучасні процесори та відеокарти. Реалізація такого алгоритму програмно на мікроконтролері класу Cortex-M7 для обробки відео в реальному часі є практично неможливою задачею.

1.3 Формулювання та специфікація вимог до системи

На основі проведеного огляду предметної області та вибраних технологій можна сформувати набір вимог до майбутнього методу і до систем, які його реалізовуватимуть. Йдеться і про функціональні, і про нефункціональні вимоги. Їхнє чітке формулювання допомагає збудувати зрозумілу структуру проєкту і вже на етапі розробки, і під час подальшої перевірки того, наскільки кінцевий результат відповідає задуму.

1.3.1 Функціональні вимоги

Комбінований метод стиснення повинен приймати на вхід растрове 24-бітне зображення у форматі RGB, виконувати перетворення в колірний простір YCrCb та розділяти кадр на основну область і вузлові точки. Основна частина стискається за принципами QOI, а вузлові точки або їх різниці – за алгоритмом Хаффмана. Результатом є єдиний вихідний потік, що містить усі дані для точного відновлення кадру.

Вебзастосунок забезпечує інтерфейс для завантаження відеофайлів або окремих кадрів, налаштування параметрів стиснення (наприклад, розміру блоку), передачу даних на сервер і отримання результатів. Серверна частина виконує обробку кадрів, розраховує статистику – коефіцієнт стиснення, розмір файлів до та після обробки, час виконання – і повертає ці дані клієнту. Клієнтська частина візуально відображає результати, дозволяє порівнювати оригінальне та відновлене зображення й переглядати статистику.

Вбудована система (STM32) повинна приймати кадри з камери або тестового генератора, виконувати стиснення в режимі, наближеному до реального часу, та передавати результати через Ethernet. Плата відображає відновлене зображення на TFT-дисплеї, а також службову інформацію – FPS (frames per second), коефіцієнт стиснення та використання пам’яті.

1.3.2 Нефункціональні вимоги

До основних нефункціональних показників належать продуктивність, ефективність, використання ресурсів та надійність.

На сервері обробка одного кадру Full HD (1920×1080) не повинна перевищувати 200 мс, щоб забезпечити комфортну взаємодію з користувачем. Для STM32 швидкість обробки має досягати не менше 15 кадрів за секунду при роздільній здатності 480×272, тобто не більше 66 мс на кадр.

Метод має демонструвати ефективність стиснення на 5–15 % вищу за чистий алгоритм QOI за допустимого зниження швидкості не більше 20 %.

Використання оперативної пам’яті обмежується 256 кбайт, із максимально ефективним залученням DMA для обміну даними.

Система має бути достатньо стійкою до помилок. Це важливий момент. Наприклад, якщо хтось (користувач) спробує завантажити пошкоджений файл або, припустимо, формат, який у нас не підтримується рішення повинно коректно це обробити.

І, до речі, важливо забезпечити стабільну роботу самого рішення. Причому як у вебзастосунку, так і на вбудованій платформі бо цільова сфера використання досить широка.

Висновки до розділу 1

Отже, у цьому розділі було розглянуто існуючи підходи до стиснення відеоданих. Це надало необхідну теоретичну базу для наступних етапів роботи.

Окремо треба ще раз підкреслити актуальність цієї задачі. По суті, мета зводиться до того, щоб створити такий метод стиснення, який матиме адекватний баланс між швидкодією, вимогами до ресурсів (бо їх завжди мало) та ефективністю. І цей підхід має бути універсальним щоб його можна було використовувати і для веб-платформ, і для вбудованих систем. Це, власне, і є основна ціль

У результаті видно, що жоден окремий метод не закриває одночасно вимоги щодо швидкості, простоти реалізації та якості відновлення даних. Саме це і підвело до потреби у комбінованому підході. Обґрунтовано вибір методу, який поєднує перетворення зображення у колірний простір YCrCb, швидке стиснення основної області алгоритмом QOI та ентропійне кодування вузлових точок методом Хаффмана. Така комбінація забезпечує оптимальний компроміс між швидкістю й ефективністю.

Для реалізації та перевірки методу визначено три технологічні платформи:

1) веборієнтовану систему на базі Python і Next.js для демонстраційних та аналітичних цілей;

2) одноплатний комп’ютер Raspberry Pi 4 як проміжну платформу для роботи з потоковим відео в умовах обмежених, але все ж суттєвіших, ніж у мікроконтролера, ресурсів;

3) вбудовану платформу STM32F746G Discovery для оцінки функціонування алгоритму в жорстких обмеженнях за пам’яттю й обчислювальними можливостями.

Сформульовано функціональні й нефункціональні вимоги до всіх трьох реалізацій, у тому числі конкретні показники продуктивності та ефективності, які визначатимуть критерії успішності проєкту. Підсумовуючи, результати першого розділу створюють базу для переходу до наступного етапу – проєктування архітектури та розроблення програмної реалізації комбінованого методу стиснення, що буде розглянуто в наступному розділі.

2 КОМБІНОВАНИЙ МЕТОД СТИСНЕННЯ ПРОМІЖНИХ КАДРІВ

2.1 Обґрунтування вибору комбінованого методу

Аналіз сучасних методів стиснення зображень і відео показує, що жоден окремий алгоритм не забезпечує одночасно високий коефіцієнт стиснення, прийнятну якість відновлення та низьку обчислювальну складність в умовах обмежених ресурсів вбудованих систем. Формати JPEG, PNG, WebP, спеціалізовані відеокодеки, а також нові «легкі» формати мають власні сильні сторони, але їх застосування як є в проміжних кадрах вбудованих систем не завжди є оптимальним.

Тому перспективним підходом є комбінування декількох методів у єдиний конвеєр, де кожен етап виконує «свою» задачу й працює з тим типом даних, для якого він найбільш ефективний. У даній роботі запропоновано комбінований метод стиснення проміжних кадрів відеопотоку, який поєднує:

1) попереднє перетворення кольорового простору з RGB у YCrCb;
2) структурне спрощення кадру через блокове усереднення та виділення вузлових точок;
3) стиснення основної області кадру без втрат за допомогою алгоритму QOI;
4) ентропійне кодування масиву вузлових точок за допомогою алгоритму Хаффмана.

Основна ідея тут тримається на кількох послідовних кроках, які разом дають помітний виграш у швидкості та розмірі даних. Спочатку кадр переводять у такий колірний простір, де стискати його простіше й логічніше (для подальших операцій це справді важливо). Потім зображення наче «спрощують» до набору опорних, або вузлових, точок. На них і будується згладжене наближення всього кадру. Решта пікселів, тобто основний масив, проходить через дуже швидкий алгоритм без втрат, який майже не навантажує систему. А вже маленький набір вузлових точок стискається максимально

щільно, за допомогою ентропійного кодування. І, як видно на практиці, саме така комбінація дозволяє втримати баланс між якістю та продуктивністю.

Таке розділення ролей є виправданим: QOI забезпечує високу швидкодію та хороше стиснення для згладжених областей кадру, тоді як Хаффман дозволяє наблизитися до теоретичної межі ентропії для компактного масиву опорних значень. У випадку проміжних кадрів відео, в яких значна частина інформації може бути виражена через відхилення від опорних кадрів, таке рішення особливо доцільне й дозволяє очікувати вищий коефіцієнт стиснення порівняно з використанням лише традиційних форматів (JPEG, PNG тощо).

Далі послідовно розглянуто структурно-алгоритмічні аспекти кожного етапу комбінованого методу.

2.2 Попередня обробка зображення (перетворення в YCrCb)

Перший етап комбінованого методу – попередня обробка кадру, метою якої є виділення основних компонент та зниження надлишковості даних перед власне стисненням. Вхідні кольорові зображення, як правило, задані в моделі RGB (Red–Green–Blue), яка пов’язана з апаратним способом формування зображення на екранах, але не є оптимальною для стиснення.

Тому виконується перехід до колірного простору YCrCb, де:

– Y – яскравість (luma);

– Cb – відхилення синьої компоненти від яскравості;

– Cr – відхилення червоної компоненти від яскравості.

На відміну від RGB, простір YCrCb розділяє інформацію про світлість та колір, що дозволяє обробляти їх окремо. Для 8-бітного подання пікселів (0–255) однією з поширених апроксимацій перетворення є:

$$Y = 0{,}299R + 0{,}587G + 0{,}114B, \quad (2.1)$$

$$Cb = -0{,}1687R - 0{,}3313G + 0{,}5B + 128, \quad (2.2)$$

$$Cr = 0{,}5R - 0{,}4187G - 0{,}0813B + 128, \quad (2.3)$$

де R, G, B– значення компонент пікселя в моделі RGB, а Y, Cb, Cr– відповідні значення в просторі YCrCb. Додавання 128 до Cbта Crзміщує діапазон цих компонент у невід’ємну область цілих чисел.

Перетворення може виконуватися як у вигляді матричного множення, так і за допомогою оптимізованих процедур з використанням цілочисельної арифметики. На рис. 2.1 схематично показано перехід від RGB до YCrCb та зміну розподілу значень після перетворення.

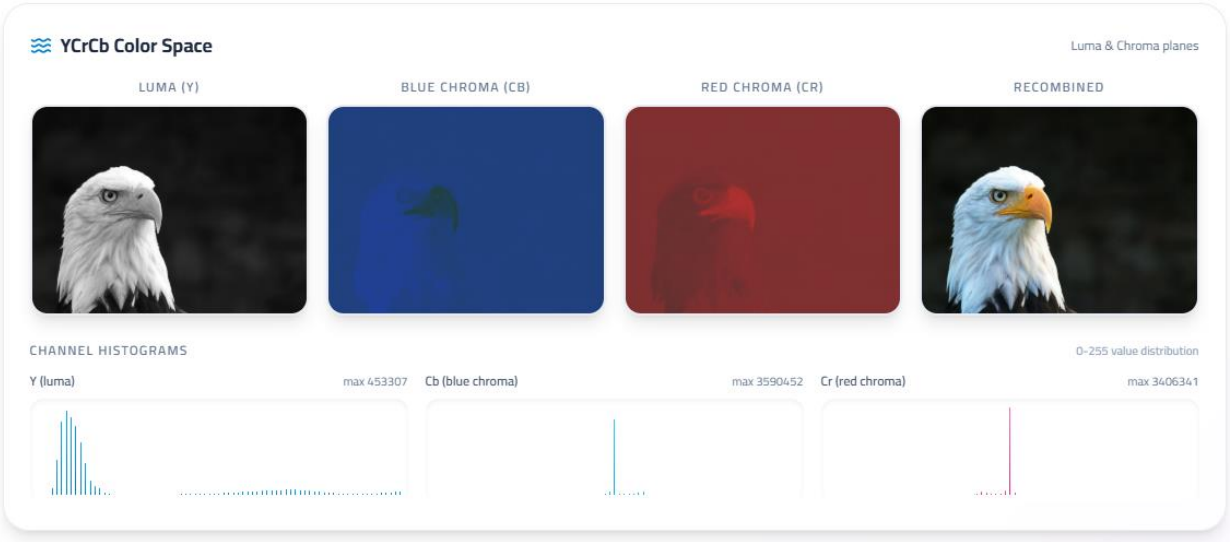


Рисунок 2.1 – Перетворення з RGB в YCrCb; гістограма розподілу значень

Перехід до YCrCb має дві ключові переваги:

– з урахуванням особливостей зору людини (вища чутливість до змін яскравості) можна більш агресивно обробляти колірні компоненти, майже не погіршуючи візуальне сприйняття;

– зменшується статистична корельованість між каналами: Yстає відносно незалежною від Cbта Cr, що означає меншу дубльованість інформації й кращі умови для подальшого стиснення.

На виході цього підетапу кадр подається у вигляді трьох матриць значень $Y(x, y)$, $Cb(x, y)$та $Cr(x, y)$. Далі одна й та сама процедура структурного спрощення застосовується окремо до кожної з них; для стислості нижче процес описується на прикладі яскравісної компоненти Y.

2.3 Виділення вузлових точок та інтерполяційна реконструкція

Далі зображення навмисно спрощують. Його розбивають на блоки на такі собі невеликі фрагменти. Для кожного блоку вираховують вузлову точку, як правило, це середнє значення.

Через це виходить, що ми зменшуємо обсяг даних, які потрібно зберігати з високою точністю. Кадр стає більш «рівним». Дрібні деталі, які, можливо, є просто шумом, частково згладжуються, і це добре. Усе це, в загальному вигляді, можна представити такими крокам:

2.3.1 Розбиття на блоки

Кадр розміром $W \times H$пікселів ділиться на блоки фіксованого розміру $N \times N$(типові значення $N = 4$або $N = 8$). Вибір Nвизначає компроміс: великі блоки дають менше вузлових точок і більший ступінь стиснення, але можуть спричиняти втрату дрібних деталей і поява блокових артефактів; малі блоки краще зберігають структуру, але збільшують обсяг опорних даних.

2.3.2 Обчислення опорних значень

Для кожного блока обчислюється середнє значення пікселів:

$$Y_{node} = \frac{1}{N^2} \sum_{x=i}^{i+N-1} \sum_{y=j}^{j+N-1} Y(x, y), \qquad (2.4)$$

де $Y(x, y)$– значення компоненти в пікселі з координатами (x, y)в межах блока, а Y_{node}– вузлова (опорна) яскравість блока з лівим верхнім кутом (i, j). Аналогічно обчислюються Cb_{node} та Cr_{node}. Таким чином вузлова точка фактично є тріадою $(Y_{\text{node}}, Cb_{\text{node}}, Cr_{\text{node}})$.

Для спрощення реалізації у вбудованих системах замість точного середнього можливі наближення (наприклад, вибір центрального пікселя блока), однак у цій роботі розглядається саме усереднення, яке забезпечує найкраще згладжування.

2.3.3 Квантування (відкидання молодших бітів)

Отримані середні значення можуть бути додатково квантизовані. Для цього відкидається певна кількість найменш значущих бітів (LSB) (рис. 2.2):

$$Y_q = \left(\frac{Y_{node}}{2^k}\right) \cdot 2^k, \tag{2.5}$$

де k– кількість відкинутих біт, а Y_{q}– квантоване значення. Аналогічна операція виконується для Cb_{node}та Cr_{node}. На рисунку 2.3 наведено приклад агресивного відкидання LSB.

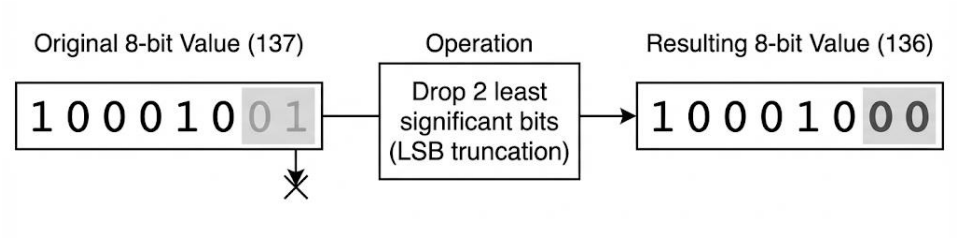


Рисунок 2.2 – Відкидання найменш значущих бітів у вузлових точках

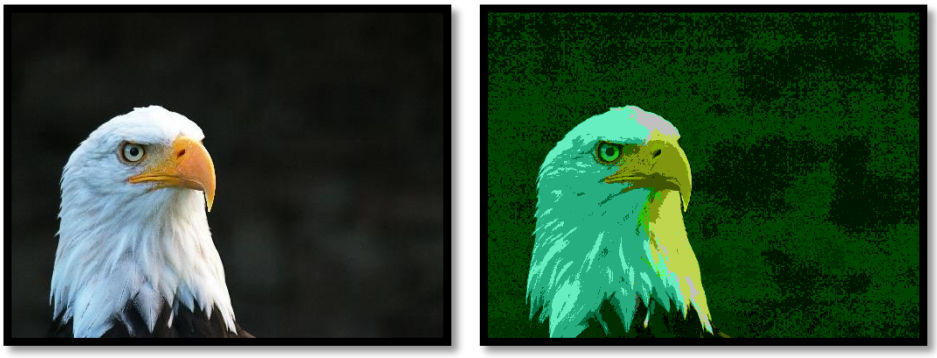


Рисунок 2.3 – Приклад агресивного відкидання біт *(blockSize: 4px; bits: 6)*

Наприклад, якщо середнє значення яскравості дорівнює 137 (0x89) і $k = 2$, після квантування воно стане 136 (0x88). Чим більше біт відкидається, тим вищий потенційний ступінь стиснення, але тим більша похибка у відновленому зображенні. Значення kобирається експериментально; для проміжних кадрів допустиме невелике зниження точності заради зменшення обсягу даних.

2.3.4 Формування масиву вузлових точок

Усі обчислені опорні значення утворюють окремий масив вузлових точок розміром приблизно $\lceil W/N \rceil \times \lceil H/N \rceil$. Таким чином розмірність даних зменшується приблизно в N^2разів. Попри втрату частини деталей, цей масив зберігає основну структуру та колірні характеристики кадру.

2.3.5 Інтерполяційна реконструкція

Для побудови наближеного зображення на основі вузлових точок використовується двовимірна білінійна інтерполяція. Значення будь-якого пікселя всередині блока обчислюється як лінійна комбінація чотирьох найближчих вузлових значень по горизонталі та вертикалі. Результат операції проілюстровано на рис. 2.5 та рис. 2.6.

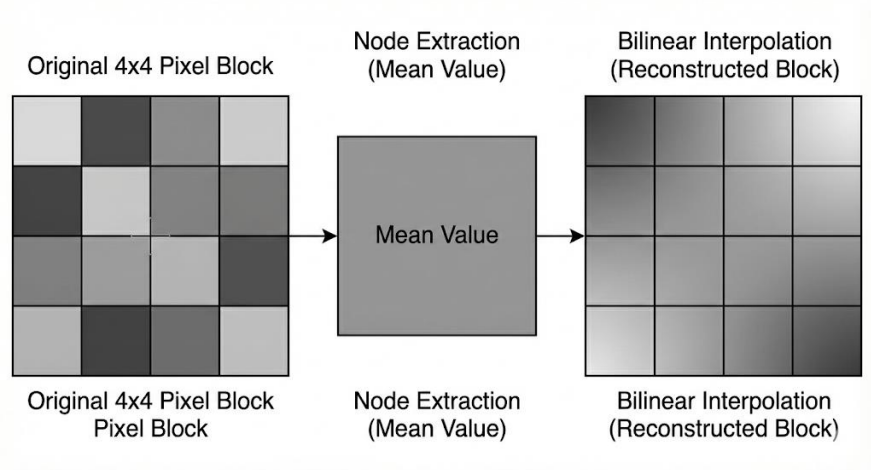


Рисунок 2.5 – Ілюстрація виділення вузлових точок та інтерполяційного відновлення кадру

Рисунок 2.6 – Приклад агресивного виділення вузлових точок та інтерполяційного відновлення кадру *(blockSize:22px bits: 2)*

Інтерпольоване зображення виступає таким собі згладженим варіантом початкового кадру. Різкі переходи між сусідніми пікселями пом’якшуються, дрібний шум майже зникає, а якісь дрібні текстури інколи взагалі «розчиняються». Через це загальна ентропія кадру падає, і він стає набагато зручнішим для подальшого безвтратного стиснення. На цьому етапі фактично залишаються дві групи даних, з якими доведеться працювати далі:

1) масив вузлових точок з розмірністю, суттєво меншою за вихідний кадр;

2) інтерпольоване (згладжене) зображення, що містить основну область кадру.

Далі ці групи стискаються окремими, спеціально підібраними алгоритмами.

2.4 Стиснення основної області кадру за алгоритмом QOI

Для стиснення основної маси пікселів інтерпольованого кадру використовується алгоритм QOI (Quite OK Image Format). Це сучасний безвтратний формат, розроблений як альтернатива PNG з фокусом на простоті реалізації та високій швидкодії. За ступенем стиснення QOI наближається до PNG, а за швидкодією істотно його перевищує, що особливо важливо у вбудованих системах.

Алгоритм QOI працює в один прохід, послідовно обробляючи пікселі зліва направо та зверху вниз. Для кожного пікселя обирається один з кількох режимів кодування залежно від його значення та контексту:

1) *QOI_OP_RUN* – кодування серій однакових пікселів (run-length);

2) *QOI_OP_INDEX* – кодування через індекс у буфері нещодавно використаних кольорів (64-елементний кеш);

3) *QOI_OP_DIFF* та *QOI_OP_LUMA* – компактне кодування невеликих різниць кольору між поточним та попереднім пікселями;

4) *QOI_OP_RGB(A)* – пряма передача значення пікселя, коли жоден з оптимізованих режимів не підходить.

Кожен фрагмент потоку починається з байта-тегу, що визначає режим кодування, а далі слідують байти даних (індекс або дельти). Усі операції вирівняні по байтах, QOI не використовує дробові біти, що спрощує реалізацію декодера (рис. 2.5).

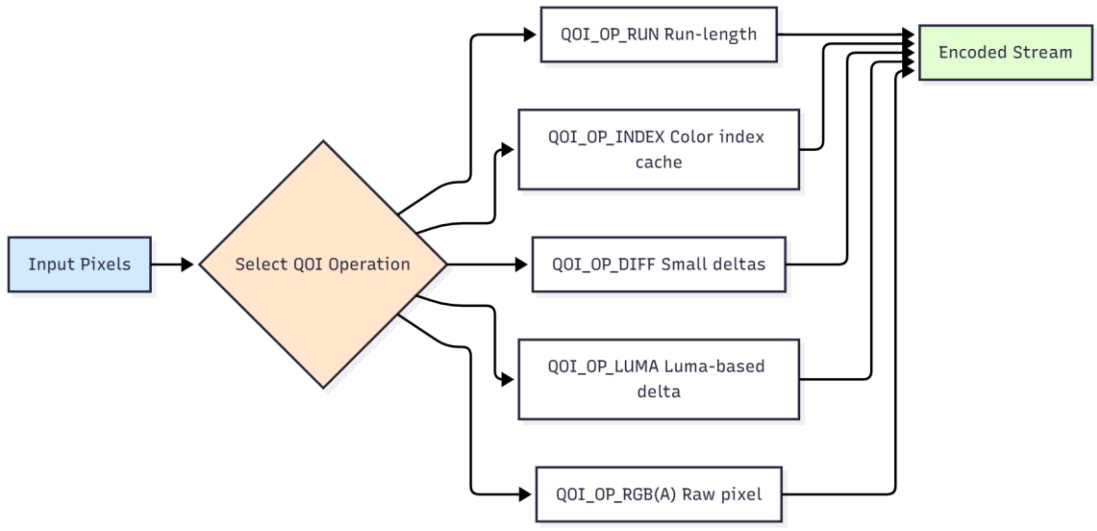


Рисунок 2.5 – Принципова схема кодування зображення алгоритмом QOI

Попередня обробка, описана в підрозділі 2.3, створює для QOI «сприятливе середовище»: інтерпольоване зображення містить багато однорідних або слабозмінних областей, тому алгоритм частіше може застосовувати режими RUN і DIFF, отримуючи компактний поток даних. На

вхід QOI у рамках комбінованого методу подається згладжене зображення, представлено послідовністю пікселів у вибраному форматі (наприклад, RGB або YCbCr після відповідного перетворення).

У практичній реалізації додатково застосовується формування залишкового зображення за предиктором Паефа (Paeth predictor): спочатку з інтерпольованого кадру та поточного варіанту відновлення обчислюються залишки, а вже до них застосовується QOI-кодування. Це ще більше знижує ентропію послідовності пікселів і покращує ступінь стиснення, залишаючись при цьому в межах безвтратної схеми декодування.

На виході етапу отримується компактний двійковий блок, що описує всі не опорні пікселі кадру. Оскільки алгоритм QOI є безвтратним, відновлений кадр у точності відповідає інтерпольованому варіанту, поданому на вхід (усі втрати якості пов’язані лише з попередніми етапами усереднення та квантування). Детальну схему принципу кодування можна переглянути у додатку А, рисунок А.2.

2.5 Стиснення вузлових точок за алгоритмом Хаффмана

Окрема задача – стиснення масиву вузлових точок, що містить опорні значення яскравості та кольорових компонент по блоках. Для цього використовується алгоритм Хаффмана, який забезпечує побудову оптимального префіксного коду для заданих частот символів.

У ролі «символів» в даному випадку виступають:

- або абсолютні значення вузлових точок ($Y_{\text{node}}, Cb_{\text{node}}, Cr_{\text{node}}$),
- або різниці між сусідніми вузловими значеннями.

Практичний досвід показує, що другий підхід є кращим. Тут логіка така: сусідні блоки зображення, як правило, мають дуже схожі середні значення. А це означає, що різниця між цими значеннями буде мінімальною.

Виходить, що розподіл різниць концентрується біля нуля. А такі дані, як відомо, дуже зручно стискати, якщо використовувати ентропійні методи. Власне, підготовка даних до самого кодування включає такі кроки:

2.5.1 Обчислення різниць

Масив вузлових значень обходиться у фіксованому порядку (наприклад, рядок за рядком). Для кожного елемента обчислюється різниця з попереднім. Це формує послідовність дельт, у якій багато малих (у тому числі нульових) значень.

2.5.2 Кодування довжин серій (RLE)

Для ще кращого стискання послідовність дельт можна додатково обробити за допомогою простого кодування довжин серій (Run-Length Encoding): довгі послідовності однакових дельт (перш за все нулів) замінюються парою «значення + довжина». Такий прийом добре узгоджується з подальшим застосуванням Хаффмана.

2.5.3 Формування таблиці частот

Для отриманої послідовності обчислюються частоти появи кожного можливого значення. У випадку 8-бітових дельт кількість різних символів не перевищує 256, що дозволяє ефективно будувати частотну таблицю.

2.5.4 Побудова дерева Хаффмана

Далі все робиться за класичною схемою: беремо два символи, які трапляються найрідше (тобто найменш частотні), і об'єднуємо їх. Це створює новий вузол, частота якого сумарна частота цих двох символів.

Потім операцію просто повторюють і так доти, доки не буде повністю побудоване дерево. Шлях від кореня до кінцевого листка (символу) визначає його двійковий код.

Логіка тут проста: символи, які використовуються частіше, опиняються ближче до кореня. Відповідно, вони отримують коротші коди. Ті, що рідкісні, навпаки матимуть довші кодові слова.

Для компактного зберігання структури дерева замість явного дерева використовується канонічний код Хаффмана: зберігається список символів та довжини їхніх кодів, а самі коди відновлюються декодером за фіксованим правилом. Це зменшує обсяг службової інформації та спрощує реалізацію (схему кодування можна переглянути в додатку А, рисунку А.3).

Вихідна послідовність дельт вузлових значень замінюється відповідними кодовими словами Хаффмана. Середня довжина кодування:

$$L_{\text{avg}} = \sum_i p_i \cdot l_i, \tag{2.6}$$

де p_i– ймовірність появи i-го символу, l_i– довжина його коду, наближається до ентропії джерела:

$$H = -\sum_i p_i \log_2 p_i. \tag{2.7}$$

Таким чином забезпечується майже мінімально можлива кількість біт на одне опорне значення.

Опорні значення після усереднення й квантування зазвичай містять велику кількість повторюваних величин (особливо в однорідних областях кадру). При використанні різницевого підходу та RLE ця властивість проявляється ще сильніше: велика кількість нульових дельт кодується дуже короткими кодами. Експериментальні дослідження показують, що застосування Хаффмана до різниць вузлових точок дозволяє зменшити обсяг даних у кілька разів порівняно з прямим зберіганням опорних значень (рис. 2.7).

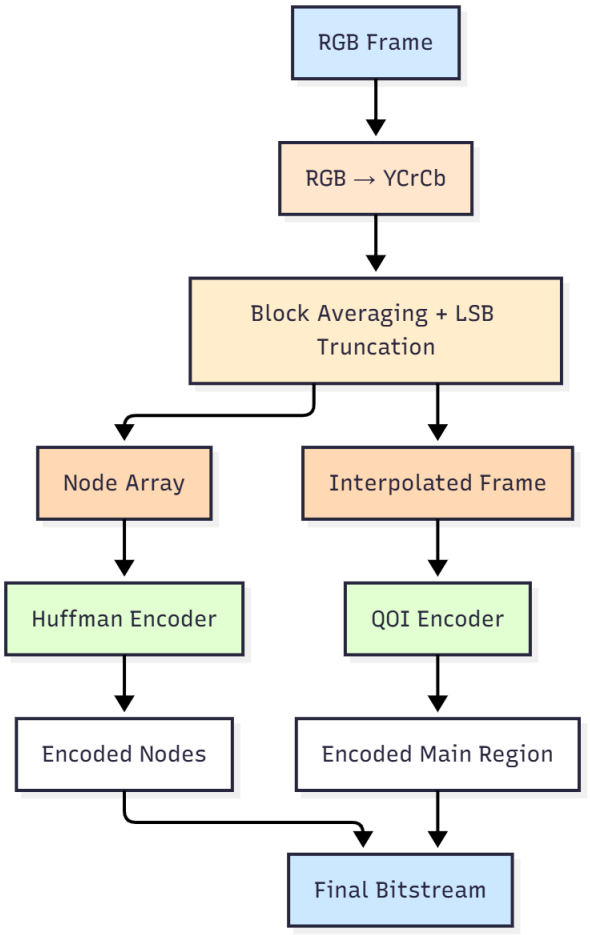


Рисунок 2.7 – Загальна архітектура комбінованого методу

На виході етапу формується другий блок стиснених даних – двійковий поток, який містить:

- службову інформацію для відновлення коду;
- послідовність кодових слів, що відповідають вузловим точкам.

Разом із результатом QOI-кодування основної області кадру цей блок утворює підсумковий бітовий потік комбінованого кодека.

Висновок до розділу 2

У даному розділі розроблено та детально описано комбінований метод стиснення зображень (проміжних кадрів відеопотоку), який поєднує:

1) попереднє перетворення кольорового простору з RGB у YCrCb з урахуванням особливостей зору людини;

2) блокове усереднення та виділення вузлових точок із можливістю квантування для зменшення ентропії;

3) інтерполяційну реконструкцію кадру, що створює згладжене наближення зменшеної розмірності;

4) швидке безвтратне стиснення основної області кадру за алгоритмом QOI (з можливим використанням предиктора Паефа для формування залишків);

5) ентропійне стиснення масиву вузлових точок за алгоритмом Хаффмана з попереднім різницевим та RLE-кодуванням.

Показано, що загальна архітектура методу має дворівневу структуру: на першому рівні виконується підготовка даних (згладжування зображення при збереженні опорної інформації), на другому – стискання підготовлених даних алгоритмами, оптимально підібраними під їхні статистичні властивості. Такий підхід дозволяє очікувати високий коефіцієнт стиснення проміжних кадрів при збереженні достатньо високої якості відновлення та прийнятної обчислювальної складності для вбудованих систем.

Розроблений комбінований метод є основою для подальшої програмної та апаратної реалізації в складі апаратно-програмного комплексу стиснення проміжних кадрів та вебзастосунку. У наступному розділі буде наведено результати експериментального дослідження його ефективності та порівняння з базовими методами стиснення.

3 АПАРАТНО-ПРОГРАМНА РЕАЛІЗАЦІЯ

Для практичної реалізації, дослідження та валідації розробленого комбінованого методу було обрано два принципово різні технологічні стеки, що відповідають двом цільовим платформам.

Такий підхід дозволяє не лише продемонструвати логічну коректність алгоритму в середовищі швидкого прототипування, але й довести його практичну життєздатність в умовах жорстких апаратних обмежень.

3.1 Реалізація у вебзастосунку

Розроблений вебзастосунок реалізовано на сучасному фреймворку Next.js з використанням мови TypeScript. Next.js (побудований над React) забезпечує високу продуктивність інтерфейсу завдяки можливостям серверного рендерингу та статичної генерації сторінок. Інтерфейс працює переважно як SPA (Single Page Application) з динамічним оновленням контенту на клієнті, але за потреби може використовувати і серверні функції для важких обчислень (рис. 3.1).

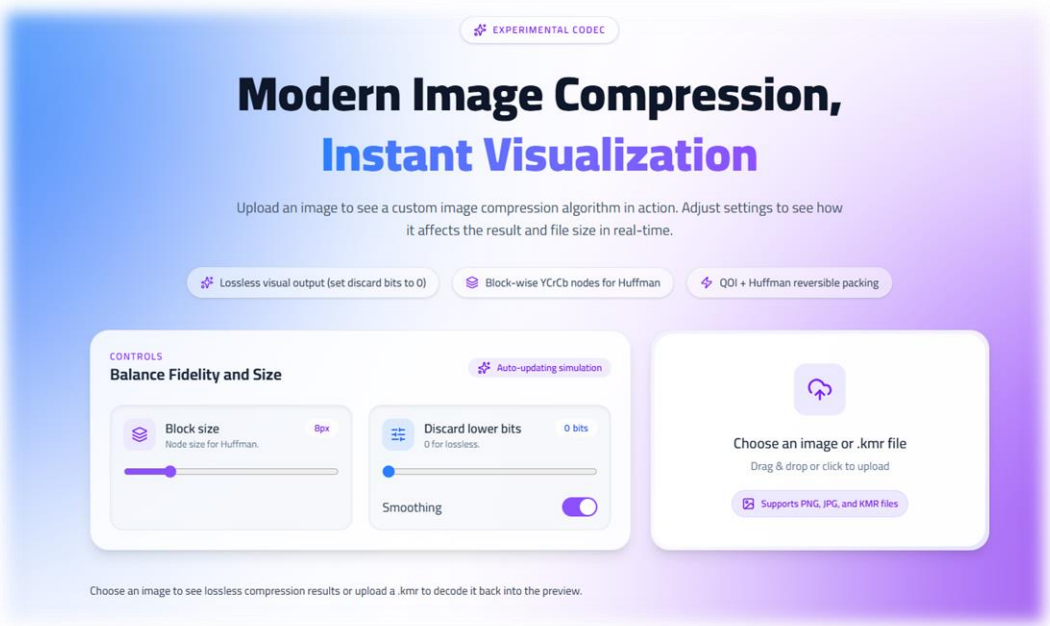


Рисунок 3.1 – Загальний вигляд вебзастосунку

Статична типізація TypeScript підвищує надійність коду та спрощує підтримку. В результаті реалізовано інтерактивний інтерфейс для завантаження зображень, налаштування параметрів компресії та візуалізації результатів у режимі реального часу.

3.1.1 Огляд вебінтерфейсу

Перший рівень – панель керування та вводу «Controls & Input». У верхній частині сторінки виводиться заголовок із коротким описом пайплайну: «RGB → YCrCb → QOI → Huffman». Тобто користувач одразу бачить, з яких етапів складається алгоритм, і розуміє, що далі він працює не з абстрактними «повзунками», а з конкретними місцями в цьому ланцюжку перетворень.

Нижче всередині цього блоку розташовано секцію «Balance fidelity and size» – головний «пульт» керування якістю. Тут два основні повзунки: «Block size» (розмір блоку для вибірки вузлових точок) та «Discard lower bits» (кількість молодших біт, які відкидаються для вузлів) (рис.3.2).

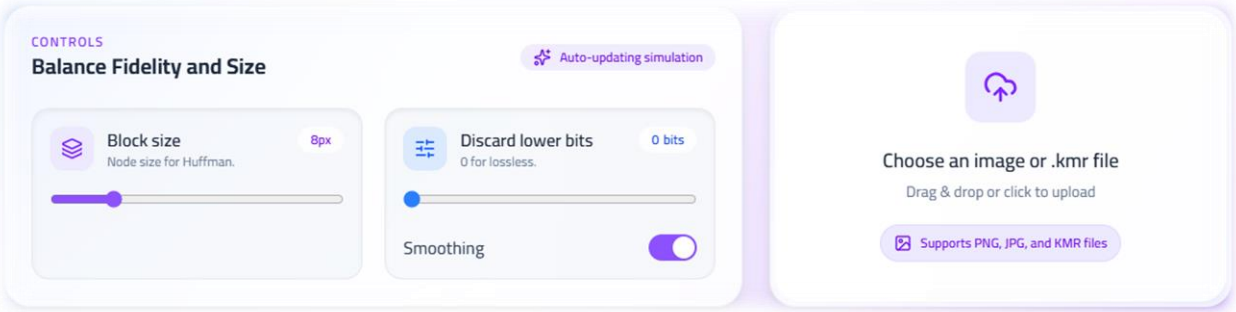


Рисунок 3.2 – Верхній блок «Controls & Input»: заголовок пайплайну, секція «Balance fidelity and size» та зона завантаження файлів

Зліва відображається поточне значення параметра, а праворуч – короткий текст-підказка. Праворуч від блоку налаштувань знаходиться зона завантаження файлів «Drag and drop or choose an image or .kmr»: у неї можна

перетягнути PNG/JPEG-зображення для кодування або власний контейнер .kmr для зворотного декодування.

Під панеллю керування розташовано блок миттєвої статистики «Summary Stats». Тут зібрано три головні показники, які цікавлять користувача одразу після обробки: коефіцієнт стиснення «Compression Ratio», розміри файлів Raw vs Compressed та «JS Time» – час, за який браузер виконав усі обчислення.

Ці значення оновлюються щоразу після зміни параметрів або завантаження нового зображення. По суті, це «спідометр» кодека: можна одним поглядом зрозуміти, наскільки агресивно працює компресія і скільки мілісекунд вона забирає на конкретній машині (рис. 3.3).

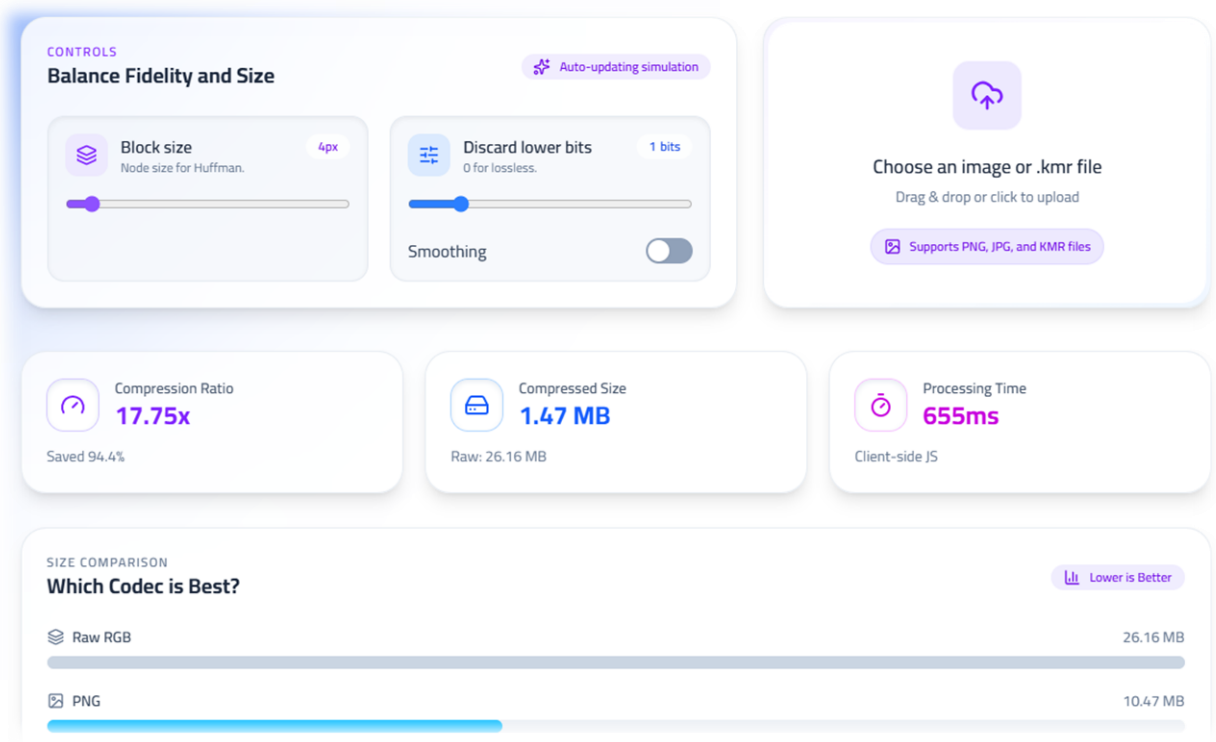


Рисунок 3.3 – Блок «Summary Stats» із показниками «Compression Ratio», «Raw vs Compressed» та «JS Time»

Наступний рівень – аналітичний блок «Deep Dive» (рис. 3.3–3.4). Тут вже цілий набір інструментів для розбору поведінки алгоритму. Зліва знаходиться

діаграма «Size Comparison», де в один ряд виводяться розміри файлів для різних форматів: вихідний Raw, PNG, JPEG, WebP і Custom codec (рис. 3.4).

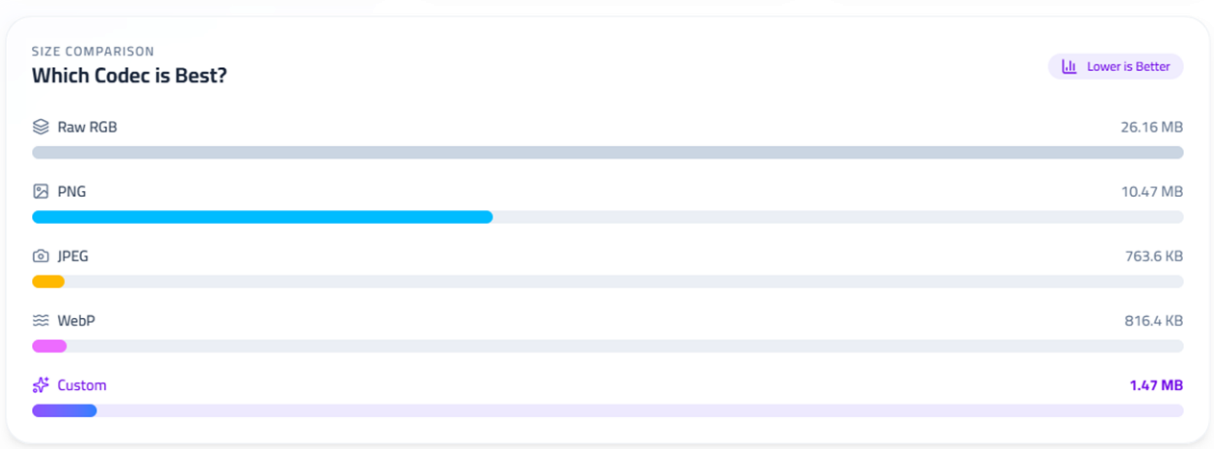


Рисунок 3.4 – Аналітичний блок «Deep Dive»: порівняння форматів

Внизу підписані конкретні значення в кілобайтах. Поруч розташований блок «Quality metrics» з метриками PSNR та SSIM, які показують, наскільки сильно відрізняється відновлене зображення від оригіналу (рис. 3.5).

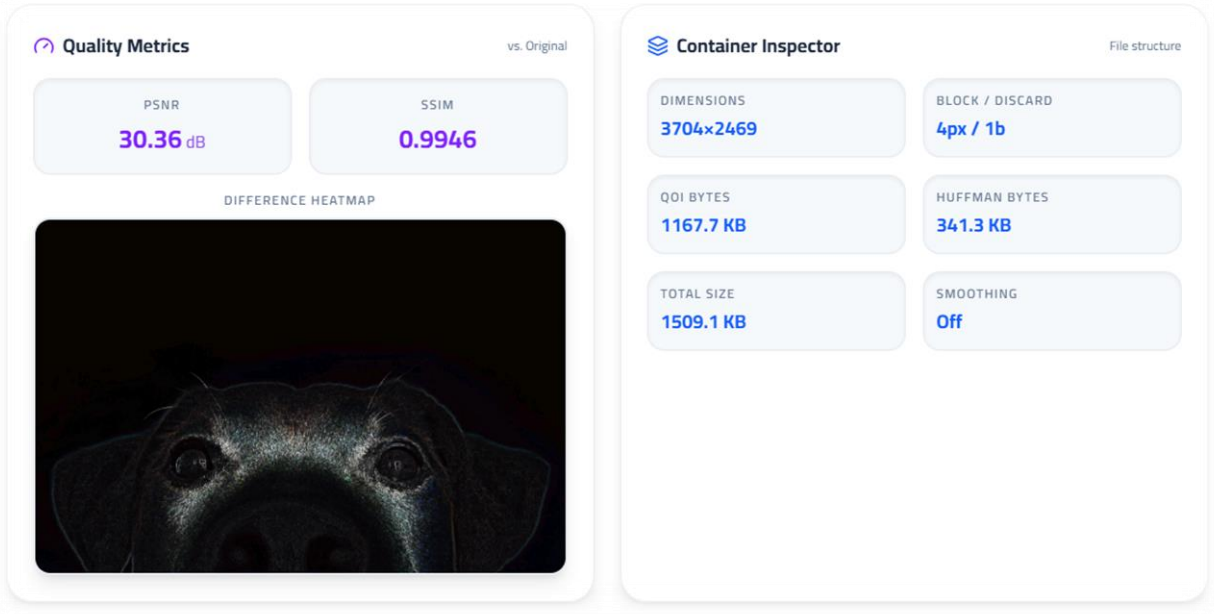


Рисунок 3.5 – Аналітичний блок «Deep Dive»: метрики PSNR/SSIM, теплова карта різниць та інспектор контейнера

Трохи нижче в цьому ж блоці виводиться «Difference Heatmap». Це така собі теплова карта, яка показує, де саме алгоритм найбільше «накосячив», тобто втратив найбільше дрібних деталей. А ось праворуч працює «Container inspector». Там можна подивитися на розміри кадру, налаштування, які ми обрали для блоку/квантування, і, що дійсно корисно, розбивку байтів контейнера. Це важливо. Причому розбивка йде окремо для QOI-частини, Huffman-частини і для службових заголовків.

Нижня частина сторінки – блок візуалізації та вихідних результатів «Visuals & Output». У верхній його частині знаходиться таблиця «YCrCb Color Space»: де показано процес розділення простору кольорів з RGB в YCrCb, нижче – «Codec timings»: для кожного формату показано час кодування та декодування. Це дозволяє бачити, де алгоритм програє чи виграє в швидкості (рис. 3.5). Нижче розташована основна зона візуального порівняння «до/після» – два вікна «Original» та «Compressed preview», між якими можна перемикатися або просто порівнювати поглядом (рис. 3.6).

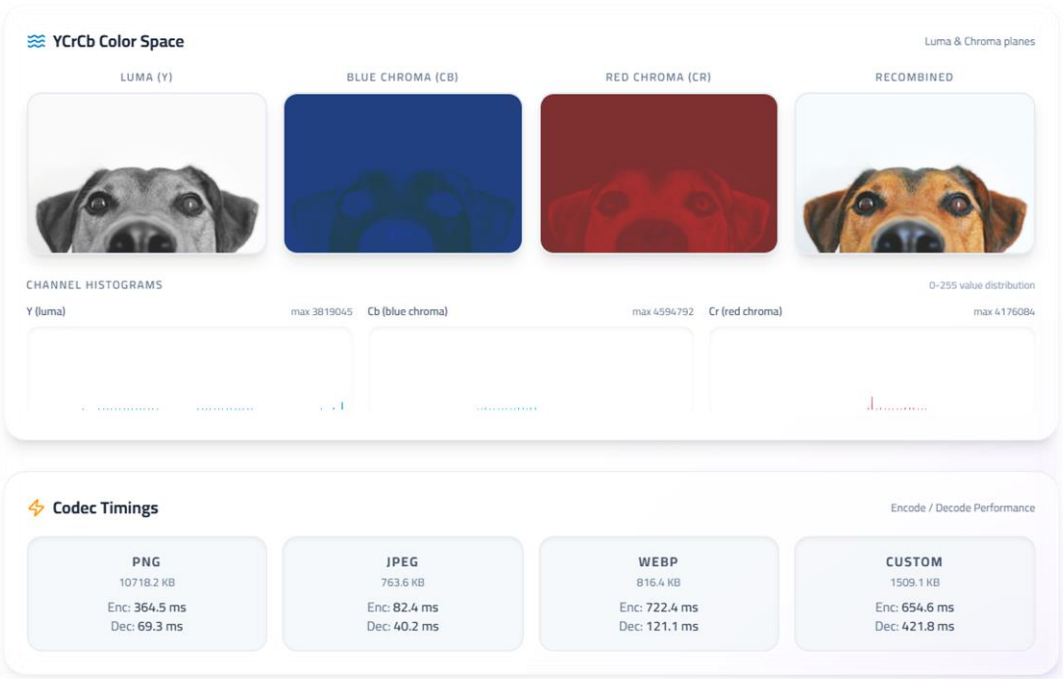


Рисунок 3.6 – Блок «Visuals & Output»: таблиці «YCrCb Color Space» та «Codec timings»

Нижче розташована основна зона візуального порівняння «до/після» – два вікна «Original» та «Compressed preview», між якими можна перемикатися або просто порівнювати поглядом.

Під ними – футер із кнопками завантаження: Download custom .kmr, Download preview PNG, Download JSON, а також кнопка Reset для очищення стану і завантаження нового зображення (рис. 3.7).

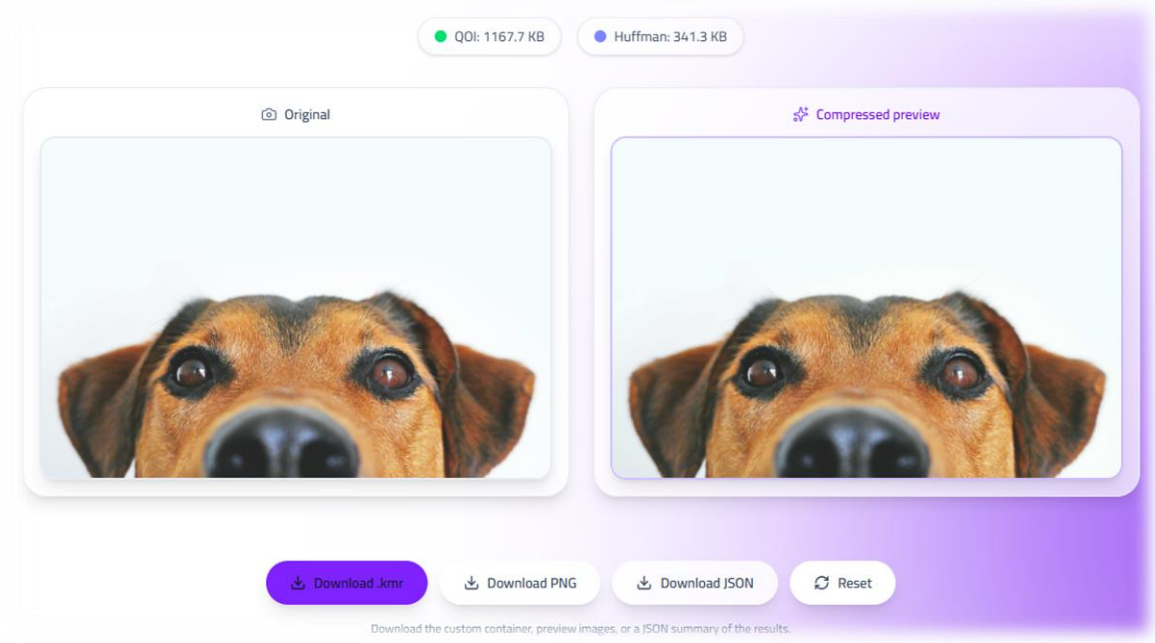


Рисунок 3.7 – Блок «Visuals & Output»: панель кнопок для експорту

Для зручності всю структуру інтерфейсу можна звести в одну компактну схему (рис. 3.8).

Таким чином, розроблена архітектура інтерфейсу забезпечує повний цикл дослідження ефективності алгоритму стиснення. Логічне розмежування функціональних зон дозволяє користувачеві не лише керувати параметрами кодування, а й миттєво оцінювати вплив змін на якість зображення та розмір файлу.

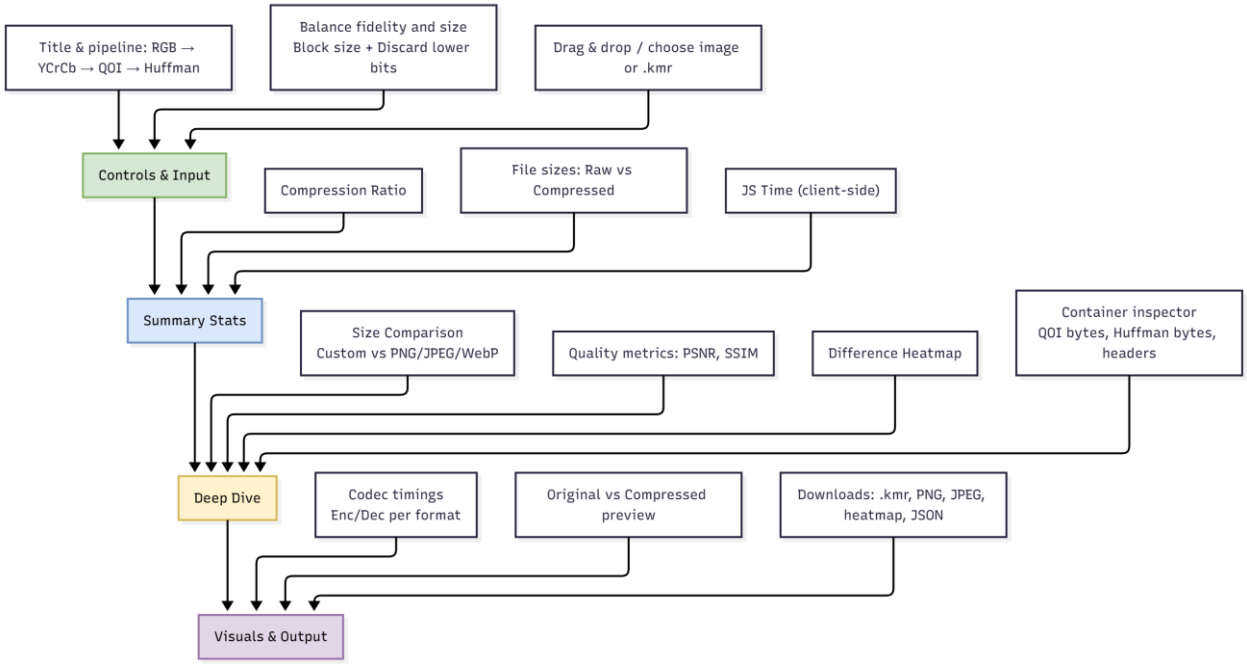


Рисунок 3.8 – Структурна схема дашборду

Реалізація такого підходу, як видно на практиці, закриває одразу кілька важливих задач. По-перше, вона робить весь процес перетворення даних набагато прозорішим. Це стосується буквально всіх етапів. По-друге, ми отримуємо інструменти для об’єктивного порівняння можна зіставити розроблений метод з наявними стандартами, причому в режимі реального часу. Крім того, з’являється можливість оперативно знаходити артефакти стиснення. Це легко робиться, якщо використовувати теплову карту відмінностей або інспектор блоків. Це допомагає нам швидко помічати, де саме алгоритм «промахнувся» тобто, де було втрачено найбільше деталей. Такі інструменти потрібні.

3.1.2 Клієнтський pipeline обробки зображення

Під цією зовні простою сторінкою ховається досить послідовний конвеєр обробки зображення. Після того як користувач перетягнув PNG/JPEG у зону «Drag and drop», зображення завантажується в Canvas і перетворюється

на масив пікселів формату RGBA. Далі вступає в дію алгоритм стиснення, який реалізовано в окремому модулі (рис. 3.9).

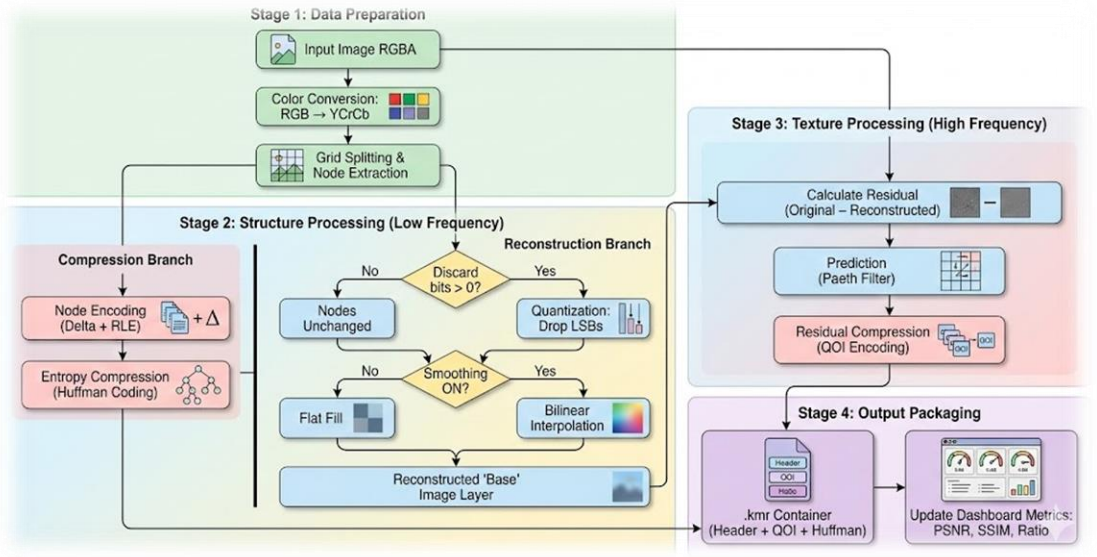


Рисунок 3.9 – Послідовність етапів обробки кадру в браузері: від RGBA-пікселів до контейнера .kmr та оновлення всіх блоків інтерфейсу

Загальна послідовність кроків виглядає так. Спершу зображення треба перевести з формату RGB у колірний простір YCrCb. Це робиться для того, щоб розділити яскравість (це Y) і, власне, кольорові компоненти (CrCb). Потім кадр ділиться на блоки розміром N×N (значення N користувач задає повзунком «Block size»).

Для кожного блоку обчислюється середнє значення Y, Cr, Cb – виходять так звані вузлові точки. Якщо Discard lower bits дорівнює нулю, вузли зберігаються без змін; якщо значення більше нульового, молодші біти кожної компоненти відкидаються, що зменшує точність, але збільшує стисливість. З вузлів будується наближене зображення: або кожен блок зафарбовується одним кольором вузла, або переходи згладжуються інтерполяцією.

Після цього обчислюється різниця (residual) між наближеним кадром та оригіналом. Для кожного пікселя значення передбачаються за трьома

сусідніми, а зберігається лише різниця. Таке різницеве зображення стискається енкодером QOI – це дає перший потік.

Масив вузлових точок проходить окрему обробку: дельта-кодування, RLE, Хаффман – і перетворюється на другий потік. На виході утворюється один контейнер, де є заголовок, QOI-частина і Huffman-частина. Усе це робиться в браузері, без сторонніх бібліотек типу WebAssembly – чистий JS/TS.

3.1.3 Логіка реакції на зміни параметрів

Уся ця обробка запускається щоразу, коли користувач змінює параметр або завантажує нове зображення. Компоненти React зберігають поточні значення blockSize, discardBits, прапорець smoothing і посилання на оригінальний кадр у state (рис. 3.10).

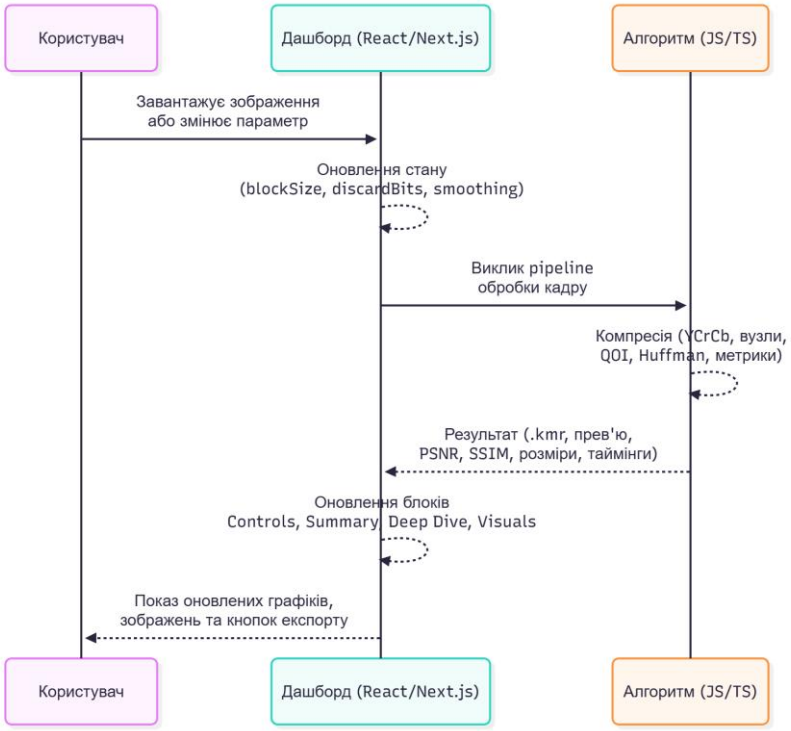


Рисунок 3.10 – Послідовність взаємодії між користувачем, інтерфейсом і модулем алгоритму при кожній зміні параметрів або завантаженні нового зображення

Зміна повзунка оновлює стан, після чого викликається функція перерахунку: якщо ввімкнено режим автооновлення, вона стартує одразу; якщо ні – очікує на натискання кнопки «Re-run» (цей режим зручний для дуже великих картинок). Цикл взаємодії добре видно на діаграмі послідовності:

У простішому варіанті обробка відбувається повністю на клієнті. Але є й альтернатива: застосунок може спробувати відправити дані в API-роут Next.js. Там, на Node.js, запуститься той самий pipeline, і він поверне результати. Якщо ж API чомусь недоступний, фронтенд просто переходить у «чистий» клієнтський режим.

У результаті вдалося створити не просто вебдемонстратор. Це, як на практиці виявилося, повноцінний робочий інструмент для аналізу. Для звичайного користувача він залишається зрозумілим: можна завантажити зображення, змінити параметри, одразу подивитися результат.

Але водночас інтерфейс дає достатньо технічних даних для подальших експериментів (що важливо). Це і метрики PSNR/SSIM, і пряме порівняння з PNG, JPEG, WebP, і деталізація структури даних у QOI та Huffman, а також часи кодування і декодування. Всі ці показники зібрані в одному середовищі, що дуже зручно.

Такий підхід фактично формує зручний фронтенд для наступних кроків: для апаратної реалізації на STM32F746 і для перевірки, як працюватиме потокова обробка на Raspberry Pi 4.

3.2 Реалізація апаратного модуля на основі STM32F746 Discovery

Для автономної реалізації кодека було обрано відлагоджувальну плату STM32F746G-DISCO на базі 32-бітного мікроконтролера STM32F746NG (ядро ARM Cortex-M7) (рис. 3.11). Ця плата зручна тим, що має все необхідне для обробки графіки: вбудований TFT-дисплей з роздільною здатністю 480×272, 16 Мбайт SDRAM для збереження кадрів, роз’єм для камери, Ethernet, слот microSD та багато інших периферій. Продуктивне ядро

Cortex-M7 на частоті 216 МГц оснащене кеш-пам’яттю інструкцій/даних і FPU, що пришвидшує роботу з обчисленнями з плаваючою комою.

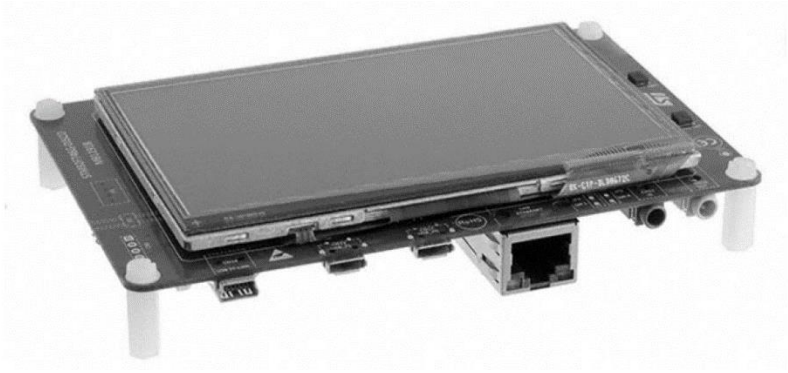


Рисунок 3.11 – Плата STM32F746G Discovery

Це дозволяє обробляти зображення у реальному часі навіть за обмежених ресурсів. У процесі розробки було використано середовище STM32CubeIDE (C/C++) та HAL-бібліотеки від STMicroelectronics для роботи з апаратними модулями плати. Низькорівневий доступ до периферії дав змогу оптимізувати ключові ділянки коду і використати можливості DMA, таймерів тощо для паралельних операцій. Таким чином, STM32F746 Discovery стала основою апаратного модуля компресора зображень.

3.2.1 Використані компоненти

На рис. 3.11 показано саму відлагоджувальну плату з екраном, на якій розгорнуто стенд. Головним елементом є мікроконтролер STM32F7, до нього під’єднано TFT-дисплей 4.3″ (480×272) для візуалізації кадрів (рис. 3.12). Було задіяно контролер SDRAM як буфер кадрів: фактично резервується два кадрові буфери по ~520 КБ, один для оригінального зображення, другий – для результату. Також ініціалізовано модуль DCMI для можливості захоплення відео з камери та Ethernet для потенційної передачі даних мережею.

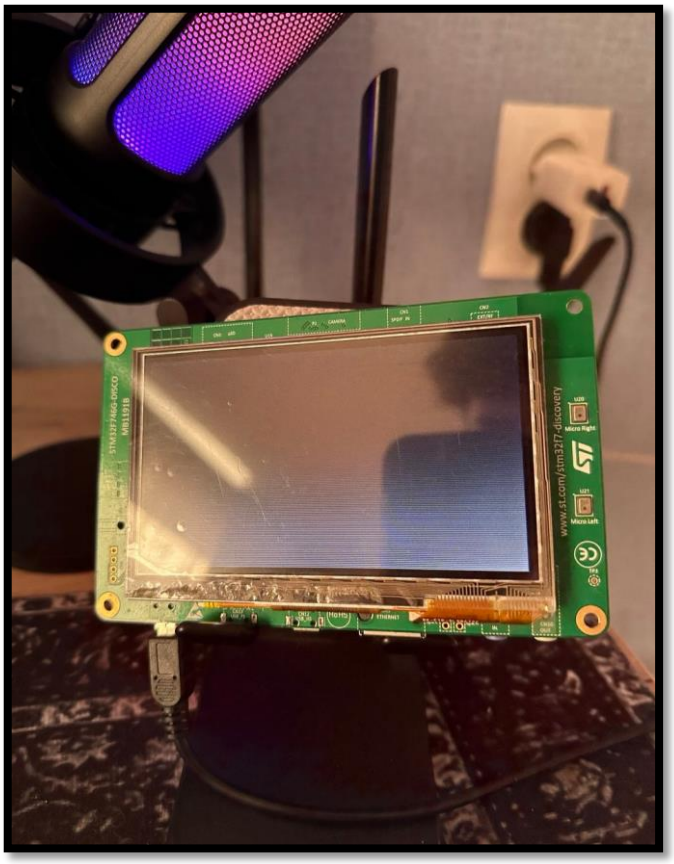


Рисунок 3.12 – Розгорнутий стенд з платою STM32F746 Discovery

На практиці в цьому прототипі для простоти було використано штучне тестове зображення замість камери, тому камера фізично не підключалась, однак код готовий отримувати кадри і з неї. Окрім цього, плата має UART-порт, який було використано для відладного виведення, та microSD, куди за потреби можна записувати результати компресії. Вся необхідна периферія (GPIO, таймери, DMA тощо) була налаштована через HAL.

Зокрема, DMA2D (Chrom-ART) можна було б використати для прискорення обробки зображень, але алгоритм специфічний, тому більшість роботи виконує CPU. Натомість DMA застосувано для швидкого перенесення готових даних на дисплей без навантаження ядра.

3.2.2 Генерація кадру

Щоб протестувати алгоритм без реальної камери, було програмно згенерувано кольоровий градієнт (рис. 3.12) та відображено його на екрані як вхідне зображення. Такий підхід дозволив отримати контрольований «кадр» з відомими характеристиками.

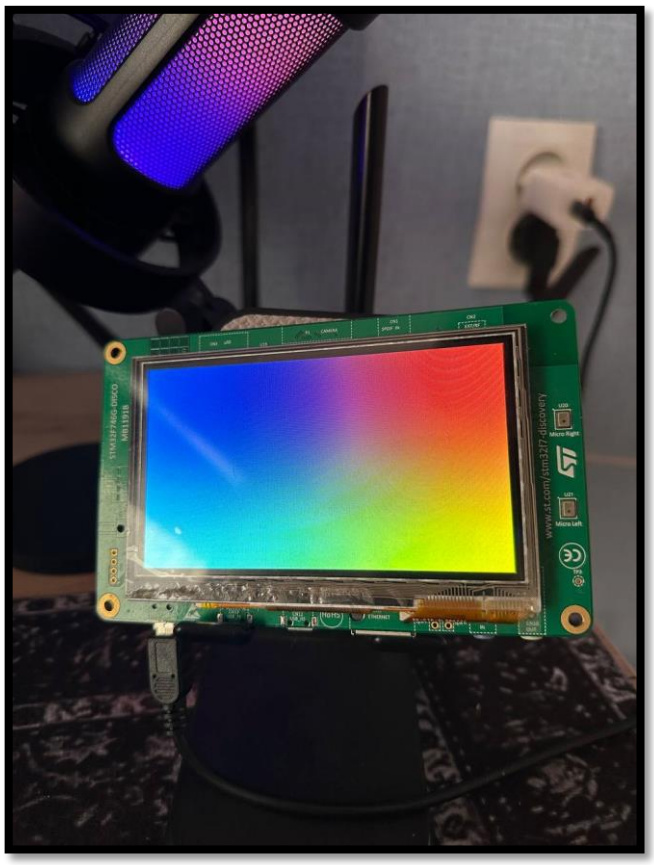


Рисунок 3.12 – Генероване градієнтове зображення на платі STM32F746

Градієнт формується у внутрішньому буфері: значення пікселів розраховуються в циклі по координатах x, y. Колір плавно змінюється від синюватого до червоного по горизонталі та від темного до яскраво-зеленого по вертикалі.

Видно, що верхній лівий кут має один відтінок, а нижній правий інший, отже було отримано різноманітну картинку для перевірки роботи енкодера. Після ініціалізації дисплея і запису градієнта в відеопам’ять, мікроконтролер

вмикає LCD-контролер на показ сформованого кадру. На екрані з’являється кольоровий градієнт, який слугує вхідними даними для подальшого стискання.

3.2.3 Алгоритм стиснення на STM32

Алгоритм стиснення, реалізований на STM32F746, повторює загальну логіку методу, описаного в попередніх розділах, тому детальну теорію тут недоцільно дублювати. Тому зроблено на практичній адаптації під ресурсні обмеження мікроконтролера та особливості виконання на платі STM32F746G-DISCO. Ідея прототипу така: плата самостійно формує тестове зображення, одразу запускає цикл компресії та показує результат і ключові показники на вбудованому дисплеї.

На рівні прошивки стиснення організовано як послідовний конвеєр. Вхідний кадр (згенерований градієнт або, в перспективі, кадр з камери) переводиться у YCrCb, після чого формується сітка вузлових значень за допомогою блочного усереднення з налаштовуваним розміром блока. Далі застосовується режим квантування через відкидання молодших біт (у тестових конфігураціях використовувалось невелике значення, щоб зберегти вигляд зображення).

Отримані вузли використовуються для реконструкції «скелетного» кадру, який і виводиться на екран як візуальне підтвердження коректності роботи методу. Такий підхід зручний для відладки, бо артефакти видно одразу, без проміжних інструментів на ПК (рис. 3.13).

```
539   chunks_len = size - (int)sizeof(qoi_padding);
540   for (px_pos = 0; px_pos < px_len; px_pos += channels) {
541     if (run > 0) {
542       run--;
543     }
544     else if (p < chunks_len) {
545       int b1 = bytes[p++];
546
547       if (b1 == QOI_OP_RGB) {
548         px.rgba.r = bytes[p++];
549         px.rgba.g = bytes[p++];
550         px.rgba.b = bytes[p++];
551       }
552       else if (b1 == QOI_OP_RGBA) {
553         px.rgba.r = bytes[p++];
554         px.rgba.g = bytes[p++];
555         px.rgba.b = bytes[p++];
556         px.rgba.a = bytes[p++];
557       }
558       else if ((b1 & QOI_MASK_2) == QOI_OP_INDEX) {
559         px = index[b1];
560       }
561       else if ((b1 & QOI_MASK_2) == QOI_OP_DIFF) {
562         px.rgba.r += ((b1 >> 4) & 0x03) - 2;
563         px.rgba.g += ((b1 >> 2) & 0x03) - 2;
564         px.rgba.b += ( b1       & 0x03) - 2;
565       }
566       else if ((b1 & QOI_MASK_2) == QOI_OP_LUMA) {
567         int b2 = bytes[p++];
568         int vg = (b1 & 0x3f) - 32;
569         px.rgba.r += vg - 8 + ((b2 >> 4) & 0x0f);
570         px.rgba.g += vg;
571         px.rgba.b += vg - 8 +  (b2       & 0x0f);
572       }
573       else if ((b1 & QOI_MASK_2) == QOI_OP_RUN) {
574         run = (b1 & 0x3f);
575       }
```

Рисунок 3.13 – Приклад коду QOI, написаному на мові C

Після побудови наближеного кадру формується різницева складова між оригіналом і реконструкцією. На платформі STM32 цей етап реалізовано у більш простому й стабільному вигляді, щоб не створювати зайве навантаження на процесор. Далі різницева частина стискається швидким растровим енкодером на логіці QOI, а потоки вузлових даних додатково ущільнюються легкими ентропійними прийомами, зокрема дельта-перетворенням і кодуванням Хаффмана.

У підсумку формується компактний набір даних. За своєю ідеєю він відповідає структурі контейнера з вебдемонстратора, але в цьому випадку рішення адаптоване під обмеження embedded-платформи та орієнтоване на подальше збереження або передачу даних.

Важливо, що в умовах STM32 основним обмеженням є не стільки кількість операцій, скільки пам’ять і передбачуваність часу виконання. Тому в реалізації зроблено ставку на прості структури даних, мінімальні проміжні буфери і відмову від важких пост-обробок. Розміщення кадрів у SDRAM дає змогу тримати необхідні буфери без надлишкового копіювання, а використання вбудованих таймерів дозволяє коректно вимірювати час повного циклу обробки. На екран виводяться ключові показники: тривалість обробки кадру та орієнтовний коефіцієнт стиснення. Це робить прототип самодостатнім: плата демонструє не лише факт роботи алгоритму, а й його практичну доцільність.

3.2.4 Вивід результатів

Після завершення стискання програма підраховує базову статистику та відображає її поверх зображення у вигляді текстового оверлею (рис. 3.14). Час обробки кадру вимірювався стандартним способом: фіксувалась мітка часу функцією HAL_GetTick() безпосередньо перед запуском компресії та одразу після неї; різниця між значеннями визначала тривалість обчислень. Для тестового градієнта з роздільністю 480 × 272 при налаштуваннях block = 8 і discardBits = 2 отримано час обробки близько 288 мс, що відповідає приблизно 3–4 кадрам/с.

Рисунок 3.14 – Результат роботи алгоритму на дисплеї STM32F746G з відображенням часу обробки та оцінки коефіцієнта стиснення.

Звісно, такий результат ми не можемо вважати цільовим для повноцінного режиму реального часу. Але для прототипу це достатньо показово. Це підтверджує, що конвеєр на рівні мікроконтролера є працездатним і, головне, передбачуваним.

Також це дає нам реальну базу для подальших оптимізацій. Наприклад, можна було б скоротити кількість проміжних копій, або ж активніше використовувати DMA та кеш. Або спробувати ще більше спростити обсяг обчислень у вузловій частині. Є куди рухатися.

Окремо було оцінено ступінь стиснення. На даному етапі повний запис результату (потоки QOI + Huffman) на SD-карту не використовувався як основний критерій демонстрації, тому виводилась орієнтовна оцінка коефіцієнта стиснення на основі співвідношення вихідного обсягу даних (ширина × висота × 3 байти для RGB) до сумарного обсягу сформованих

стислих компонентів. Для конфігурації discardBits=2 отримано значення близько 6:1, що і відображено в інтерфейсі як «Compression: 6.00x». На рис. 3.13 наведено приклад такого оверлею, де одночасно показано час обробки кадру та оцінку коефіцієнта стиснення.

3.3 Впровадження алгоритму стиску зображень на Raspberry Pi 4

Реалізувати алгоритм на Raspberry Pi виявилося окремою маленькою історією – не стільки про кодування, скільки про сумісність камер, черги кадрів і про пошук тієї конфігурації, яка просто працює. Після кількох спроб із різними дистрибутивами стало очевидно, що найстабільніше оточення для камер Raspberry Pi формується навколо libcamera. Тому в підсумку було обрано конфігурацію, де Python бере на себе функцію захоплення сирого відеопотоку, а вся логіка стиску передається в Node.js – у той самий модуль, який раніше був використаний для обробки зображень на звичайному ПК.

3.3.1 Попередня структура рішення

На практиці система працює за принципом «двох світів»: Python виконує роль низькорівневого захоплення кадрів у форматі RGBA, тоді як Node.js – це більш гнучкий простір для експериментів із самим алгоритмом стиску. Така комбінація виявилась не просто зручною, а й необхідною: Pi 4 не надто тягне складну математику в чистому Python у реальному часі, але чудово стрімить буфери в stdout. Нижче схематично зображено логіку руху кадру в системі (рис. 3.15):

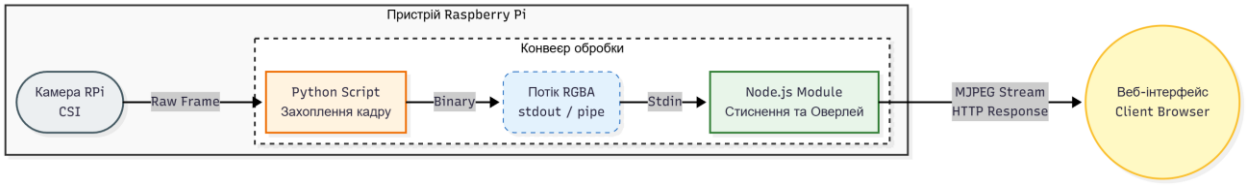


Рисунок 3.15 – Структура передавання та обробки відеопотоку

На цьому рівні все виглядає просто, але далі починається найцікавіше стискання блочно-вузловим методом, адаптованим до реального відеопотоку.

3.3.2 Захопленян кадру у Python

Потік кожного кадру конвертується у RGBA й надсилається просто в stdout. Node.js читає цей потік без зайвих накладних витрат. Цей підхід дозволяє досягти кадру завширшки 640×480 із частотою приблизно 20–25 FPS.

```
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, WIDTH)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, HEIGHT)
cap.set(cv2.CAP_PROP_FPS, 30)
```

Рисунок 3.16 – Лістинг коду захоплення зображення

З боку системи цей блок (рис. 3.16) працює доволі стабільно, єдине, що інколи доводиться коригувати черги буферів, але для дипломного проєкту поведінка виявилась більш ніж достатньою.

3.3.3 Інтеграція алгоритму стиску у Node.js

Наступний фрагмент – це місце, де налаштовується конфіг стиснення. Тут він працює вже не з окремими файлами, а з живими кадрами, що летять потоком (рис. 3.17).

```
const WIDTH = 640;
const HEIGHT = 480;
const FRAME_SIZE = WIDTH * HEIGHT * 4;
const ORIGINAL_SIZE = WIDTH * HEIGHT * 3;

let codecConfig = {
    blockSize: 4,
    discardBits: 2,
    smooth: false
};
```

Рисунок 3.17 – Лістинг коду з налаштуванням рівня стиснення

Головними параметрами є blockSize (розмір блоків), discardBits (відкидання найменш значущих біт) та smooth (білінійна інтерполяція). Як показала практика, максимальне значення для «прийнятної» картинки має рівнятися 8x8 blockSize, і discardBits максимум 2, саме тоді рівень стиснення найбільший, але й при тому найбільші втрати.

3.3.4 Підрахунок статистики стиску в реальному часі

Щоб оцінка роботи алгоритму була не абстрактною, а видимою тут і зараз, до кожного кадру накладається прозора плашка зі статистикою. Вона генерується за допомогою SVG і вставляється поверх кадру вже під час JPEG-кодування в Sharp (рис. 3.18).

```
const statsSvg = `
<svg width="${WIDTH}" height="80">
    <text ...>Raw: ${rawKB} KB -> Comp: ${compKB} KB</text>
    <text ...>Ratio: ${ratioStr}x | Time: ${timeMs}ms</text>
</svg>
`;
```

Рисунок 3.18 – Прозора плашка статистики

Така статистика виявилася дуже корисною при дослідженні поведінки алгоритму на рухомих сценах. Також видно, наскільки сильно змінюється коефіцієнт стиску при навіть невеликому русі.

3.3.5 Результати роботи алгоритму на Raspberry Pi 4

Для аналізу якості було підібрано кілька конфігурацій параметрів: Варіант 1: 8 × 8 блоки, відкидання 2 біти (найвищий коефіцієнт стиску – 7.74x) (рис. 3.19); варіант 2: 4 × 4 блоки, відкидання 2 біти (середній коефіцієнт – 2.49x) (рис. 3.20); варіант 3: 2 × 2 блоки, відкидання 1 біт (найвища якість, коефіцієнт ~2.14x, фактично без втрат текстури) (рис. 3.21).

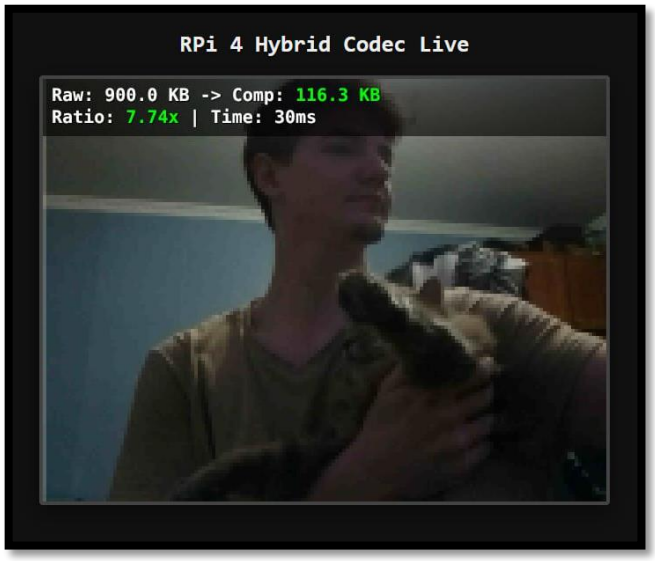


Рисунок 3.19 – Відеокадр зі стиском 7.74x (параметри 8 × 8, 2 біти)

Тут добре видно, що структура з великим розміром блоку поводиться майже як QOI у режимах високої агресії маскується дрібний шум, а загальна кольорова «піксельність» робить кадр дуже компактним.

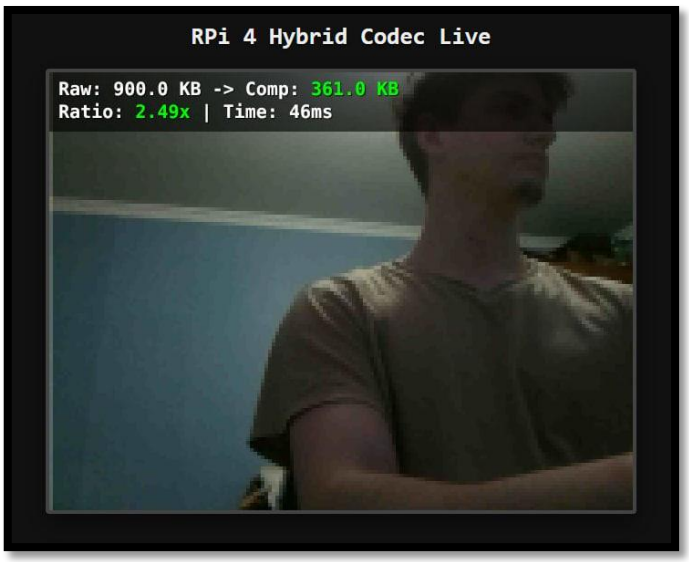


Рисунок 3.20 – Відеокадр зі стиском 2.49x (параметри 4 × 4, 2 біти)

В цьому випадку компресія більш помірна. Якість уже близька до стандартних MJPEG-кодерів на низьких профілях, але деталі в темних ділянках сцени все ще частково зникають.

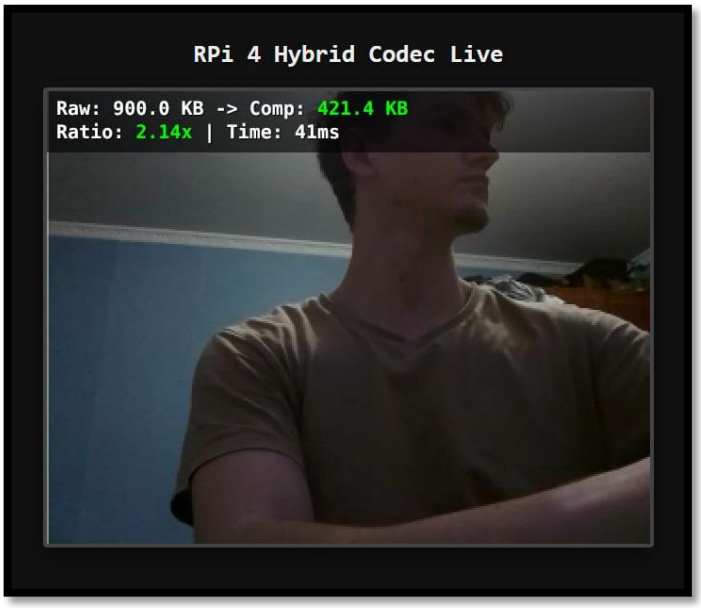


Рисунок 3.21 – Високоякісний кадр (2.14x, параметри 2×2, 1 біт)

Візуально це майже повністю відповідає оригіналу. На практиці саме цей режим виявився найкращим для «живого» відео – помірна компресія, але плавна передача дрібних деталей.

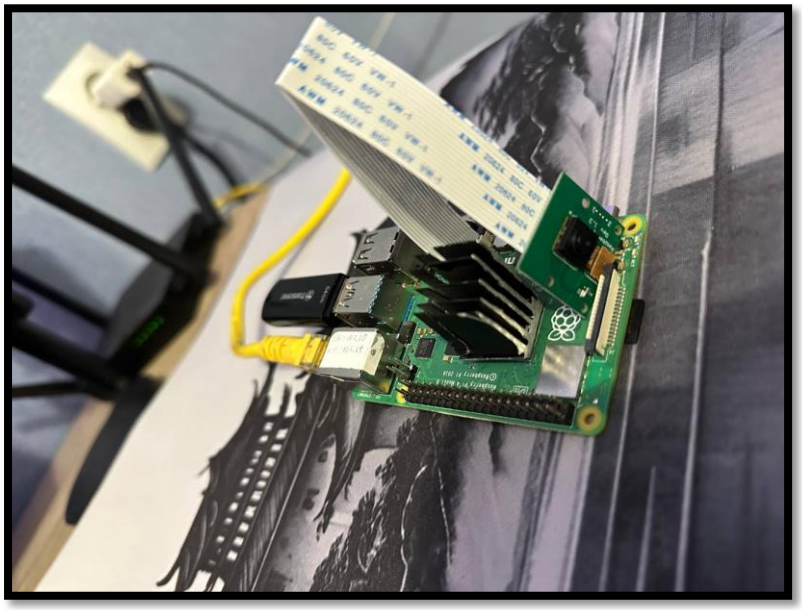


Рисунок 3.22– Схема підключення Raspberry Pi 4 з камерою

На фото видно стандартне підключення Raspberry Pi Camera Module через CSI-шлейф, а також USB-накопичувач, який використовувався для розгортання робочого середовища та логів (рис. 3.22).

Висновок до розділу 3

У третьому розділі була виконана повна реалізація гібридного алгоритму стиску зображень і його перевірка на трьох різних платформах. Кожен етап мав свій характер і свої «вузькі місця», тому результати вийшли доволі показові.

Реалізація на Next.js дала можливість досить швидко перевірити логіку самого алгоритму. Йдеться про обчислення вузлових значень, роботу з квантованими компонентами YCbCr, реконструкцію блоками та вплив згладжування. На цьому етапі було зручно бачити кожен проміжний крок. І як змінюється зображення, і як поводиться коефіцієнт стиснення при зміні розміру блоку чи кількості відкинутих біт. Фронтенд фактично став тестовим середовищем, де параметри можна було варіювати й підганяти під потрібний результат.

Перехід на STM32F746G показав іншу сторону – коли обчислення мають ужитися в мікроконтролер з обмеженою пам’яттю й обмеженим доступом до буферів. Робота з DMA, SDRAM і простими графічними структурами дозволила оцінити, наскільки алгоритм взагалі придатний для таких пристроїв. Тут довелося відмовлятися від складних операцій і орієнтуватися тільки на найпростіші варіанти обробки блоків, адже навіть базові операції кольорового перетворення можуть стати «вузьким місцем». Це допомогло зрозуміти межу, нижче якої алгоритм уже не можна спростити без втрати сенсу.

Найважливіша частина роботи – експеримент із Raspberry Pi 4, де вдалося реалізувати стиснення у режимі реального часу. Саме тут алгоритм отримав «бойові» умови: камера, потік кадрів, Python-захоплення, передача в Node.js і MJPEG-стрім назад у браузер. На цьому етапі добре видно, як параметри впливають на практичний результат. Наприклад, можна навести такі результати:

1) якщо взяти блок 8 × 8 і відкинути 2 біти, коефіцієнт стиснення вийшов високий 7.74x. Але якість, звісно, вже помітно падала;

2) налаштування 4 × 4 при тих самих 2 бітах давало співвідношення десь 2.49x. Це виглядає як найбільш збалансований варіант;

3) конфігурація 2 × 2 з відкиданням 1 біта забезпечила близько 2.14x. Зате зображення майже не відрізнялося від оригіналу. Тому для live-відео, як на практиці, це виглядає найкращим режимом;

4) сам режим без відкидання бітів показав, що алгоритм, у принципі, може працювати і як lossless-рішення, хоча швидкість роботи в такому випадку, зрозуміло, зменшується.

Загалом, така картина підтверджує, що алгоритм є досить гнучким. Він може бути швидким кодером, наприклад, для потокових сценаріїв. Але також легко переводиться в більш «обережний» режим, коли пріоритетом є саме якість, а не швидкість стиснення.

Окремо треба згадати про візуалізацію. Додавання метрик у реальному часі (Raw, Comp, Ratio, час обробки кадру) дуже допомогло. Це дозволило оцінити, як алгоритм поводиться саме в потоковому режимі, а не просто на статичних зображеннях. Це дало більш практичне уявлення про те, які параметри варто вважати оптимальними.

У результаті практична частина показала, що запропонований алгоритм може працювати на різних апаратних платформах. Проте вимоги до ресурсів і швидкості змушують підбирати параметри під конкретне середовище. Raspberry Pi виявився найбільш зручним варіантом для прикладної перевірки. Саме на ньому вдалося отримати стабільне відео зі стисненням, яке можна оцінювати і за метриками, і візуально.

Таким чином, методика стиснення не обмежується формальною реалізацією. Вона демонструє працездатність у реальних умовах, що дає підстави розглядати алгоритм як придатний для подальших оптимізацій і для інтеграції у системи з обмеженими ресурсами та високими вимогами до швидкодії.

4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Метою даного розділу є комплексна верифікація та валідація розробленого апаратно-програмного комплексу. Експериментальні дослідження спрямовані на визначення реальної ефективності запропонованого комбінованого методу стиснення (YCrCb, QOI та Huffman), оцінку його швидкодії на різних обчислювальних платформах та виявлення меж його застосування.

На етапі тестування основним було перевірити гіпотезу, що запропонований гібридний підхід може дати коефіцієнт стиснення, близький до сучасних стандартів на кшталт JPEG і WebP, але з помітно меншими обчислювальними витратами. Це важливо саме з практичного боку, бо тоді метод можна розглядати як придатний і для веб-середовища, і для вбудованих систем, де ресурси обмежені.

4.1 Методика проведення експерименту та критерії оцінки

Щоб результати дослідження були об’єктивними й їх можна було повторити, було зібрано тестовий стенд із трьома рівнями обчислювальної потужності. Він охоплює різні сценарії використання. Від продуктивної робочої станції до вбудованих систем із обмеженими ресурсами.

4.1.1 Характеристика апаратного забезпечення

Тестування проводилося на наступних конфігураціях:

1) високопродуктивна станція (PC): центральний процесор: AMD Ryzen 5 5600X (6 ядер, 12 потоків, частота до 4,6 ГГц); оперативна пам’ять: 32 Гбайт DDR4 3200 МГц; середовищем виконання було Google Chrome v120+ (V8 Engine) для клієнтських тестів та Node.js v18 для пакетної обробки; основне призначення – оцінка базової складності алгоритму, пакетна обробка великих масивів даних, аналіз пікового навантаження на CPU;

2) платформа Інтернету речей (IoT): одноплатний комп’ютер: Raspberry Pi 4 Model B (4 Гбайт RAM); камера: Raspberry Pi Camera Module v2; основне призначення – перевірка роботи в режимі реального часу (Real-time), тестування потокового відео;

3) вбудована система (Embedded): мікроконтролер: STM32F746NG (ARM Cortex-M7, 216 МГц); пам’ять: 16 Мбайт SDRAM (зовнішня); основне призначення – перевірка життєздатності алгоритму в умовах жорстких ресурсних обмежень.

4.1.2 Характеристика тестової вибірки даних

Для проведення статистичного аналізу нам довелося зібрати датасет. Він складається з 300 зображень і включає три різні групи контенту, щоб покрити якомога більше сценаріїв використання.

1) група А (фотореалізм): Це 100 фотографій (природа, архітектура, люди). Тут висока деталізація і, звісно, природний розподіл градієнтів.

2) група Б (технічна графіка): Це теж 100 зображень. Але тут навпаки схеми, креслення, скріншоти інтерфейсів. Тобто все, що має різкі переходи кольорів.

3) група В (змішаний контент): І ще 100 зображень, але вже різної роздільної здатності (від 640 × 480 до 4K). Ці дані було взято з вебкамери та різних синтетичних тестів.

Таке різноманіття було потрібне, щоб отримати максимально об’єктивні результати.

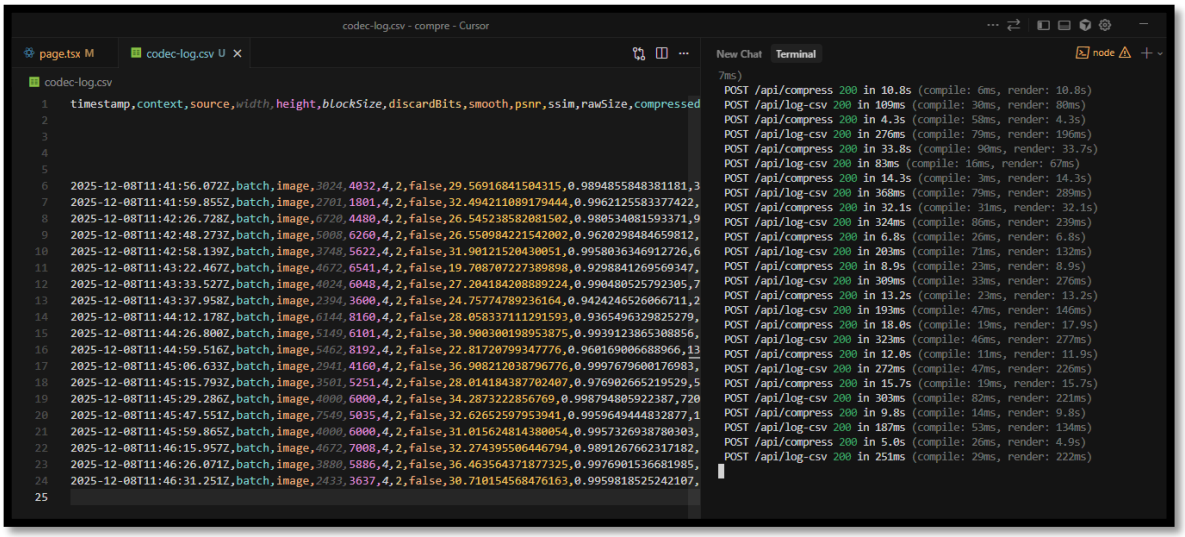


Рисунок 4.1 – Скріншот процесу збору логів

Збір метрик здійснювався в автоматичному режимі за допомогою розробленого модуля логування, інтегрованого у вебзастосунок (рис. 4.1).

4.2 Глибинний статистичний аналіз ефективності алгоритму

У ході експерименту досліджувались три базові конфігурації алгоритму, які представляють різні сценарії компромісу між якістю та розміром файлу:

– «висока якість»: розмір блоку 4 × 4, відкидання 2 біт;

– «збалансований режим»: розмір блоку 6 × 6, відкидання 1 біта;

– «максимальне стиснення: розмір блоку 8 × 8, відкидання 2 біт.

4.2.1 Аналіз коефіцієнта стиснення (Compression Ratio)

Результати пакетної обробки (рис. 4.2) демонструють суттєву залежність ступеня стиснення від розміру блоку вузлових точок.

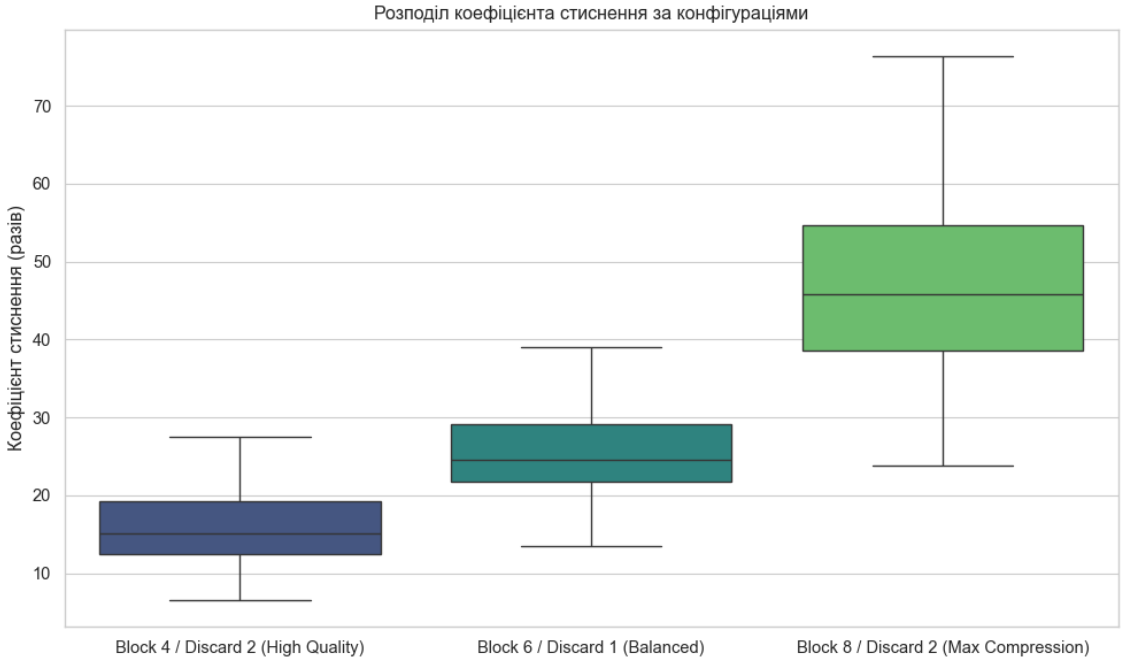


Рисунок 4.2 – Розподіл коефіцієнта стиснення за конфігураціями

Для конфігурації Block 8 / Discard 2 середній коефіцієнт стиснення досягає 48.7x (медіанне значення). Це означає, що вихідне зображення обсягом 5 Мбайт стискається до ~100 кбайт. Такий результат є критично важливим для передачі даних через низькошвидкісні канали зв’язку (LoRaWAN, GPRS).

У режимі Block 4 / Discard 2, орієнтованому на збереження деталей, середній коефіцієнт становить 17.3x. Хоча це значення менше за максимальне, воно все одно у 5–7 разів перевищує показники алгоритмів без втрат (PNG).

4.2.2 Аналіз якості відносно ступеня стиснення (Trade-off)

Одним із найважливіших результатів роботи є побудова та аналіз діаграми розсіювання, що відображає компроміс між ступенем стиснення та якістю відтворення (рис. 4.3).

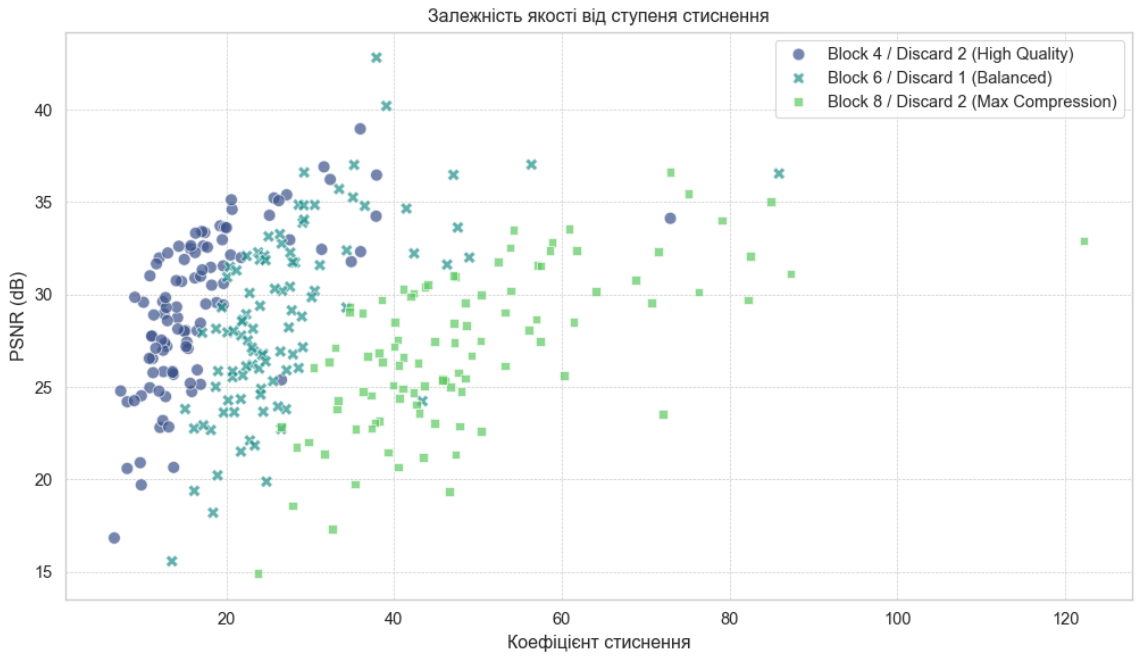


Рисунок 4.3 – Залежність якості (PSNR) від ступеня стиснення

На графіку досить добре видно формування так званого фронту Парето. Якщо подивитися на хмару точок уважніше, помітно така закономірність. Коли коефіцієнт стиснення зростає приблизно з 10x до 30x (перехід від Block 4 до Block 6), значення PSNR падає без різких провалів і залишається в межах прийнятної якості 28–32 дБ. Це виглядає як ознака того, що алгоритм у цьому діапазоні прибирає надлишковість відносно акуратно і не «ламає» базову структуру зображення.

При надвисоких ступенях стиснення (>50x, режим Block 8) розкид значень PSNR збільшується. Це пояснюється тим, що для складних зображень з великою кількістю дрібних деталей (висока ентропія) грубе блокування призводить до помітніших помилок апроксимації.

Також точки на графіку групуються за кольором, тобто за конфігураціями. Це важливо з практичного боку. Робота алгоритму виглядає передбачуваною: потрібний рівень стиснення можна отримати, просто обравши відповідний розмір блоку. Без несподіваних стрибків у результаті і без ризику, що однакова настройка дасть зовсім різну картинку.

4.2.3 Аналіз якості відтворення (PSNR та SSIM)

Оцінка якості відновлених зображень проводилась за допомогою об’єктивних метрик PSNR (Peak Signal-to-Noise Ratio) та SSIM (Structural Similarity Index) (рис. 4.4).

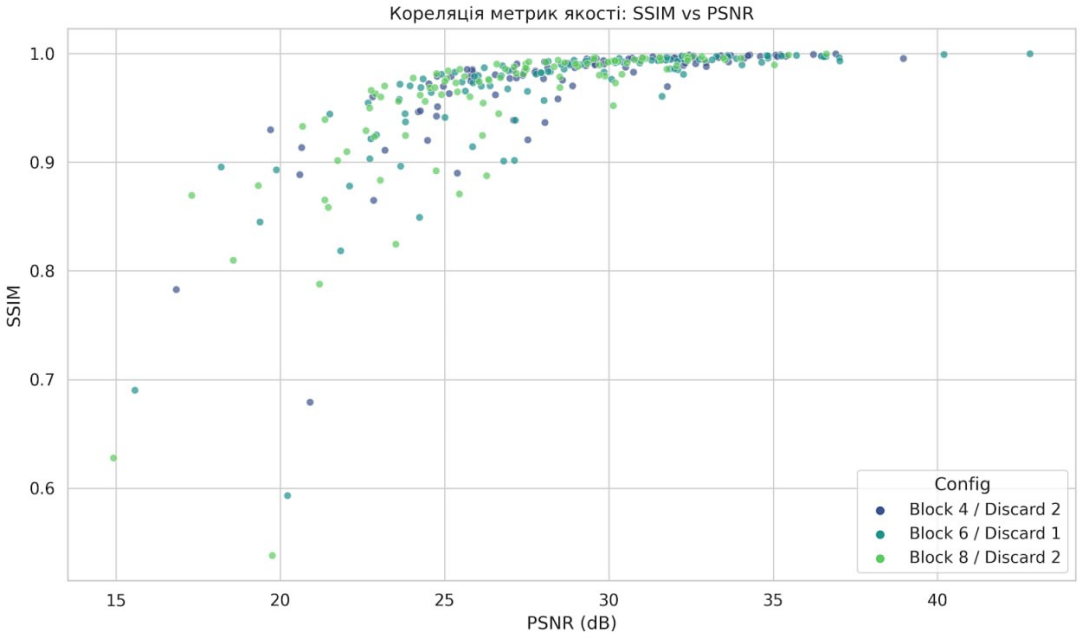


Рисунок 4.4 – Кореляція метрик якості: SSIM vs PSNR

Аналіз діаграми розсіювання (рис. 4.4) дозволяє зробити наступні висновки:

Навіть при агресивному стисненні (Block 8) метрика структурної подібності SSIM залишається на рівні >0.95. Це свідчить про те, що розроблений метод ефективно зберігає основні структурні елементи сцени, «жертвуючи» лише високочастотним шумом та мікротекстурою, що є прийнятним для задач відеоспостереження.

Середнє значення PSNR для режиму 4 × 4 становить 29.3 дБ, що наближається до порогу візуальної прозорості (30 дБ). Зі збільшенням розміру блоку PSNR прогнозовано знижується, однак не падає нижче критичних значень, що призвели б до повної деградації зображення.

4.2.4 Аналіз ефективності відносно стандарту JPEG

Для оцінки практичної цінності методу було розраховано відносну економію дискового простору порівняно зі стандартом JPEG (Quality 90) за формулою:

$$E_{rel} = \frac{Size_{JPEG} - Size_{Custom}}{Size_{JPEG}} \times 100\,\%\,. \qquad (4.1)$$

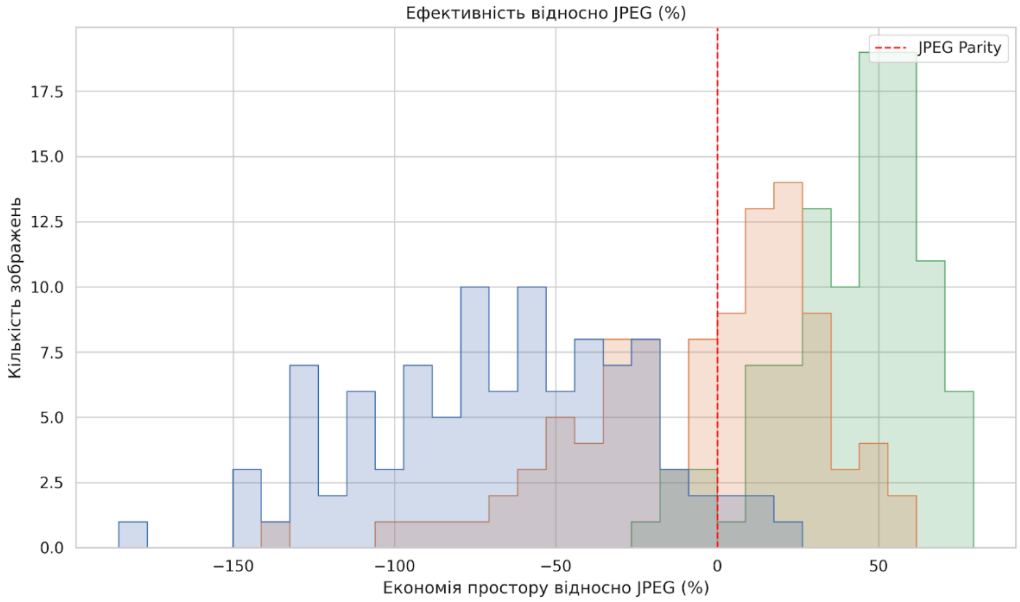


Рисунок 4.5 – Ефективність відносно JPEG

Гістограма розподілу демонструє чітку сегментацію ефективності залежно від обраної конфігурації (рис. 4.5).

1) у режимі «Максимальне стиснення» алгоритм показує позитивну ефективність у 98 % випадків, забезпечуючи економію простору від 20 % до 60 % порівняно з JPEG.

2) у режимі «Висока якість» спостерігається паритет або незначне відставання (–10 %...0 %) на зображеннях з високою ентропією (фотографії лісу, трави).

Це дозволяє чітко позиціювати розробку: метод є ефективнішим за JPEG для зображень з великими однорідними областями та градієнтами, поступаючись йому на високочастотних текстурах.

4.3 Порівняльний аналіз із промисловими стандартами

Для визначення місця розробленого методу серед існуючих рішень було проведено пряме порівняння з форматами PNG, JPEG та WebP на ідентичному наборі даних (рис. 4.6).

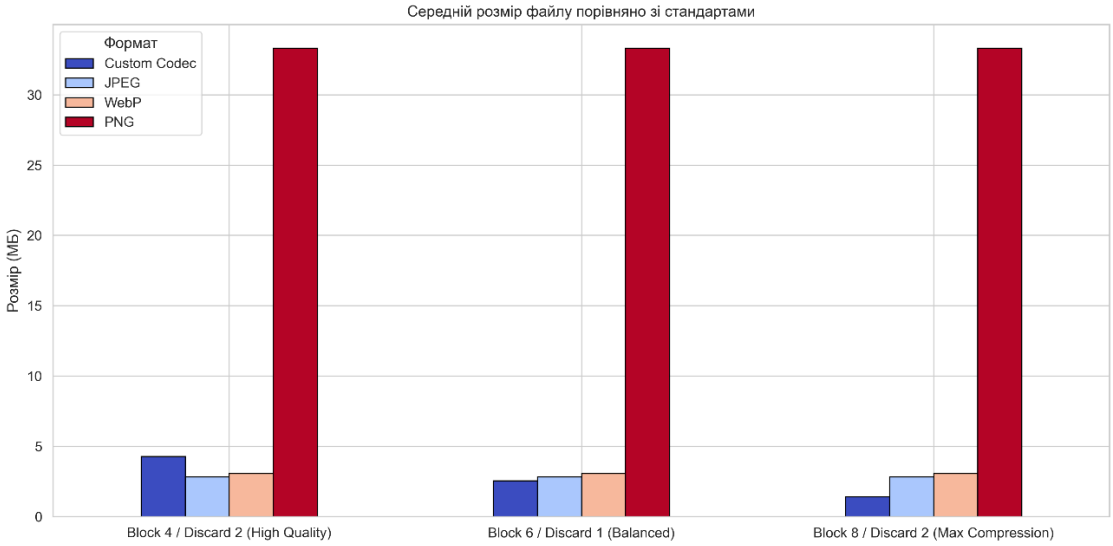


Рисунок 4.6 – Середній розмір файлу порівняно зі стандартами

Аналіз гістограми середніх розмірів файлів (рис. 4.4) дає досить однозначну картину. Якщо порівнювати з PNG, то розроблений кодек показує виграш у 6–15 разів залежно від обраної конфігурації. Для задачі з проміжними кадрами це виглядає дуже переконливо, бо lossless-формати тут явно програють саме за обсягом, а зайві мегабайти в потоці швидко перетворюються на проблему.

З JPEG ситуація більш цікава і не така «чорно-біла». У режимі максимального стиснення (Block 8) метод у середньому формує файли на 20–40 % менші за JPEG із Quality 90. Це вже серйозна різниця. У режимі високої якості (Block 4) розміри виходять близькими, тобто десь на рівні. Але тут проявляється інша перевага. «Custom Codec» простіший у декодуванні на слабких процесорах, бо не використовує складні перетворення на кшталт DCT.

Як на практиці, саме це часто й вирішує, чи буде метод реально корисним на обмеженому залізі.

Відносно WebP: хоча WebP демонструє дещо кращу щільність кодування, розроблений метод значно випереджає його за швидкістю кодування (див. п. 4.4), що є вирішальним фактором для систем реального часу.

4.4 Дослідження швидкодії та ресурсоємності

Ефективність використання ресурсів є критичним параметром для вбудованих систем. Аналіз стабільності роботи алгоритму було проведено шляхом побудови гістограми розподілу часу кодування (рис. 4.7).

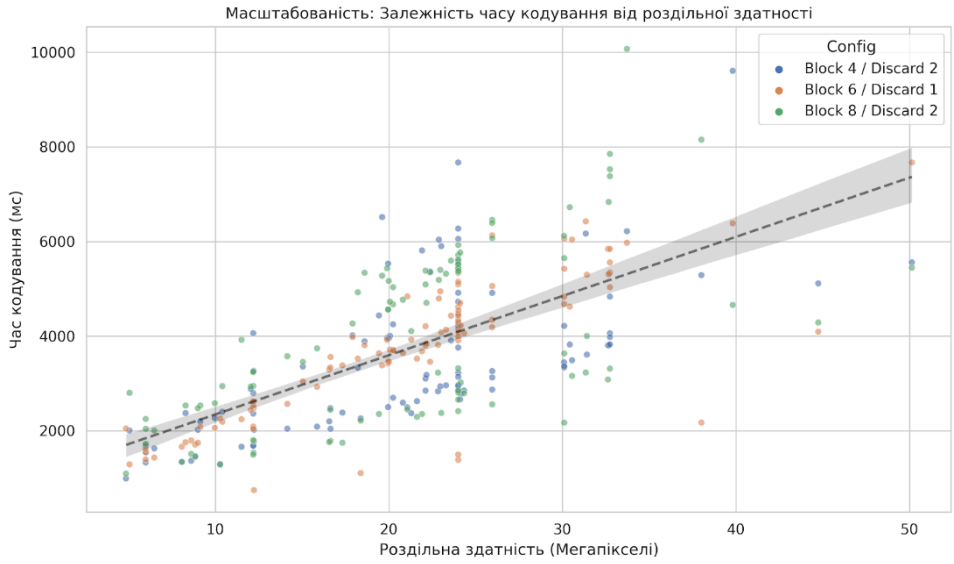


Рисунок 4.7 – Гістограма розподілу часу кодування

Графік показує, що час обробки має близький до нормального розподіл із невеликим стандартним відхиленням. Це виглядає як ознака достатньо детермінованої поведінки алгоритму. Час виконання тут здебільшого визначається кількістю пікселів ($O(N)$) і лише незначно змінюється залежно від змісту кадру. Тобто основний фактор навантаження очевидний, без «сюрпризів» на рівні контенту.

Якщо ж порівнювати підхід із методами, що базуються на складному ентропійному кодуванні (наприклад, як у WebP чи PNG), то ситуація зовсім інша.

Там, як на практиці виявляється, час може відчутно зростати. Особливо на «шумних» зображеннях, і це дуже помітно в реальних сценаріях. Запропонований підхід, навпаки, дає більш передбачувану затримку (Latency).

І для систем реального часу – особливо якщо ми говоримо про мікроконтролери чи одноплатні комп’ютери – така стабільність часу відгуку важлива. Майже так само, як і сам коефіцієнт стиснення, а іноді й більше.

4.4.1 Асиметрія кодування та декодування (нормовано на МП)

Для нівелювання впливу різної роздільної здатності зображень дані було нормовано до 1 мегапікселя (мс/МП) (рис. 4.8).

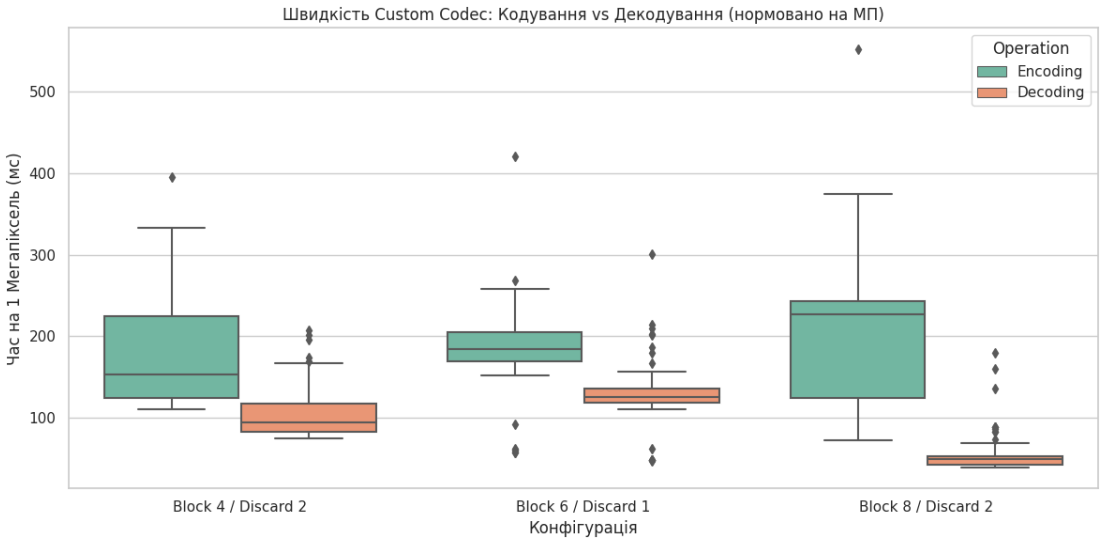


Рисунок 4.8 – Швидкість Custom Codec: Кодування проти Декодування

Результати вимірювань демонструють виражену асиметрію алгоритму, особливо в режимі Block 8 / Discard 2:

– час кодування: ~203 мс/МП;

– час декодування: ~54 мс/МП.

Декодування відбувається майже в 4 рази швидше за кодування. Це зумовлено тим, що найважча операція усереднення пікселів блоку та розрахунок різниці виконується лише на етапі запису. Декодеру ж потрібно лише відновити значення з вузлових точок (кількість яких у 64 рази менша за кількість пікселів при блоці 8 × 8) та накласти різницевий шар QOI.

Така асиметрія є ідеальною для моделі використання «One Write, Many Reads» (один запис, багато читань), що характерно для медіа-серверів та архівів відеоспостереження, де контент стискається один раз, а переглядається багаторазово.

4.4.2 Аналіз навантаження на CPU (AMD Ryzen 5600X)

Під час серії стрес-тестів на ПК було проаналізовано завантаження центрального процесора. Оскільки алгоритм реалізовано мовою JavaScript (яка виконується в одному потоці), очікувалось навантаження, еквівалентне одному логічному ядру (рис. 4.9).

Name	Status	21% CPU	74% Memory	0% Disk	0% Network
Node.js JavaScript Runtime (3)		8.1%	1,668.4 MB	19.0 MB/s	0 Mbps
Node.js JavaScript Runtime		8.1%	1,591.5 MB	19.0 MB/s	0 Mbps
Node.js JavaScript Runtime		0%	23.6 MB	0 MB/s	0 Mbps
Node.js JavaScript Runtime		0%	53.4 MB	0 MB/s	0 Mbps

Рисунок 4.9 – Графік навантаження на CPU Ryzen 5600X

Як видно з діаграми навантаження, під час інтенсивної пакетної обробки утилізація CPU не перевищувала 15–18 %. Це добре підкреслює, що алгоритм досить «легкий». Він не блокує роботу операційної системи й залишає помітний запас ресурсів для паралельних задач, наприклад для мережевої взаємодії чи аналітики. Такий результат виглядає практично важливим, бо в реальних умовах обробка рідко існує окремо від усього іншого. Також це натякає на нормальний потенціал масштабування. Якщо залучити Web

Workers, можна розподілити обчислення на всі ядра й отримати прискорення приблизно у 6–12 разів.

4.5 Стрес-тестування на висококонтрастних даних

Щоб реально зрозуміти, де метод працює, а де – ні (тобто зрозуміти межі застосування), провели додаткове тестування. Взяли специфічний контент, так званий «Line Art» (чорно-біла графіка).

Для таких зображень характерні дуже різкі контрастні переходи, а плавні градієнти там майже відсутні. Це, як видно, завжди складний випадок для більшості схем стиснення.

Для аналізу відібрали два типи картинок. Перший – це зображення з високою щільністю деталей («Urban Graphic»), і другий – з низькою щільністю («Portrait»). Це дозволяє нам подивитися, як метод поводиться не лише на «жорстких» переходах, але й на сценах із різним рівнем насиченості деталями. Це важливо.

4.5.1 Сценарій А: висока щільність деталей (Negative Case)

На зображенні урбаністичної графіки (рис. 4.10), насиченому дрібними текстурами (штрихування, дощ), спостерігається значне падіння якості.

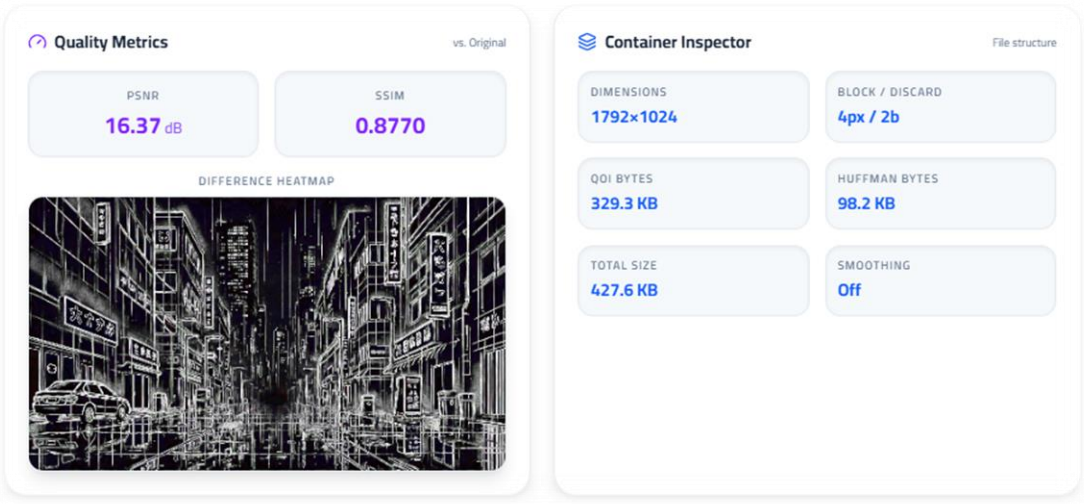


Рисунок 4.10 – Скріншот статистики для міста

Як бачимо з рисунку 4.10, PSNR знизився до критичних 16.37 дБ. Причиною є щільність деталей перевищує роздільну здатність сітки вузлових точок. Блокове усереднення (4×4) діє як фільтр низьких частот, перетворюючи групи штрихів на суцільні сірі плями.

Але при цьому досягнуто рекордного стиснення 12.57x (0.43 Мбайт проти 5.25 Мбайт Raw), що підтверджує здатність методу радикально зменшувати ентропію складних сцен.

4.5.2 Сценарій Б: Низька щільність деталей (Positive Case)

Натомість, при обробці лінійного портрета (рис. 4.11) алгоритм продемонстрував принципово інші результати.

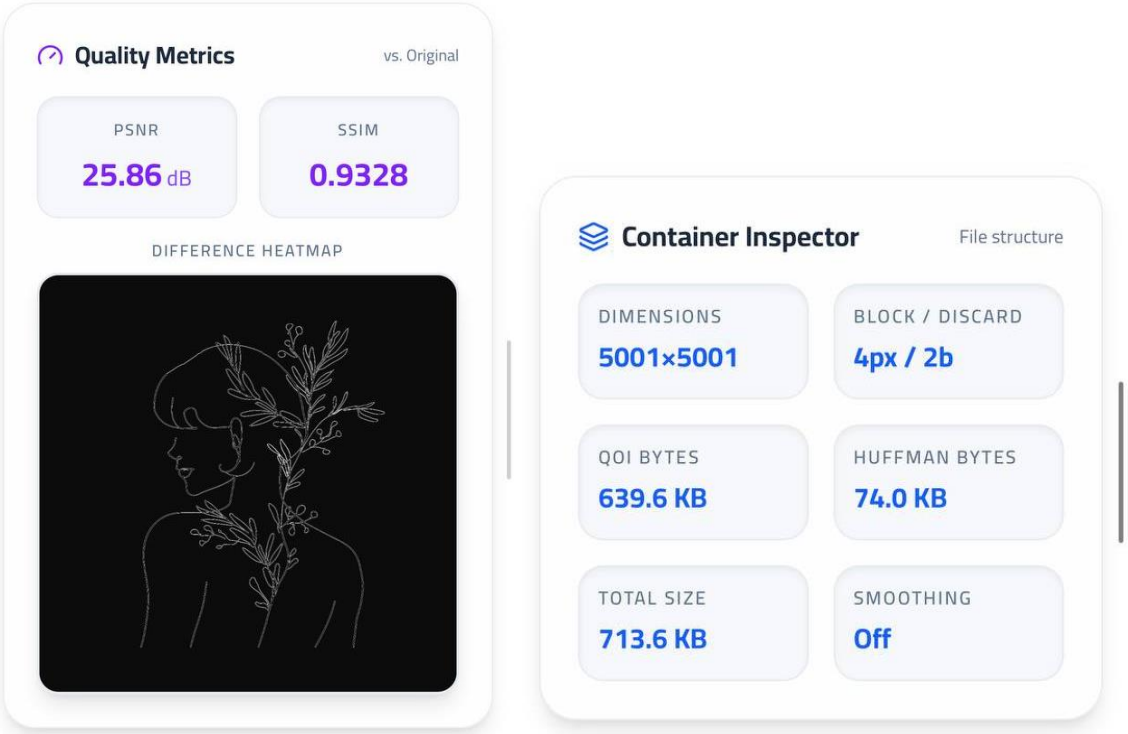


Рисунок 4.11 – Скріншот статистики для портрета

Показники PSNR та SSIM залишаються на високому рівні, порівнянному з фотореалістичними зображеннями. Причиною є великі однорідні області (фон, заливка), які кодуються без втрат. Спотворення вносяться лише

безпосередньо на лініях контурів, але їхня частка в загальній площі кадру є незначною.

Ефективність методу на графічних зображеннях більше визначається не самим типом контенту («фото» чи «графіка»), а тим, наскільки щільно розміщені деталі, тобто локальною дисперсією. Якщо зображення просте, з великими однорідними зонами, як у схемах, діаграмах або акуратній ілюстративній графіці, метод працює впевнено і дає відчутний результат. Але на складніших випадках це вже помітно. Високочастотний шум, густе штрихування, дрібні повторювані елементи. Особливо коли розмір блоку виявляється більшим за крок цих деталей. У такій ситуації частина структури починає «змішуватися», і ефективність знижується, що треба враховувати при виборі параметрів.

4.6 Глибинний аналіз внутрішньої структури

Дослідження структури контейнера .kmr показало адаптивність формату до параметрів стиснення.

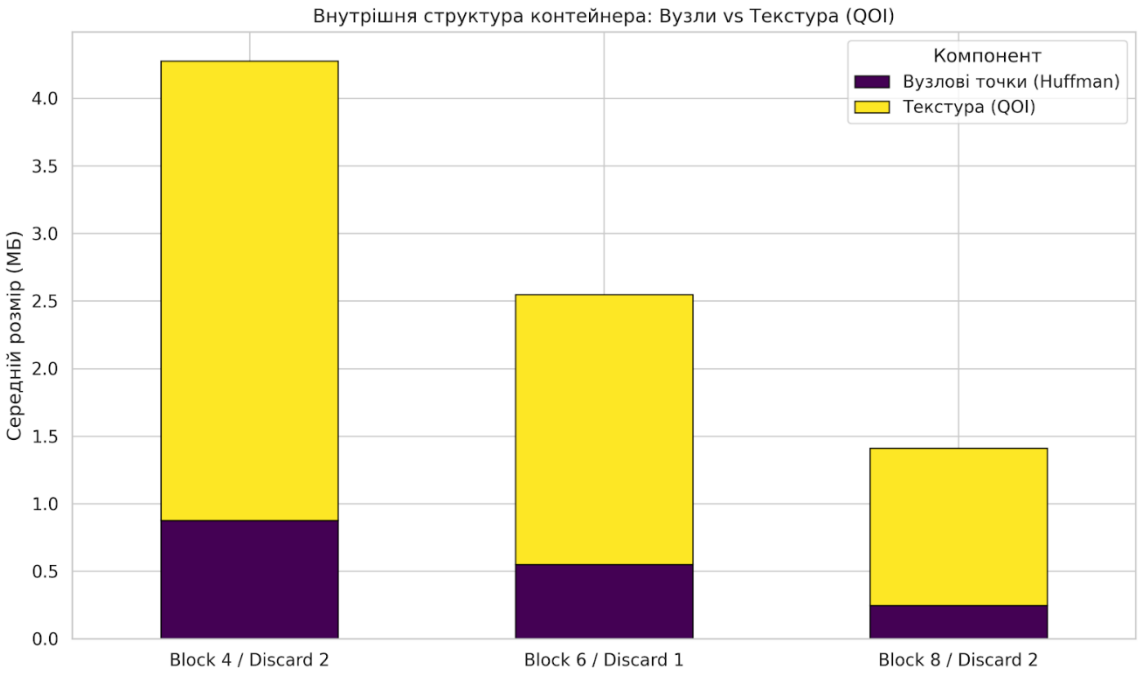


Рисунок 4.12 – Внутрішня структура контейнера

При збільшенні розміру блоку з 4 × 4 до 8 × 8 частка даних, що припадає на вузлові точки (Huffman), помітно падає: орієнтовно з 30 % до 5–10 %. У цей момент основну частину файлу починає займати потік текстурних різниць, який кодується через QOI. Це виглядає логічно. Вузлів стає менше, а значить і потреба в їх щільному кодуванні знижується.

У результаті гібридний підхід показує себе доволі стабільно: бітовий «бюджет» фактично перерозподіляється між низькочастотною складовою (вузлові точки) та високочастотною (текстура). І це відбувається природно через зміну параметра блоку, без ускладнення самого алгоритму.

Висновки до розділу 4

У четвертому розділі проведено експериментальну перевірку роботи апаратно-програмного комплексу та порівняння запропонованого методу стиснення з існуючими форматами. На основі тестування на вибірці з 300 зображень та відеопотоці можна зробити такі висновки.

Комбінований підхід (YCrCb + QOI + Huffman) показав досить високий коефіцієнт стиснення. Якщо брати режим підвищеної якості, то це десь 13.65x, а у режимі максимального стиснення ми отримали 48.7x. Це дуже добре. Наприклад, це у 6–15 разів більше, ніж у форматів без втрат на кшталт PNG. Причому за розміром фінального файлу метод цілком може конкурувати з JPEG (наприклад, при якості Q=90), особливо якщо йдеться про зображення з великими, однорідними ділянками. Як видно, такий результат підтверджує ефективність обраної схеми.

Об’єктивні метрики підтверджують прийнятну для ока якість. У робочих режимах стиснення показник структурної подібності SSIM тримається в межах 0,95–0,99, тобто геометрія сцени зберігається. Окремо варто відзначити, що метод фактично працює як фільтр низьких частот і помітно приглушує шум сенсора, що є корисним для систем технічного зору.

Алгоритм показав хорошу продуктивність. На звичайному ПК обробка кадру 4K займає близько 180 мс/МП при завантаженні CPU не вище 18 %. Виявлена асиметрія між кодуванням і декодуванням: декодування відбувається приблизно в 4 рази швидше (~54 мс/МП), що знижує вимоги до клієнтського обладнання під час перегляду архівів.

На Raspberry Pi 4 вдалося досягти стабільних 25–30 FPS із затримкою 30–45 мс, що підтверджує можливість застосування методу в потоковому відео для IoT-систем. Реалізація на STM32F746 показала, що алгоритм може працювати й в умовах обмеженої оперативної пам’яті, без підключення важких бібліотек обробки зображень.

Стрес-тестування на висококонтрастній графіці типу «Line Art» виявило слабке місце: при великій кількості дрібних деталей PSNR падає до 16.40 дБ. Тому область застосування методу логічно обмежується відеоспостереженням, передачею фотореалістичних сцен та інтерфейсів, але не підходить для архівації креслень чи текстових зображень, де критично важлива максимальна чіткість.

ВИСНОВКИ

У магістерській кваліфікаційній роботі вирішено актуальну науково-прикладну задачу розробки ефективного методу стиснення проміжних кадрів відеоп На основі теоретичних напрацювань і практичних експериментів отримано такі результати:

1) виконано аналіз предметної області. Він показав, що стандартні відеокодеки H.264/H.265 дають високу ефективність, але для мікроконтролерів це занадто важко за ресурсами. А прості підходи на кшталт RLE чи QOI, навпаки, працюють швидко, проте не дають потрібного рівня компресії для поставленої задачі;

2) обґрунтовано потребу в гібридному методі, де поєднуються попередня обробка у просторі YCrCb, виділення вузлових точок і подальше ентропійне кодування. Такий підхід виглядає логічним саме для проміжних кадрів;

3) розроблено комбінований метод стиснення, що базується на розділенні зображення на дві частини. Низькочастотна складова представлена вузловими точками, а високочастотна деталізація задається різницевим сигналом. Для вузлових точок адаптовано кодування Хаффмана, для різницевої частини використано QOI. Якість і розмір можна регулювати доволі прямо через зміну розміру блоку усереднення. Це зручно на практиці;

4) реалізовано метод у вигляді вебзастосунку з клієнт-серверною архітектурою на базі Next.js. Інструмент дозволяє виконувати стиснення у браузері, показувати артефакти через теплові карти та детальніше дивитися на структуру стисненого контейнера. Тобто це не лише демонстрація, а й робочий інструмент для аналізу;

5) виконано апаратну реалізацію на базі STM32F746G-DISCO та Raspberry Pi 4. Показано, що алгоритм може працювати за жорстких апаратних обмежень (зокрема 16 Мбайт RAM і без складних обчислювальних блоків

для частини операцій) і при цьому залишатися придатним для потокової обробки в реальному часі;

6) проведено комплексне тестування ефективності. Воно показало, що запропонований метод зменшує обсяг відеоданих у 17–48 разів порівняно з сирим форматом, суттєво випереджає PNG і в окремих режимах наближається до рівня JPEG. При цьому декодування відбувається швидко (до 4 разів швидше за кодування), а структурна подібність зображення залишається високою (SSIM > 0,95).

Практична цінність роботи полягає в тому, що отримано готовий програмний модуль, який можна інтегрувати в системи відеоспостереження, IoT-пристрої та вебсервіси. Це дає змогу зменшити навантаження на канали зв’язку й сховища без потреби в дорогому апаратному кодуванніотоку для вбудованих систем та веб-застосунків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Krainyk Y. Combined Run-Length and Huffman Encoding for Image Compression. Aug., 2022. 15 p. (Preprint). DOI: 10.21203/rs.3.rs-1982410/v1.

2. Yang Y., Peng Y., Liu Zh. A Fast Algorithm for YCbCr to RGB Conversion. *IEEE Transactions on Consumer Electronics*. 2007. Vol. 53, Is. 4. P. 1490–1493. DOI: 10.1109/tce.2007.4429242.

3. Stabno M., Wrembel R. RLH: Bitmap compression technique based on run-length and Huffman encoding. Information Systems. 2009. T. 34, № 4–5. C. 400–414. DOI: doi.org/10.1016/j.is.2008.11.002 (дата звернення: 10.10.2025).

4. Miano J. Compressed image file formats: JPEG, PNG, GIF, XBM, BMP. Reading, Ma : Addison Wesley Longman, 1999. 264 p.

5. Крайник Я., Дзяман Є. Застосування алгоритму QOI у стисненні зображень. *Інформаційні технології та інженерія* : Всеукр. науково-практ. конф. молодих вчен., аспірантів і студентів. 2023. С. 70–71.

6. Dominic S., QOI – The Quite OK Image Format. URL: https://qoiformat.org/qoi-specification.pdf (Last accessed: 10.10.2025).

7. Apostolico A. Fast gapped variants for Lempel-Ziv-Welch compression. *Information and Computation*. 2007. Vol. 205, Is. 7. P. 1012–1026. DOI: 10.1016/j.ic.2007.03.001.

8. Ziv J. The Universal LZ77 Compression Algorithm Is Essentially Optimal for Individual Finite-Length *N*-Blocks. IEEE Transactions on Information Theory. 2009. Vol. 55, № 5. P. 1941–1944. DOI: 10.1109/tit.2009.2016069.

9. Wu Y. Learned Block-based Hybrid Image Compression. IEEE Transactions on Circuits and Systems for Video Technology. 2021. P. 1. DOI: 10.1109/tcsvt.2021.3119660.

10. A Novel Image Compression method using Wavelet Coefficients and Huffman Coding / S. Thomas та ін. Journal of Engineering Research. 2023. DOI: 10.1016/j.jer.2023.08.015.

11. Hybrid compression technique for image hiding using Huffman, RLE and DWT / P. V Materials Today: Proceedings. 2022. DOI: 10.1016/j.matpr.2021.12.346.

12. Hu Y.-C., Chang C.-C. A new lossless compression scheme based on Huffman coding scheme for image compression. Signal Processing: Image Communication. 2000. Vol. 16, no. 4. P. 367–372. DOI: 10.1016/s0923-5965(99)00064-8.

13. Adjustable compression method for still JPEG images / J. Mora Pascual та ін. Signal Processing: Image Communication. 2015. Vol. 32. P. 16–32. DOI: 10.1016/j.image.2015.01.004.

14. Koc B., Arnavut Z., Koçak H. The pseudo-distance technique for parallel lossless compression of color-mapped images. Computers & Electrical Engineering. 2015. Vol. 46. P. 456–470. DOI: 10.1016/j.compeleceng.2015.01.003.

15. Wang J., Zeng L. A region-based hierarchical image compression method with simulated visual perception. Digital Signal Processing. 2024. Vol. 145. P. 104339. DOI: 10.1016/j.dsp.2023.104339.

16. Liu R., Li M., Ma L. Efficient in-situ image and video compression through probabilistic image representation. Signal Processing. 2023. C. 109268. DOI: 10.1016/j.sigpro.2023.109268.

17. System and method for a multi-primary wide gamut color system : patent US11069280B2 United States. Applied on 28.10.2020 ; published on 20.07.2021. 143 p. URL: https://patents.google.com/patent/US11069280B2/en?q=(RGB)&oq=RGB (Last accessed: 10.10.2025).

18. Image processing method, image forming apparatus, and storage medium : patent US20220053107A1 United States. Applied on 17.08.2021 ; published on 17.02.2022. 13 p. URL: https://patents.google.com/patent/US20220053107A1/en?q=(YCrCb)&oq=YCrCb (Last accessed: 10.10.2025).

19. Video compression method, device and computer readable storage medium : patent CN109618173B China. Applied on 17.12.2018 ; published on 12.04.2019. 17 p. URL:https://patents.google.com/patent/CN109618173B/en?q=(YCrCb)&oq=YCrCb (Last accessed: 10.10.2025).

ДОДАТОК А
Графічні матеріали

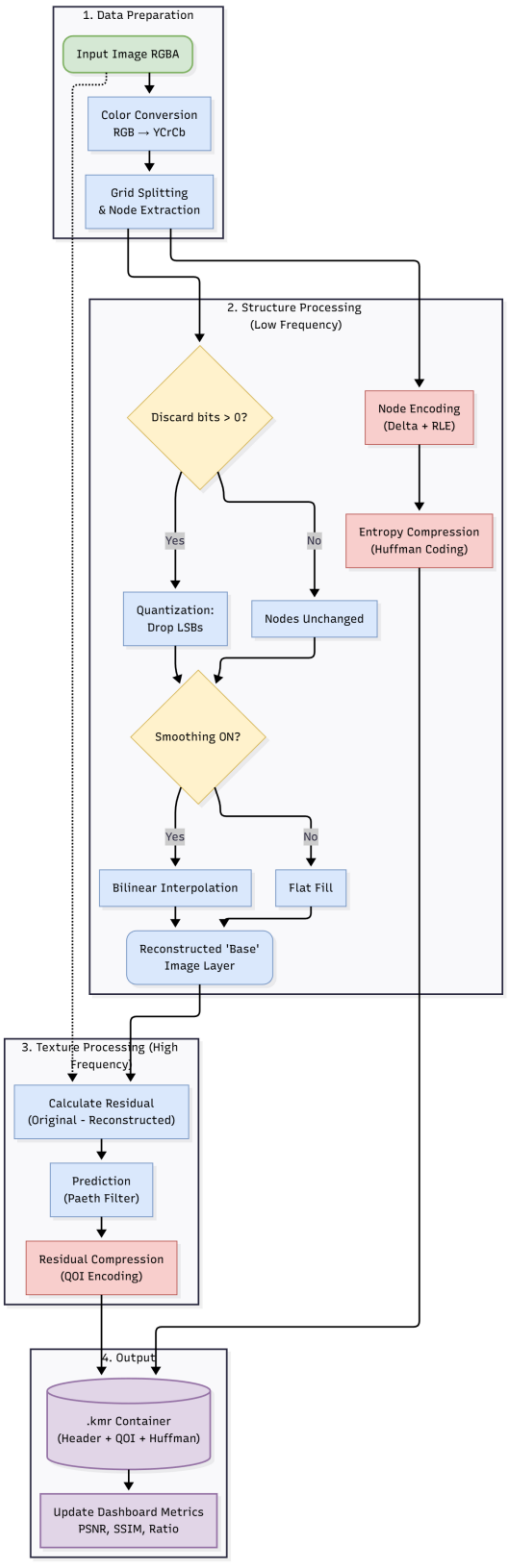


Рисунок А.1 – Архітектура вебзастосунку

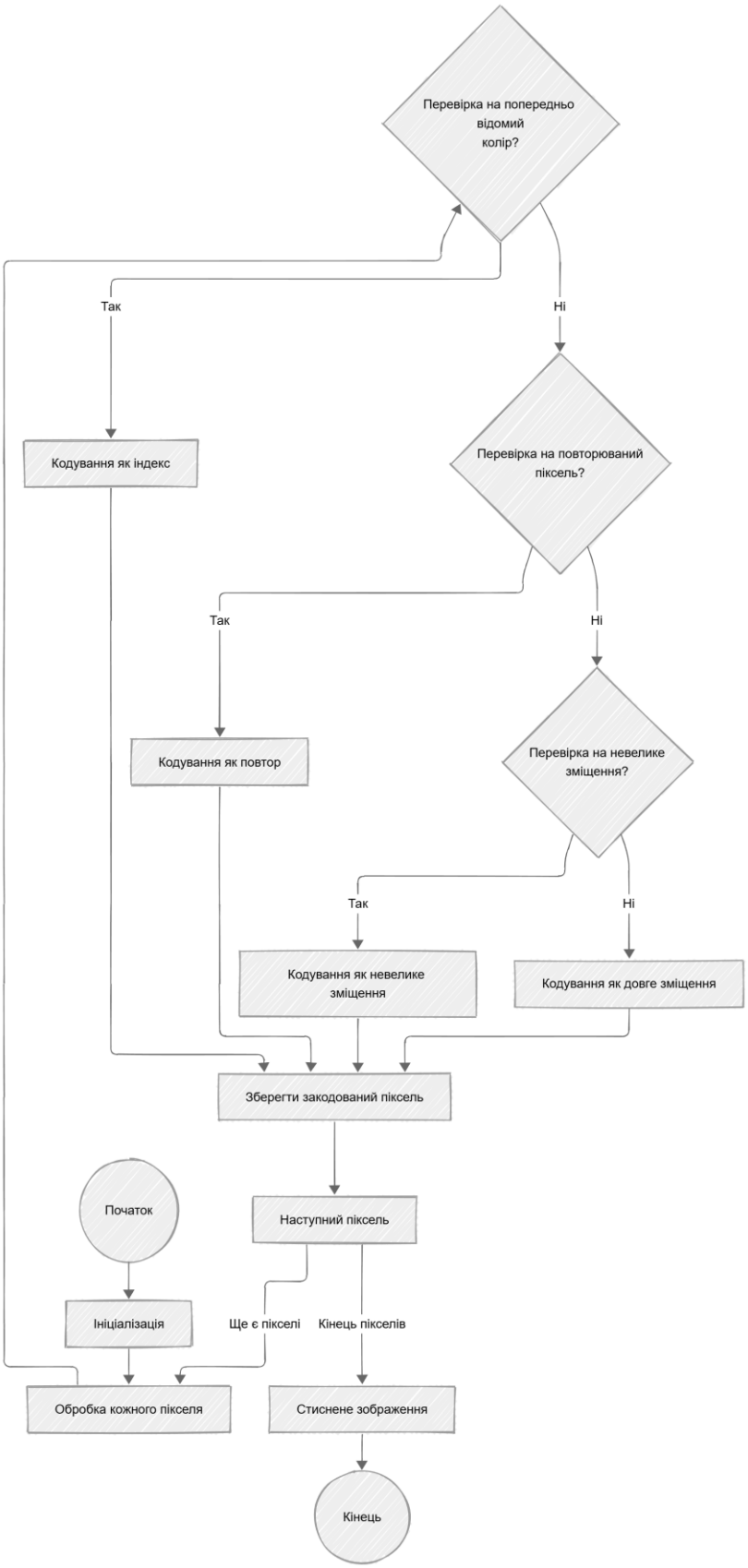


Рисунок А.2 – Принцип кодування алгоритмом QOI

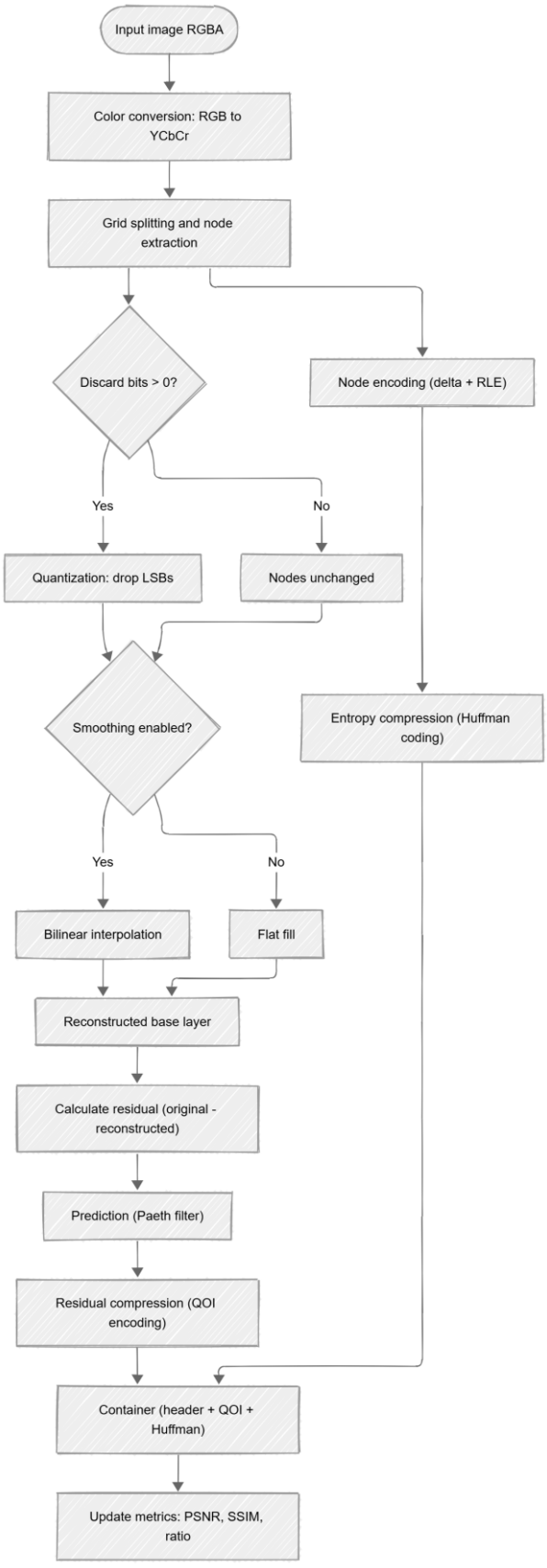
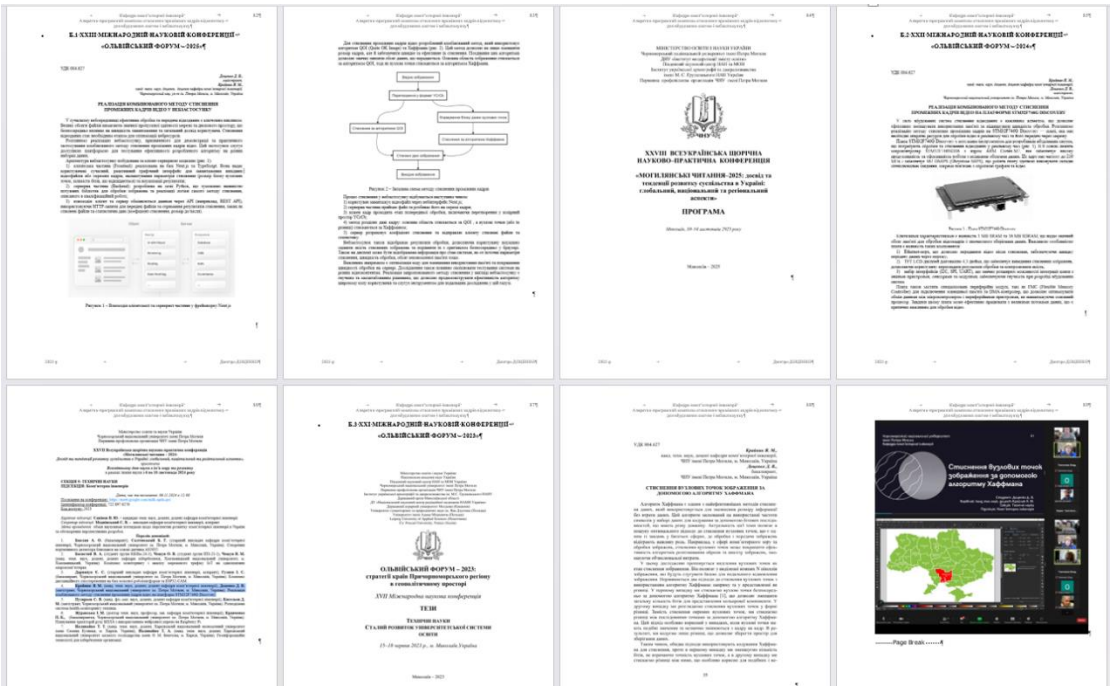


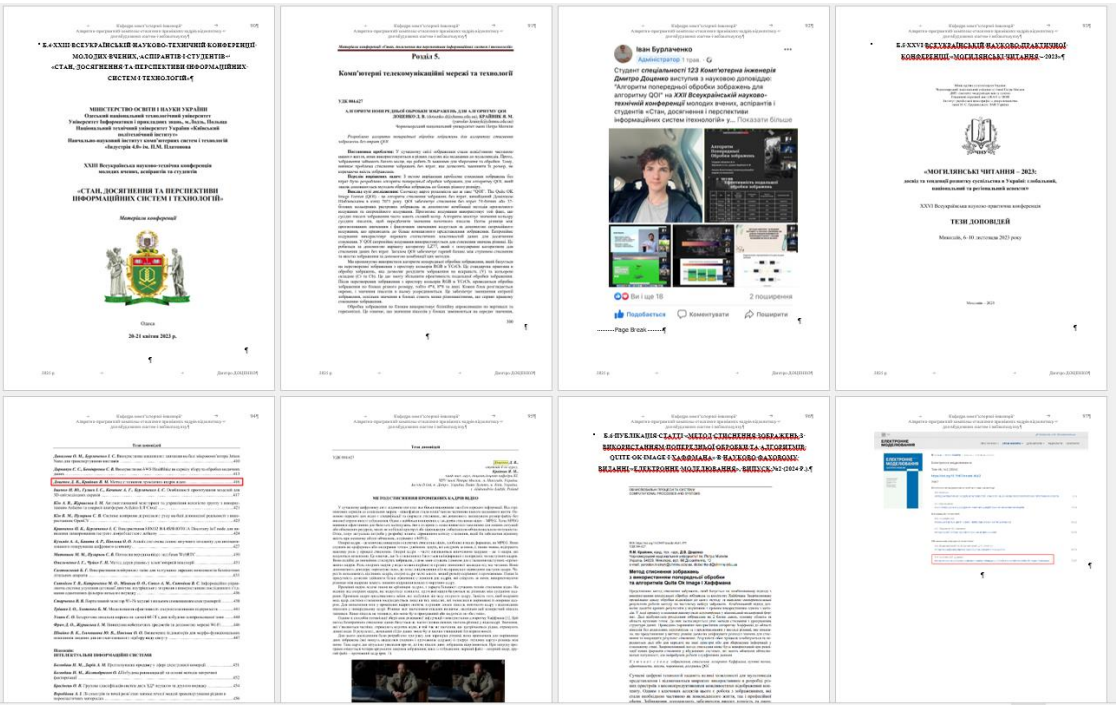
Рисунок А.3 – Принцип кодування Хаффманом

Рисунок А.4 – Процес кодування Хаффмана

ДОДАТОК Б

Матеріали апробації роботи





Б.1 XXVIII Всеукраїнська науково-практична конференція «Могилянські читання– 2025»

УДК 004.627

Доценко Д. В.,
магістрант,
Крайник Я. М.,
канд. техн. наук, доцент, доцент кафедри комп'ютерної інженерії,
Чорноморський нац. ун-т ім. Петра Могили, м. Миколаїв, Україна

РЕАЛІЗАЦІЯ КОМБІНОВАНОГО МЕТОДУ СТИСНЕННЯ ПРОМІЖНИХ КАДРІВ ВІДЕО У ВЕБЗАСТОСУНКУ

У сучасному вебсередовищі ефективна обробка та передача відеоданих є ключовим викликом. Великі обсяги файлів вимагають значної пропускної здатності мережі та дискового простору, що безпосередньо впливає на швидкість завантаження та загальний досвід користувача. Стиснення відеоданих стає необхідним етапом для оптимізації вебресурсів.

Розглянемо реалізацію вебзастосунку, призначеного для демонстрації та практичного застосування комбінованого методу стиснення проміжних кадрів відео. Цей застосунок слугує доступною платформою для тестування ефективності розробленого алгоритму на різних наборах даних.

Архітектура вебзастосунку побудована за клієнт-серверною моделлю (рис. 1):

1) клієнтська частина (Frontend): реалізована на базі Next.js та TypeScript. Вона надає користувачеві сучасний, реактивний графічний інтерфейс для завантаження вихідних відеофайлів або окремих кадрів, налаштування параметрів стиснення (розмір блоку вузлових точок, кількість бітів, що відкидаються) та візуалізації результатів;

2) серверна частина (Backend): розроблена на мові Python, що зумовлено наявністю потужних бібліотек для обробки зображень та реалізації логіки самого методу стиснення, описаного в кваліфікаційній роботі;

3) взаємодія: клієнт та сервер обмінюються даними через API (наприклад, REST API), використовуючи HTTP-запити для передачі файлів та отримання результатів стиснення, таких як стиснені файли та статистичні дані (коефіцієнт стиснення, розмір до/після).

Рисунок 1 – Взаємодія клієнтської та серверної частини у фреймворку Next.js

Для стиснення проміжних кадрів відео розроблений комбінований метод, який використовує алгоритми QOI (Quite OK Image) та Хаффмана (рис. 2). Цей метод дозволяє не лише зменшити розмір кадрів, але й забезпечити швидке та ефективне їх стиснення. Поєднання цих алгоритмів дозволяє значно знизити обсяг даних, що передаються. Основна область зображення стискається за алгоритмом QOI, тоді як вузлові точки стискаються за алгоритмом Хаффмана.

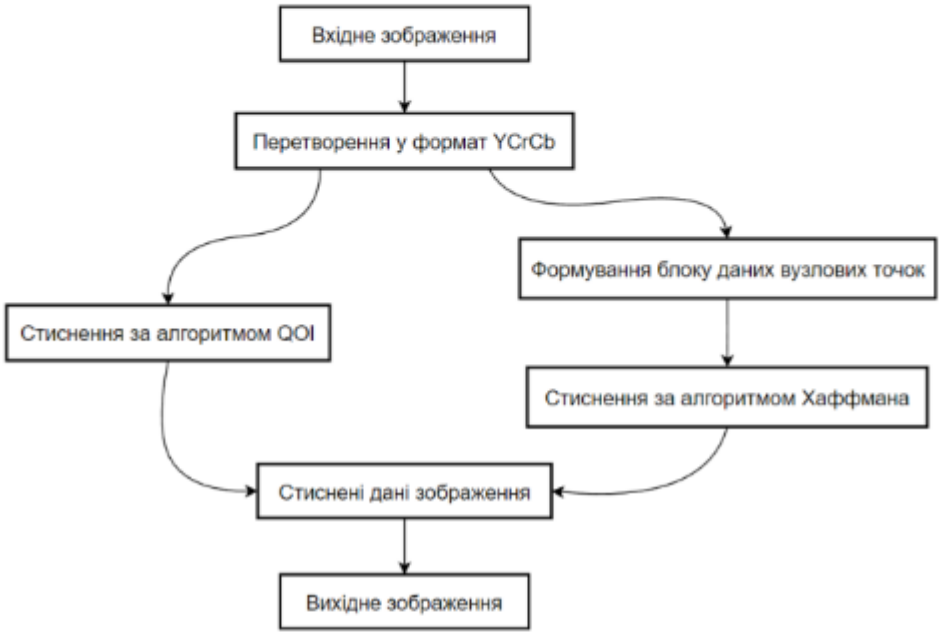


Рисунок 2 – Загальна схема методу стиснення проміжних кадрів

Процес стиснення у вебзастосунку відбувається наступним чином:

1) користувач завантажує відеофайл через вебінтерфейс Next.js;

2) серверна частина приймає файл та розбиває його на окремі кадри;

3) кожен кадр проходить етап попередньої обробки, включаючи перетворення у колірний простір YCrCb;

4) метод розділяє дані кадру: основна область стискається за QOI , а вузлові точки (або їх різниці) стискаються за Хаффманом;

5) сервер розраховує коефіцієнт стиснення та відправляє клієнту стиснені файли та статистику.

Вебзастосунок також відображає результати обробки, дозволяючи користувачу візуально оцінити якість стиснених зображень та порівняти їх з оригіналом безпосередньо у браузері. Також на дисплеї може бути відображена інформація про стан системи, як-от поточні параметри стиснення, швидкість обробки, обсяг зекономленої пам'яті тощо.

Важливим напрямком є оптимізація коду для зменшення використання пам'яті та покращення швидкості обробки на сервері. Дослідження також повинно охоплювати тестування системи на різних відеоконтентах. Реалізація запропонованого методу стиснення у вигляді вебзастосунку є гнучким та масштабованим рішенням, що дозволяє продемонструвати ефективність алгоритму широкому колу користувачів та слугує інструментом для подальших досліджень у цій галузі.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
ДНУ «Інститут модернізації змісту освіти»
Південний науковий центр НАН та МОН
Інститут української археографії та джерелознавства
імені М. С. Грушевського НАН України
Первинна профспілкова організація ЧНУ імені Петра Могили

XXVIII ВСЕУКРАЇНСЬКА ЩОРІЧНА НАУКОВО–ПРАКТИЧНА КОНФЕРЕНЦІЯ

«МОГИЛЯНСЬКІ ЧИТАННЯ–2025: досвід та тенденції розвитку суспільства в Україні: глобальний, національний та регіональний аспекти»

ПРОГРАМА

Миколаїв, 10–14 листопада 2025 року

Миколаїв – 2025

Б.2 XXVII Всеукраїнська щорічна науково-практична конференція «Могилянські читання – 2024»

УДК 004.627

Крайник Я. М.,
канд. техн. наук, доцент, доцент кафедри комп'ютерної інженерії,
Доценко Д. В.,
магістрант,
Чорноморський національний університет ім. Петра Могили, м. Миколаїв, Україна

РЕАЛІЗАЦІЯ КОМБІНОВАНОГО МЕТОДУ СТИСНЕННЯ ПРОМІЖНИХ КАДРІВ ВІДЕО НА ПЛАТФОРМІ STM32F746G DISCOVERY

У світі вбудованих систем стиснення відеоданих є важливим аспектом, що дозволяє ефективно зменшувати використання пам'яті та підвищувати швидкість обробки. Розглянемо реалізацію методу стиснення проміжних кадрів на STM32F746G Discovery — платі, яка має необхідні апаратні ресурси для обробки відео в реальному часі та його передачі через мережу.

Плата STM32F746G Discovery є потужним інструментом для розробників вбудованих систем, що потребують обробки та стиснення відеоданих у реальному часі (рис. 1). В її основі лежить мікроконтролер STM32F746NGH6 з ядром ARM Cortex-M7, яке забезпечує високу продуктивність та ефективність роботи з великими обсягами даних. Це ядро має частоту до 216 МГц і забезпечує 462 DMIPS (Dhrystone MIPS), що робить плату здатною виконувати складні обчислювальні завдання, зокрема пов'язані з обробкою графіки та відео.

Рисунок 1 – Плата STM32F746G Discovery

Ключовими характеристиками є наявність 1 MB SRAM та 16 MB SDRAM, що надає значний обсяг пам'яті для обробки відеокадрів і тимчасового зберігання даних. Важливою особливістю плати є наявність таких компонентів:

1) Ethernet-порт, що дозволяє передавати відео після стиснення, забезпечуючи швидку передачу даних через мережу;
2) TFT LCD-дисплей діагоналлю 4,3 дюйма, що забезпечує виведення стиснених зображень, дозволяючи користувачу переглядати результати обробки та контролювати якість;
3) набір інтерфейсів (I2C, SPI, UART), що значно розширює можливості інтеграції плати з іншими пристроями, сенсорами та модулями, забезпечуючи гнучкість при розробці вбудованих систем.

Плата також містить спеціалізовані периферійні модулі, такі як FMC (Flexible Memory Controller) для підключення зовнішньої пам'яті та DMA-контролер, що дозволяє оптимізувати обмін даними між мікроконтролером і периферійними пристроями, не навантажуючи основний процесор. Завдяки цьому плата може ефективно працювати з великими потоками даних, що є критично важливим для обробки відео.

Міністерство освіти та науки України
Чорноморський національний університет імені Петра Могили
Первинна профспілкова організація ЧНУ імені Петра Могили

XXVII Всеукраїнська щорічна науково-практична конференція
«Могилянські читання – 2024:
Досвід та тенденції розвитку суспільства в Україні: глобальний, національний та регіональний аспекти»,
присвячена
Всесвітньому дню науки в ім'я миру та розвитку
в рамках тижня науки з 6 по 10 листопада 2024 року

СЕКЦІЯ 5: ТЕХНІЧНІ НАУКИ
ПІДСЕКЦІЯ: Комп'ютерна інженерія

Дата, час та посилання: 06.11.2024 о 12:00
Посилання на конференцію: https://meet.google.com/mdk-updu-prc
Ідентифікатор конференції: 722 897 6276
Код доступу: 2023

Керівник підсекції: **Савінов В. Ю.** – кандидат техн. наук, доцент, доцент кафедри комп'ютерної інженерії
Секретар підсекції: **Медвінський С. В.** – викладач кафедри комп'ютерної інженерії, аспірант
Мета проведення: обмін науковими поглядами щодо перспектив розвитку комп’ютерної інженерії в Україні та обговорення перспективних розробок.

Перелік доповідей:

1. **Баклан А. О.** (бакалаврант), **Салтовський Б. Г.** (старший викладач кафедри комп’ютерної інженерії, Чорноморський національний університет ім. Петра Могили, м. Миколаїв, Україна). Створення портативного детектора блискавок на основі датчика AS3935.
2. **Басистий В. А.** (студент групи КБЗІм-24-1), **Чешун О. В.** (студент групи ІПЗ-23-1), **Чешун В. М.** (канд. техн. наук, доцент, доцент кафедри кібербезпеки, Хмельницький національний університет, м. Хмельницький, Україна). Комплекс моніторингу і аналізу мережевого трафіку IoT на одноплатних мікрокомп’ютерах.
3. **Дарнапук Є. С.** (старший викладач кафедри комп’ютерної інженерії, аспірант), **Гуляєв І. С.** (бакалаврант, Чорноморський національний університет ім. Петра Могили, м. Миколаїв, Україна). Комплекс дистанційного спостереження на базі колісної робоплатформи та ESP32-CAM.
4. **Крайник Я. М.** (канд. техн. наук, доцент, доцент кафедри комп’ютерної інженерії), **Доценко Д. В.** (магістрант, Чорноморський національний університет ім. Петра Могили, м. Миколаїв, Україна). Реалізація комбінованого методу стиснення проміжних кадрів відео на платформі STM32F746G Discovery.
5. **Пузирьов С. В.** (канд. фіз.-мат. наук, доцент, доцент кафедри комп'ютерної інженерії), **Кисельов Д. М.** (магістрант, Чорноморський національний університет ім. Петра Могили, м. Миколаїв, Україна). Розподілена система health-моніторингу теплиць.
6. **Журавська І. М.** (доктор техн. наук, професор, зав. кафедри комп'ютерної інженерії), **Кравченко П. К.,** (бакалаврантка, Чорноморський національний університет ім. Петра Могили, м. Миколаїв, Україна). Планування траєкторій руху БПЛА з використанням нейронної мережі на Raspberry Pi.
7. **Наливайко Т. Т.** (канд. техн. наук, доцент, Харківський національний економічний університет імені Семена Кузнеця, м. Харків, Україна), **Наливайко Т. А.** (канд. техн. наук, доцент, Харківський національний університет міського господарства імені О. М. Бекетова, м. Харків, Україна). Геоінформаційні технології для кібербезпеки організації.

Б.3 XVII Міжнародна наукова конференція «Ольвійський форум – 2023»

Міністерство освіти і науки України
Національна академія наук України
Південний науковий центр НАН та МОН України
Чорноморський національний університет імені Петра Могили
Первинна профспілкова організація ЧНУ імені Петра Могили
Інститут української археографії та джерелознавства ім. М.С. Грушевського НАНУ
Державний архів Миколаївської області
ДУ «Національний науковий центр радіаційної медицини НАМН України»
Державний аграрний університет Молдови (Кишинів)
Університет гуманітарних та природничих наук ім. Яна Длугоша (Польща)
Університет імені Адама Міцкевича (Польща)
Leipzig University of Applied Sciences (Німеччина)
Ca' Foscari University, Venice (Італія).

ЧНУ

ОЛЬВІЙСЬКИЙ ФОРУМ – 2023: стратегії країн Причорноморського регіону в геополітичному просторі

XVII Міжнародна наукова конференція

ТЕЗИ

ТЕХНІЧНІ НАУКИ
СТАЛИЙ РОЗВИТОК УНІВЕРСИТЕТСЬКОЇ СИСТЕМИ ОСВІТИ

15–18 червня 2023 р., м. Миколаїв, Україна

Миколаїв – 2023

УДК 004.627

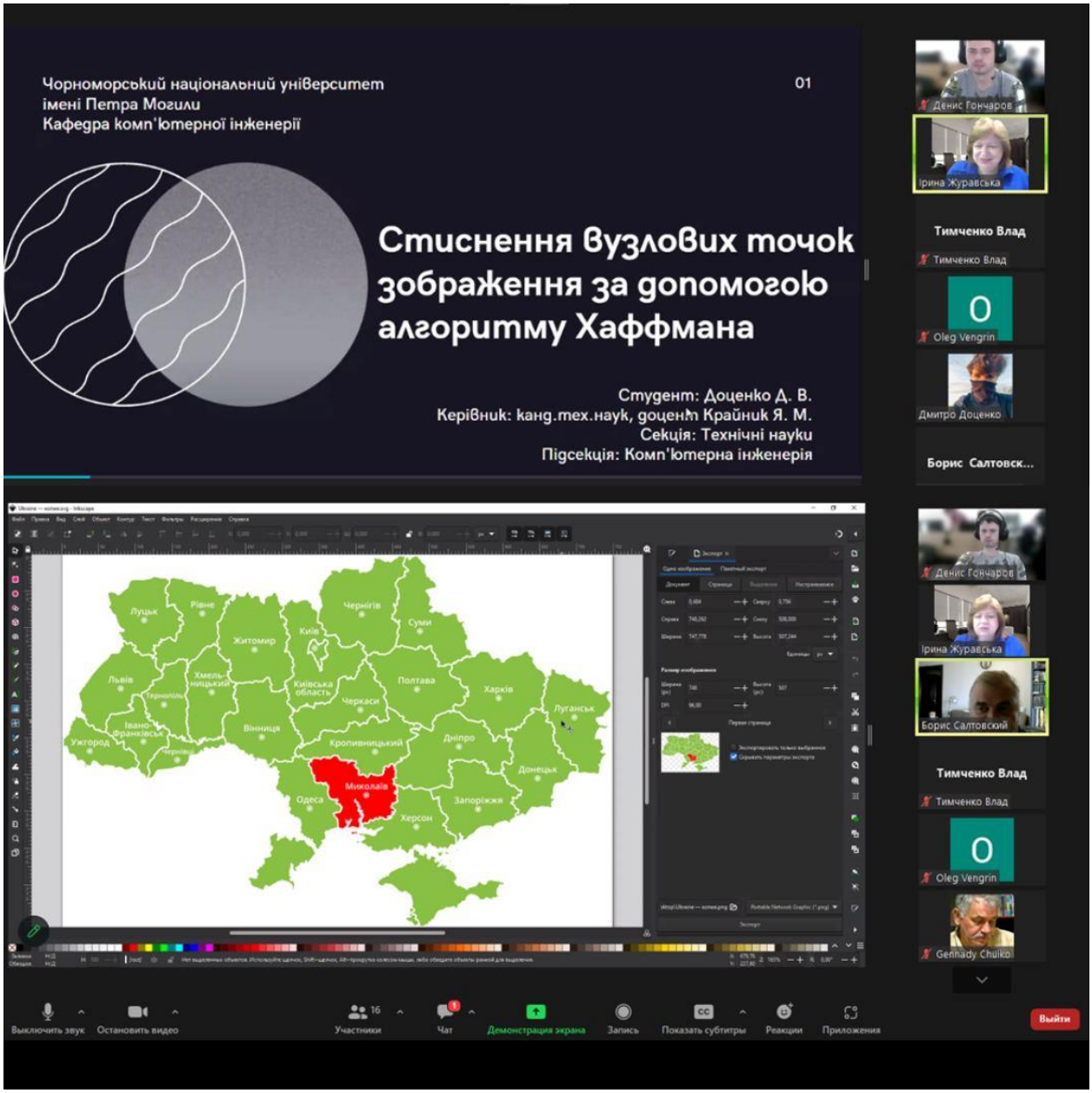
Крайник Я. М.,
канд. техн. наук, доцент кафедри комп'ютерної інженерії,
ЧНУ імені Петра Могили, м. Миколаїв, Україна
Доценко Д. В.,
бакалаврант,
ЧНУ імені Петра Могили, м. Миколаїв, Україна

СТИСНЕННЯ ВУЗЛОВИХ ТОЧОК ЗОБРАЖЕННЯ ЗА ДОПОМОГОЮ АЛГОРИТМУ ХАФФМАНА

Алгоритм Хаффмана є одним з найефективніших методів стиснення даних, який використовується для зменшення розміру інформації без втрати даних. Цей алгоритм заснований на використанні частоти символів у наборі даних для кодування за допомогою бітових послідовностей, що мають різну довжину. Актуальність цієї теми полягає в пошуку оптимального підходу до стиснення вузлових точок, що є одним із завдань у багатьох сферах, де обробка і передача зображень відіграють важливу роль. Наприклад, у сфері комп'ютерного зору та обробки зображень, стиснення вузлових точок може покращити ефективність алгоритмів розпізнавання образів та аналізу зображень, зменшуючи обчислювальні витрати.

У цьому дослідженні пропонується виділення вузлових точок як етап стиснення зображення. Він полягає у виділенні кожних N пікселів зображення, які будуть слугувати базою для подальшого відновлення зображення. Порівнюються два підходи до стиснення вузлових точок з використанням алгоритму Хаффмана: напряму та у представленні як різниці. У першому випадку ми стискаємо вузлові точки безпосередньо за допомогою алгоритму Хаффмана [1], що дозволяє зменшити загальну кількість бітів для представлення кольорової компоненти. У другому випадку ми розглядаємо стиснення вузлових точок у формі різниці. Замість стиснення окремих вузлових точок, ми стискаємо різниці між послідовними точками за допомогою алгоритму Хаффмана. Цей підхід особливо корисний у випадках, коли вузлові точки мають подібні значення та незначно змінюються з кадру на кадр. В результаті, ми кодуємо лише різниці, що дозволяє зберегти простір для зберігання даних.

Таким чином, обидва підходи використовують кодування Хаффмана для стиснення, проте в першому випадку ми зменшуємо кількість бітів, не втрачаючи точність вузлових точок, а в другому випадку ми стискаємо різниці між ними, що особливо корисно для подібних і не-


Чорноморський національний університет
імені Петра Могили
Кафедра комп'ютерної інженерії
01
Стиснення вузлових точок зображення за допомогою алгоритму Хаффмана
Студент: Доценко Д. В.
Керівник: канд.мех.наук, доцент Крайник Я. М.
Секція: Технічні науки
Підсекція: Комп'ютерна інженерія
Денис Гончаров
Ірина Журавська
Тимченко Влад
Oleg Vengrin
Дмитро Доценко
Борис Салтовск...
Борис Салтовский
Gennady Chuiko
Выключить звук
Остановить видео
Участники
Чат
Демонстрация экрана
Запись
Показать субтитры
Реакции
Приложения
Выйти

Б.4 XVII Міжнародна наукова конференція «Ольвійський форум – 2023»

Міністерство освіти і науки України
Національна академія наук України
Південний науковий центр НАН та МОН України
Чорноморський національний університет імені Петра Могили
Первинна профспілкова організація ЧНУ імені Петра Могили
Інститут української археографії та джерелознавства ім. М.С. Грушевського НАНУ
Державний архів Миколаївської області
ДУ «Національний науковий центр радіаційної медицини НАМН України»
Державний аграрний університет Молдови (Кишинів)
Університет гуманітарних та природничих наук ім. Яна Длугоша (Польща)
Університет імені Адама Міцкевича (Польща)
Leipzig University of Applied Sciences (Німеччина)
Ca' Foscari University, Venice (Італія).

**ОЛЬВІЙСЬКИЙ ФОРУМ – 2023:
стратегії країн Причорноморського регіону
в геополітичному просторі**

XVII Міжнародна наукова конференція

ТЕЗИ

**ТЕХНІЧНІ НАУКИ
СТАЛИЙ РОЗВИТОК УНІВЕРСИТЕТСЬКОЇ СИСТЕМИ ОСВІТИ**

15–18 червня 2023 р., м. Миколаїв, Україна

Миколаїв – 2023

УДК 004.45

Д. В. Доценко, бакалавр
І. С. Бурлаченко, старший викладач кафедри комп’ютерної інженерії
ЧНУ ім. Петра Могили, м. Миколаїв, Україна

ПРОДУКТИВНІСТЬ ORM ДЛЯ СЕРВЕРНИХ ФРЕЙМВОРКІВ ВЕБЗАСТОСУНКІВ

Hibernate та Sequelize - це дві з найпопулярніших технологій, які використовуються для роботи з базами даних в програмному забезпеченні. Знання цих технологій є важливим для розробників програмного забезпечення, які працюють з реляційними базами даних. У світі програмування використання ORM-бібліотек для роботи з базами даних є дуже поширеним, і Hibernate та Sequelize не є винятками. Актуальність цих технологій полягає в тому, що реляційні бази даних все ще залишаються одними з найбільш популярних типів баз даних, що використовуються в розробці програмного забезпечення. Hibernate та Sequelize дозволяють розробникам програмного забезпечення легше та ефективніше взаємодіяти з базами даних, що забезпечує швидший та більш надійний процес розробки.

ORM - це технологія, яка дозволяє програмістам працювати з базами даних, використовуючи об'єктно-орієнтований підхід. Вона автоматизує процес перетворення даних між об'єктами програми та таблицями в базі даних, спрощуючи розробку та підтримку програмного забезпечення. ORM дозволяє розробникам зосередитись на бізнес-логіці своєї програми, не задумуючись про деталі роботи з базою даних.

Розглянемо діаграми класів проєктів на базі Nest.js та Spring Boot. Діаграми класів є потужним інструментом для моделювання структури програмного забезпечення. Вони дозволяють візуалізувати класи, їх атрибути та зв'язки між ними.

Рисунок 1 - Діаграми класів проєкту на базі Nest.js

Nest.JS проєкт реалізовує додаток для бронювання готелів. В ньому використовуються чотири контролери. Наприклад "CategoriesController" та "RoomsController" відповідають за обробку HTTP-запитів, пов'язаних з категоріями номерів та номерами кімнат. "CategoriesController" використовує "categoriesService" для виконання операцій з категоріями номерів та "hotelsService" для отримання даних про готелі. Кожен метод контролера відповідає певному HTTP-запиту та викликає відповідний метод з "categoriesService", щоб виконати потрібну дію. В той же час, контролер "GuestsController" використовує об'єкти "guestsService" та "reservationsService", які передаються у конструкторі контролера для здійснення потрібних операцій з гостями та бронюваннями. В свою чергу "ReservationsController", визначає маршрути та відповідні методи для обробки запитів HTTP, пов'язаних із резервуванням. Контролер взаємодіє з іншими службами, такими як "CategoriesService" та "RoomsService", щоб виконувати операції CRUD щодо бронювань, категорій і кімнат.

Spring Boot проєкт реалізує додаток для керування списком справ. В ньому використовуються різні технології і бібліотеки, щоб забезпечити функціональність додатку. Наприклад конфігурація "ExposeEntityIdRestConfiguration" використовується для викладання ідентифікатора сутності. У методі "configureRepositoryRestConfiguration" відбувається конфігурація "RepositoryRestConfiguration" для викладання ідентифікатора сутності "Todo.class".

Б.5 XXIII Всеукраїнська науково-технічна конференція молодих вчених, аспірантів і студентів «Стан, досягнення та перспективи інформаційних систем і технологій»

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Одеський національний технологічний університет
Університет Інформатики і прикладних знань, м.Лодзь, Польща
Національний технічний університет України «Київський політехнічний інститут»
Навчально-науковий інститут комп’ютерних систем і технологій «Індустрія 4.0» ім. П.М. Платонова

XXIII Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів

«СТАН, ДОСЯГНЕННЯ ТА ПЕРСПЕКТИВИ ІНФОРМАЦІЙНИХ СИСТЕМ І ТЕХНОЛОГІЙ»

Матеріали конференції

Одеса

20-21 квітня 2023 р.

Матеріали конференції «Стан, досягнення та перспективи інформаційних систем і технологій»

Розділ 5.

Комп'ютерні телекомунікаційні мережі та технології

УДК 004.627

АЛГОРИТМ ПОПЕРЕДНЬОЇ ОБРОБКИ ЗОБРАЖЕНЬ ДЛЯ АЛГОРИТМУ QOI

ДОЦЕНКО Д. В. (dotsenko.d@chmnu.edu.ua), **КРАЙНИК Я. М.** (yaroslav.krainyk@chmnu.edu.ua)
Чорноморський національний університет імені Петра Могили

Розроблено алгоритм попередньої обробки зображень для алгоритму стиснення зображень без втрат QOI.

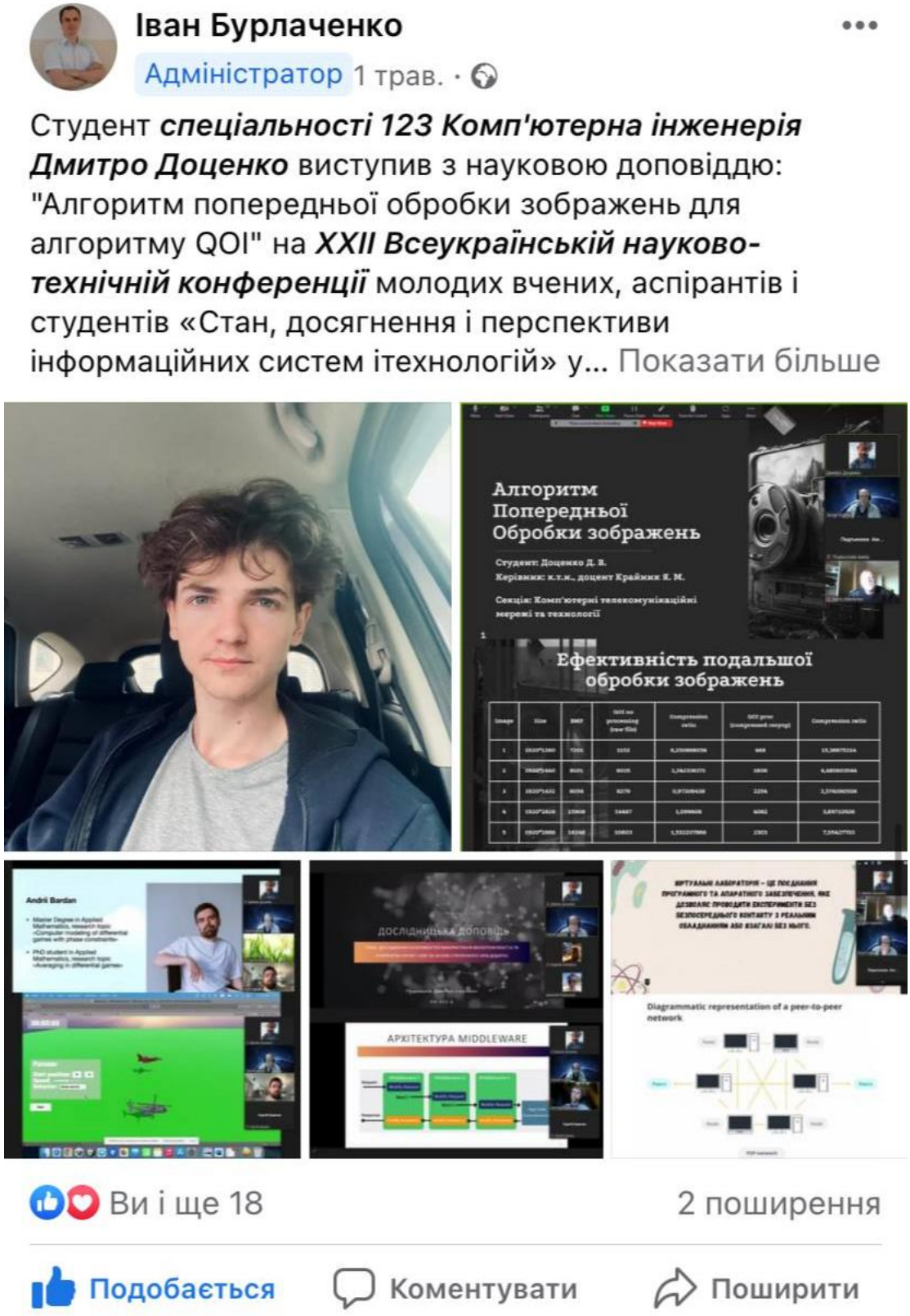
Постановка проблеми: У сучасному світі зображення стали невід'ємною частиною нашого життя, вони використовуються в різних галузях від медицини до мультимедіа. Проте, зображення займають багато місця, що робить їх важкими для зберігання та обробки. Тому, виникає проблема стиснення зображень без втрат, яке дозволить зменшити їх розмір, не втрачаючи якість зображення.

Перелік вирішених задач: З метою вирішення проблеми стиснення зображень без втрат було розроблено алгоритм попередньої обробки зображень для алгоритму QOI, який також доповнюється методом обробки зображень по блоках різного розміру.

Виклад суті дослідження: Спочатку варто розповісти що ж таке “QOI”. The Quite OK Image Format (QOI) - це алгоритм стиснення зображень без втрат, винайдений Домініком Шаблевським в кінці 2021 року. QOI забезпечує стиснення без втрат 24-бітних або 32-бітових кольорових растрових зображень за допомогою комбінації методів прогнозного кодування та ентропійного кодування. Прогнозне кодування використовує той факт, що сусідні пікселі зображення часто мають схожий колір. Алгоритм аналізує значення кольору сусідніх пікселів, щоб передбачити значення поточного пікселя. Потім різниця між прогнозованим значенням і фактичним значенням кодується за допомогою ентропійного кодування, що призводить до більш компактного представлення зображення. Ентропійне кодування використовує переваги статистичних властивостей даних для досягнення стиснення. У QOI ентропійне кодування використовується для стиснення значень різниці. Це робиться за допомогою варіанту алгоритму LZ77, який є популярним алгоритмом для стиснення даних без втрат. Загалом QOI забезпечує гарний баланс між ступенем стиснення та якістю зображення за допомогою комбінації цих методів.

Ми пропонуємо використати алгоритм попередньої обробки зображення, який базується на перетворенні зображення з простору кольорів RGB в YCrCb. Це стандартна практика в обробці зображень, яка дозволяє розділити зображення на яскравість (Y) та кольорові складові (Cr та Cb). Це дає змогу збільшити ефективність подальшої обробки зображення. Після перетворення зображення з простору кольорів RGB в YCrCb, проводиться обробка зображення по блоках різного розміру, тобто 4*4, 8*8 та інші. Кожен блок розглядається окремо, і значення пікселів в ньому усереднюються. Це забезпечує зменшення ентропії зображення, оскільки значення в блоках стають менш різноманітними, що сприяє кращому стисненню зображення.

Обробка зображення по блокам використовує білінійну апроксимацію по вертикалі та горизонталі. Це означає, що значення пікселів у блоках замінюються на середнє значення,


Іван Бурлаченко
Адміністратор 1 трав. ·
Студент спеціальності 123 Комп'ютерна інженерія Дмитро Доценко виступив з науковою доповіддю: "Алгоритм попередньої обробки зображень для алгоритму QOI" на XXII Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Стан, досягнення і перспективи інформаційних систем ітехнологій» у... Показати більше
Алгоритм Попередньої Обробки зображень
Ефективність подальшої обробки зображень
Andrii Bardan
АРХІТЕКТУРА MIDDLEWARE
Diagrammatic representation of a peer-to-peer network
Ви і ще 18
2 поширення
Подобається
Коментувати
Поширити

Б.6 XXVI Всеукраїнська науково-практична конференція «Могилянські читання – 2023»

Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили
ДНУ «Інститут модернізації змісту освіти»
Південний науковий центр НАН та МОН
Інститут української археографії та джерелознавства
імені М. С. Грушевського НАН України

«МОГИЛЯНСЬКІ ЧИТАННЯ – 2023:
досвід та тенденції розвитку суспільства в Україні: глобальний, національний та регіональний аспекти»

XXVI Всеукраїнська науково-практична конференція

ТЕЗИ ДОПОВІДЕЙ

Миколаїв, 6–10 листопада 2023 року

Миколаїв – 2023

Тези доповідей

Підсекція:
ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ СИСТЕМИ

Тези доповідей

УДК 004.627

Доценко ***Д. В.,***
студент 4-го курсу,
Крайник Я. М.,
канд. тех. наук, доцент,доцент кафедри КІ,
ЧНУ імені Петра Могили, м. Миколаїв, Україна,
Aerotech Ltd, м. Дніпро, Україна, Radar Systems, м. Київ, Україна;
t. Aleksandrów Łódzki, Poland

МЕТОД СТИСНЕННЯ ПРОМІЖНИХ КАДРІВ ВІДЕО

У сучасному цифровому світі відеоконтент стає все більш поширеним засобом передачі інформації. Від стрімінгових сервісів до соціальних мереж – відеофайли стали невід'ємною частиною нашого щоденного життя. Основою передачі цих відео є специфікації та формати стиснення, які дозволяють зменшувати розмір файлу без високої втрати якості зображення. Один з найбільш поширених стандартів стиснення відео – MPEG. Хоча MPEG виявився ефективним для багатьох застосувань, його складність може виявитися завеликою для певних ситуацій або обмежених ресурсів, таких як мобільні пристрої або відеокамери з обмеженою обчислювальною потужністю. Отже, існує актуальна потреба у розробці нового, спрощеного методу стиснення, який би забезпечив відмінну якість при меншому обсязі обчислень, порівняно з MPEG.

Опорні кадри – це ключова концепція в системах стиснення відео, особливо в таких форматах, як MPEG. Вони служать як «референс» або «відправна точка» для інших кадрів, які слідують за ними, і, таким чином, відіграють важливу роль у процесі стиснення. Опорні кадри – часто називаються ключовими кадрами – це ті кадри, які кодуються незалежно. Це означає, що їх стиснення не базується на інформації з попередніх чи наступних кадрів. Вони подібні до звичайних стиснутих зображень, і саме вони служать основою для стиснення наступних «проміжних» кадрів. Роль опорних кадрів у відео можна порівняти з роллю книжкової закладки під час читання. Вони допомагають декодеру «орієнтуватися», де почати відтворення або як правильно відтворити наступні кадри. Через їх незалежність від інших кадрів, опорні кадри часто мають вищий розмір порівняно з проміжними. Однак їх присутність дозволяє здійснити більш ефективне стиснення для кадрів, які слідують за ними, використовуючи різницю між кадрами замість повного кодування кожного окремого кадру.

Проміжні кадри, відомі також як «різницеві кадри», є серцем більшості сучасних технік стиснення відео. На відміну від опорних кадрів, які кодуються повністю, проміжні кадри базуються на різницях між сусідніми кадрами. Проміжні кадри представляють зміни, які відбулися від часу опорного кадру. Замість того, щоб кодувати весь кадр, система стиснення зосереджується лише на тих пікселях, які змінилися в порівнянні із опорним кадром. Для визначення змін у проміжних кадрах система порівнює кожен піксель поточного кадру з відповідним пікселем у попередньому кадрі. Різниця між значеннями пікселів визначає, наскільки цей конкретний піксель змінився. Якщо піксель не змінився, він може бути пропущений або кодується як «без змін».

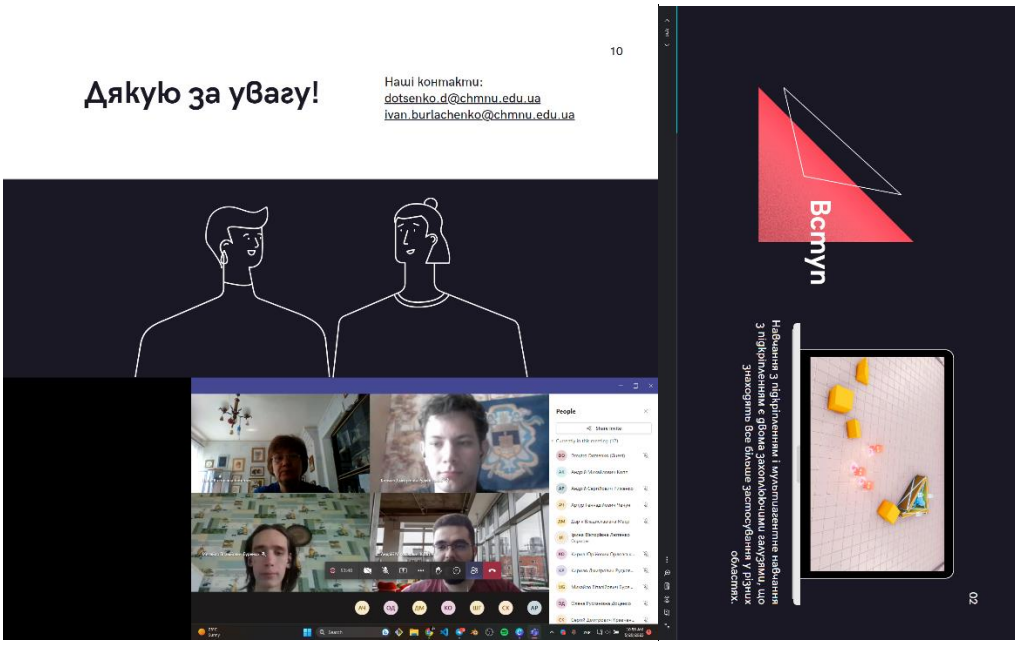
Одним зі способів оптимізації зберігання різницевої інформації є використання алгоритму Хаффмана [1]. Цей метод безвтратного стиснення даних базується на частоті появи певних значень різниці у відеокадрі. Значення, які з'являються частіше, отримують коротші коди, в той час як значення, що зустрічаються рідше, отримують довші коди. В результаті, загальний обсяг даних може бути значно зменшений без втрати якості.

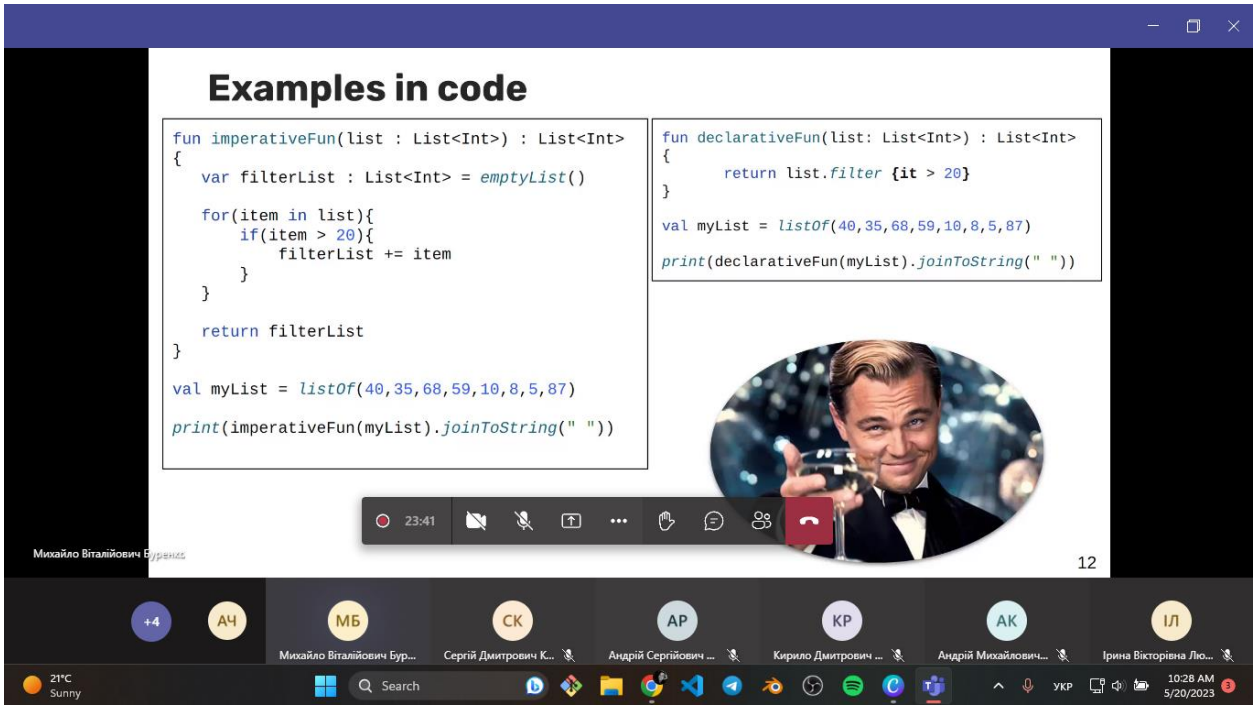
Для цього дослідження було розроблено програму для перевірки різниці, вона призначена для порівняння двох зображень (які можуть вважатися опорним і проміжним кадрами) і генерує «теплову карту» різниць між ними. Така карта дає візуальне уявлення про те, де і як пікселі двох зображень відрізняються. При запуску програми очікується чотири аргументи: ширина зображення; висота зображення; перший файл – опорний кадр; другий файл – проміжний кадр (рис. 1).

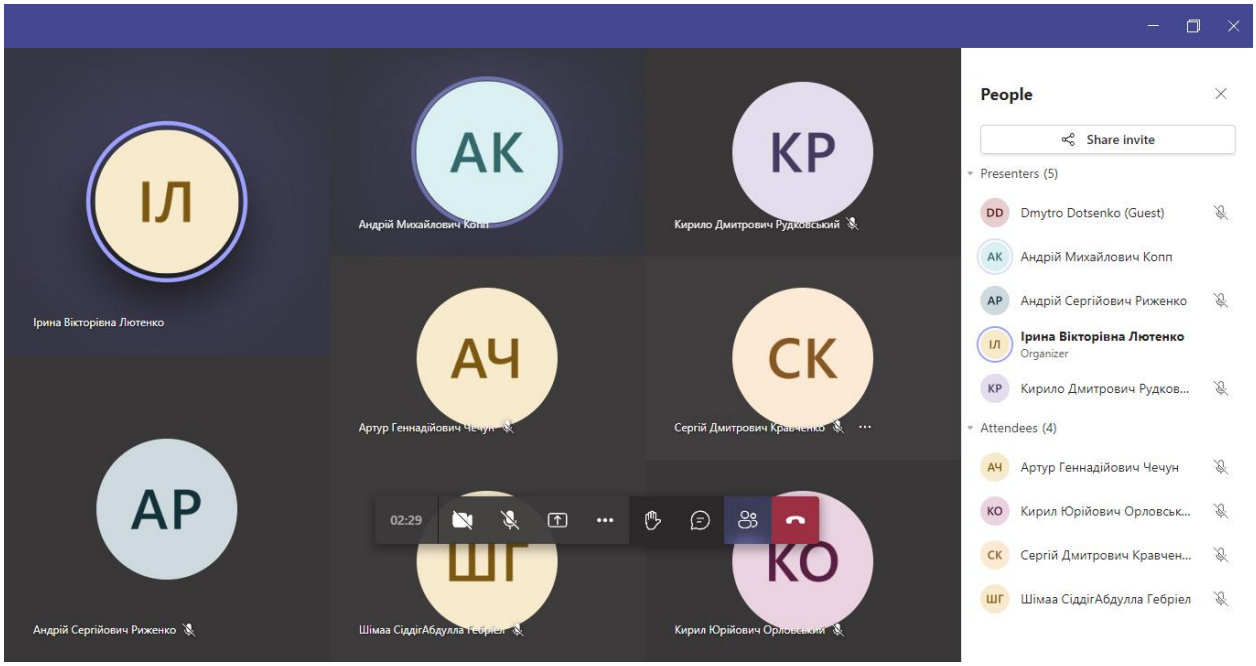
Рисунок 1 – Опорний кадр (а), проміжний кадр (б)

Далі програма читає обидва зображення піксель за пікселем, після чого для кожного пікселя визначається різниця в значеннях між опорним та проміжним кадром. Якщо різниця дорівнює нулю (тобто пікселі однакові),

Б.7 XXXI Міжнародна науково-практична конференція «Інформаційна технології: наука, техніка, технологія, освіта, здоров’я» MicroCAD-2023»




Examples in code
fun imperativeFun(list : List<Int>) : List<Int>
{
var filterList : List<Int> = emptyList()
for(item in list){
if(item > 20){
filterList += item
}
}
return filterList
}
val myList = listOf(40,35,68,59,10,8,5,87)
print(imperativeFun(myList).joinToString(" "))
fun declarativeFun(list: List<Int>) : List<Int>
{
return list.filter {it > 20}
}
val myList = listOf(40,35,68,59,10,8,5,87)
print(declarativeFun(myList).joinToString(" "))
12
23:41
Search


ІЛ
Ірина Вікторівна Лютенко
АК
Андрій Михайлович Копп
КР
Кирило Дмитрович Рудковський
АЧ
Артур Геннадійович Чечун
СК
Сергій Дмитрович Кравченко
АР
Андрій Сергійович Риженко
ШГ
Шімаа СіддігАбдулла Гебріел
КО
Кирил Юрійович Орловський
02:29
People
Share invite
Presenters (5)
Dmytro Dotsenko (Guest)
Андрій Михайлович Копп
Андрій Сергійович Риженко
Ірина Вікторівна Лютенко
Organizer
Кирило Дмитрович Рудков...
Attendees (4)
Артур Геннадійович Чечун
Кирил Юрійович Орловськ...
Сергій Дмитрович Кравчен...
Шімаа СіддігАбдулла Гебріел

ВИКОРИСТАННЯ MODEL-BASED ПІДХОДУ ДЛЯ МУЛЬТИАГЕНТНОГО НАВЧАННЯ З ПІДКРІПЛЕННЯМ

Доценко Д. В., Бурлаченко І. С.
Чорноморський національний університет імені Петра Могили
м. Миколаїв

В останні роки навчання з підкріпленням, особливо мультиагентне навчання з підкріпленням, знайшло широке застосування у багатьох областях, від робототехніки до ігрової індустрії. Величезна кількість ресурсів, що виділяється на взаємодію агентів з середовищем, тоді як сама властивість sample efficiency передбачає навчання агентів на якомога меншій кількості даних. Ідея model-based підходу формулюється наступним чином: замість того, щоб постійно взаємодіяти з середовищем для тестування нових дій під час навчання агента, потрібно збирати дані з середовища, а саме в який стан агент переходить, роблячи дії в поточному стані. Враховуючи зазначені дані можна запустити окремий процес навчання з передбаченням за станом та дією агента для підвищення ефективності навчання з допомогою нейронної мережі. Підходи [1], які використовують цю техніку, зазвичай називають методами комунікації, оскільки повна автономія агентів скомпрометована для кращої продуктивності.

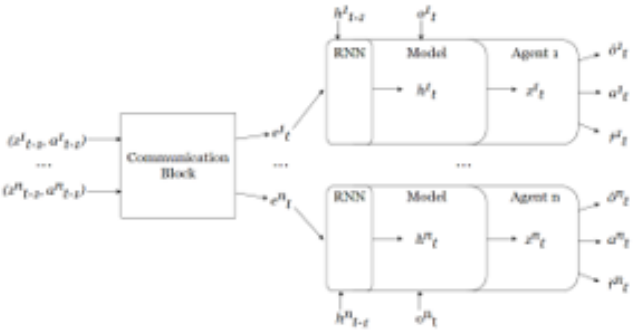


Рисунок 1 - Етапи мультиагентного навчання НМ

На Рис.1 ви можете побачити, що у нас є латентні стани z та дії агентів a. Ці вектори ми передаємо до Communication Block, який є звичайним стеком Attention, який у свою чергу, видає вектори e для кожного агента, яким він може оновити свою модель середовища. Для цього може використовуватись рекурентна нейронна мережа, що переводить прихований стан h на поточний крок, а вже з цього стану та спостереження агента o, передбачається поточний латентний стан z у кожного агента. Було розглянуто особливості підходу навчання з підкріпленням на основі моделей в мультиагентних середовищах. Реалізація підходу викликала низку проблем, як з практичної, так і з теоретичної точки зору. Виходячи з отриманих результатів, можна зробити висновок, що застосування цього підходу має значний потенціал, він може зробити дослідження в галузі навчання нейронних мереж більш доступним і ефективним.

Література

1. Baker B., Markov T., McGrew B. - Emergent tool use from multi-agent interaction - [Online resource] - Access mode: https://openai.com/research/emergent-tool-use

Б.8 Публікація статті в науково фаховому виданні «Електронне моделювання» (кат. Б)

ОБЧИСЛЮВАЛЬНІ ПРОЦЕСИ ТА СИСТЕМИ
COMPUTATIONAL PROCESSES AND SYSTEMS

DOI: https://doi.org/10.15407/emodel.46.01.075
УДК 004.627

Я.М. Крайник, канд. тех. наук, **Д.В. Доценко**
Чорноморський національний університет ім. Петра Могили
Україна, 54003, Миколаїв, вул. 68 Десантників, 10
e-mail: yaroslav.krainyk@chmnu.edu.ua, dotsenko.d@chmnu.edu.ua

Метод стиснення зображень з використанням попередньої обробки та алгоритмів Quite Ok Image і Хаффмана

Представлено метод стиснення зображень, який базується на комбінованому підході з використанням попередньої обробки зображень та алгоритму Хаффмана. Запропоновано організацію циклу обробки відповідно до цього методу та наведено експериментальні результати роботи методу на тестовому наборі зображень. Комбінований підхід дозволяє досягти кращих результатів у порівнянні з прямим використанням одного з методів. У ході процесу стиснення виконується перетворення у відповідний кольоровий формат. Далі відбувається розділення зображення на 2 блоки даних, основну область та область вузлових точок. До них застосовуються різні методи стиснення з урахуванням структури даних. Проведено порівняння використання алгоритму Хаффмана до області пікселів без додаткових перетворень та з представленням у вигляді різниці, яке показало, що представлення у вигляді різниці дозволяє уніфікувати розподіл значень для стиснення та покращити результат стиснення. Результати обох процесів комбінуються та передаються далі або для передачі на інші пристрої або для збереження інформації у стисненому стані. Запропонований метод стиснення може бути використаний при реалізації нових форматів стиснення у вбудованих системах, які мають обмежені обчислювальні потужності, але потребують роботи з графічними даними.

Ключові слова: зображення, стиснення, алгоритм Хаффмана, вузлові точки, ефективність, якість, порівняння, алгоритм QOI.

Сучасні цифрові технології надають великі можливості для мультимедіа представлення і відзначаються широким використанням в розробці різних пристроїв з високопродуктивними можливостями відображення контенту. Одним з ключових аспектів цього є робота з зображеннями, які стали необхідною частиною як повсякденного життя, так і професійної сфери. Зображення допомагають забезпечити високу точність та наочність в різних галузях, починаючи від медицини, де вони використо-

ЕЛЕКТРОННЕ МОДЕЛЮВАННЯ

UK EN

+38 (044) 424-14-66 em@ipme.kiev.ua

ПРО ЖУРНАЛ АРХІВ НОМЕРІВ ДЛЯ АВТОРІВ РЕДКОЛЕГІЯ КОНТАКТИ

Головна / АРХІВ НОМЕРІВ / 2024 рік / Том 46, № 2 (2024)

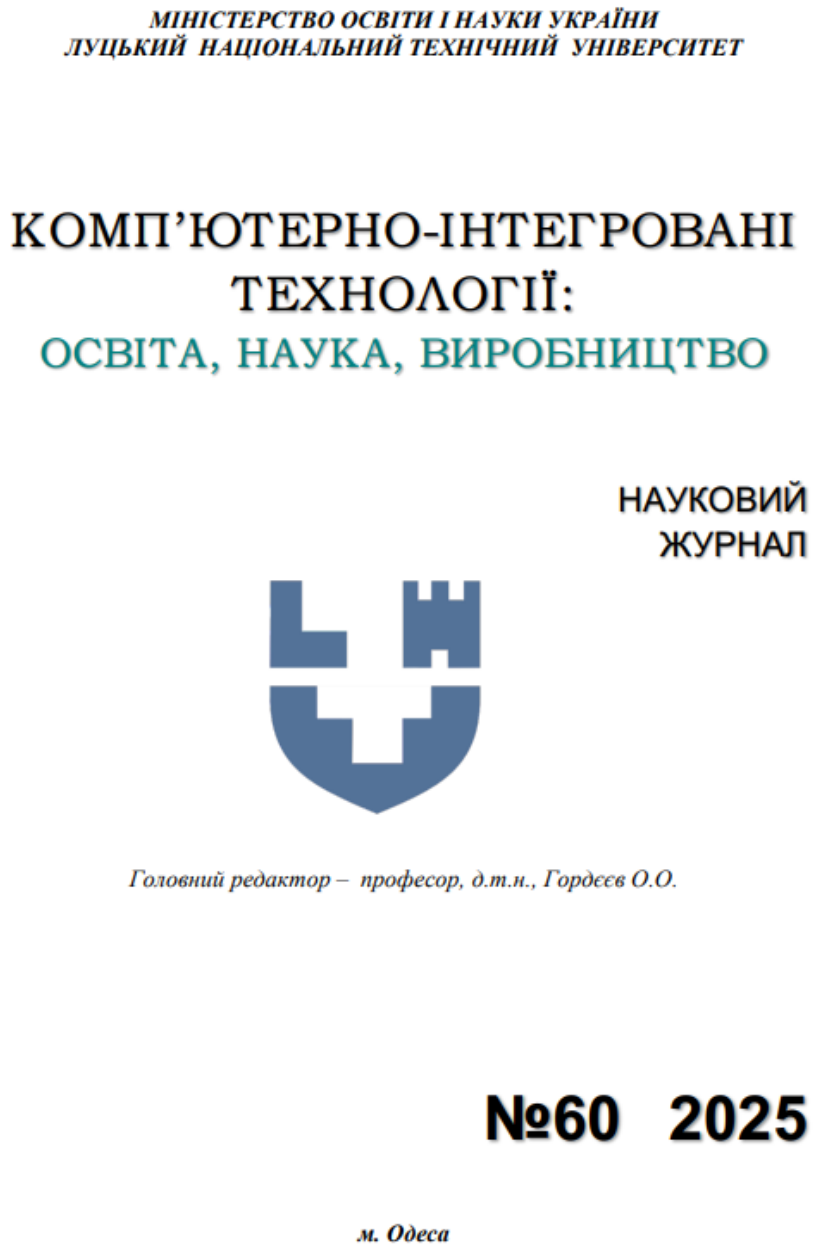
Електронне моделювання

Том 46, №2 (2024)

https://doi.org/10.15407/emodel.46.02

ЗМІСТ

Б.9 Публікація статті в науковому фаховому виданні «Комп’ютерно-інтегровані технології: освіта, наука, виробництво» (кат. Б)



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

КОМП’ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ:
ОСВІТА, НАУКА, ВИРОБНИЦТВО

НАУКОВИЙ ЖУРНАЛ

Головний редактор – професор, д.т.н., Гордєєв О.О.

№60 2025

м. Одеса

78 *Науковий журнал* "Комп’ютерно-інтегровані технології: освіта, наука, виробництво"
Луцьк, 2025. Випуск № 60

DOI: https://doi.org/10.36910/6775-2524-0560-2025-60-08
УДК 004.73:621.396.946
Доценко Дмитро Володимирович, магістрант з комп’ютерної інженерії
https://orcid.org/0009-0008-6689-6440
Пузирьов Сергій Володимирович, канд. фіз.-мат. наук, доцент
https://orcid.org/0000-0003-0520-1496
Чорноморський національний університет імені Петра Могили, м. Миколаїв, Україна

ПОБУДОВА ТЕПЛОВОЇ КАРТИ Wi-Fi ЗА ДОПОМОГОЮ IOT-МОДУЛЯ ESP8266

Доценко Д. В., Пузирьов С. В. Побудова теплової карти покриття Wi-Fi за допомогою IoT-модуля ESP8266.
У статті запропоновано концепт апаратно-програмного комплексу для побудування теплової карти Wi-Fi на базі IoT-модуля ESP8266 з вбудованим WiFi-модулем. На основі аналізу існуючих публікацій та відкритих проєктів було виявлено, що існуючі інструменти та рішення не дозволяють з достатньою точністю будувати теплові карти WiFi. Також важливою проблемою є наявність артефактів та хибних зон, які потрібно відфільтрувати. Розроблено методику побудови теплової карти WiFi-покриття в приміщенні на основі RSSI бездротової мережі. Сканування бездротових каналів включає пошук за SSID з опціональною фіксацією BSSID (щоб уникнути стрибків між радіомодулями з однаковою назвою мережі). Початкове положення визначається як найближча позиція до точки доступу і може бути довільною. Після фіксації положення запускається сканування приміщення за квадратною спіраллю з фіксованим кроком. Отримані дані зберігаються у форматі CSV через Serial-інтерфейс і одночасно передаються на сервер за протоколом MQTT у форматі JSON (формати можуть бути довільними). Візуалізація даних здійснюється за допомогою Python та відповідних бібліотек (NumPy/SciPy/Matplotlib) методом griddata із маскуванням країв. Передбачена побудова окремих карт для кожного BSSID. В результаті проведених експериментів було протестовано методику та апаратно-програмний комплекс для збору даних теплової карти. Виявлено досить суттєвий вплив на рівень та якість сигналу наявність сторонніх людей та предметів, особливо металічних саме поблизу точки доступу. Найбільш якісний результат було отримано по скануванню BSSID.
Ключові слова: бездротова мережа, моніторинг, IoT-пристрої, обмежене енергозабезпечення, теплова карта, Wi-Fi, ESP8266, RSSI, MQTT, griddata, BSSID/SSID, Python, NumPy, SciPy, Matplotlib, griddata

Dotsenko D., Puzyrov S. Construction of Wi-Fi coverage heat map using the ESP8266 IoT module.
The article proposes a concept of a hardware-software solution to build the Wi-Fi heat map based on the IoT-module ESP8266. Based on the analysis of existing publications and open projects, it was found that existing tools and solutions do not allow building WiFi heat maps with sufficient accuracy and artifact level. There was created a method to build an indoor WiFi heat map coverage based on the RSSI of a wireless network. Scanning wireless channels includes a search by SSID and optionally by BSSID to avoid hopping between radio modules with the same network. The initial position is defined as the closest position to the AP and can be arbitrary. After positioning fix the room scanning is started with the fixed step square spiral . The obtained data is stored in CSV format and simultaneously transmitted to the server as JSON via MQTT. Data visualization uses Python and NumPy/SciPy/Matplotlib with the edge masking griddata method. Also it is possible to build a BSSID WiFi heat map. The method and hardware-software solution were tested successfully to build a heat map. The influence on the heat map is significant from outsiders and metal objects. The highest quality result was obtained by BSSID scanning.
Keywords: wireless network, monitoring, IoT devices, limited power supply, heat map, Wi-Fi, ESP8266, RSSI, MQTT, griddata, BSSID/SSID, Python, NumPy, SciPy, Matplotlib, griddata.

Постановка наукової проблеми. Сучасне суспільство все більше залежить від якісного та стабільного бездротового доступу до Інтернету. Наявність Wi-Fi покриття у житлових приміщеннях, навчальних закладах, офісах та громадських місцях є необхідною умовою ефективної роботи інформаційних систем. Проте на практиці рівень сигналу Wi-Fi є нерівномірним через архітектурні особливості будівель, наявність перешкод, розташування маршрутизаторів та вплив зовнішніх чинників. Це призводить до «мертвих зон», де з’єднання нестабільне або відсутнє, що ускладнює користування мережевими сервісами.

Існуючі професійні рішення для аналізу WiFi-покриття (heatmap tools) здебільшого є