НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету інформаційних технологій	**Завідувач кафедри комп'ютерних наук**
_____ / Ігор Болбот /	_____ / Белла Голуб /
(підпис) (ПІБ)	(підпис) (ПІБ)
«___»_____2025 р.	«___»_____2025 р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему Програмне забезпечення системи аналізу можливостей та ефективності форматів файлів растрових зображень

Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)

Освітня програма Програмне забезпечення інформаційних систем
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

к.ф.-м.н., доц. / Віктор Кириченко /
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник магістерської кваліфікаційної роботи

ст. викл. / Георгій Бородкін /
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Науковий консультант магістерської кваліфікаційної роботи

к.т.н., доц. / Юлія Боярінова /
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ / Іван Юрченко /
(підпис) (ПІБ)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

доцент, к.т.н. _____ Голуб Б.Л.
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"_10_" листопада 2024 року

ЗАВДАННЯ

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

Юрченко Івану Сергійовичу
(прізвище, ім'я, по батькові)

Спеціальність 121 "Інженерія програмного забезпечення"
(код і найменування)

Освітня програма Програмне забезпечення інформаційних систем
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи Програмне забезпечення системи аналізу можливостей та ефективності форматів файлів растрових зображень

затверджена наказом проректора НУБіП України від "1" листопада 2024 р. № 1963 "С"

Термін подання завершеної роботи на кафедру 14 листопада 2025 р.
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи набори даних високоякісних растрових зображень різного типу та роздільної здатності у відкритому доступі

Перелік питань, що підлягають дослідженню:
1. Дослідити ключові технічні особливості форматів файлів растрових зображень
2. Провести кодування набору зображень використовуючи актуальні растрові формати задля отримання продуктивних та якісних метрик
3. Проаналізувати отримані результати для порівняння ефективності розглянутих форматів
4. Розробити та реалізувати графічні ілюстративні матеріали (за потреби)

Дата видачі завдання "7" листопада 2024 р.

Керівник магістерської кваліфікаційної роботи _____ Бородкін Г. О.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Юрченко І.С.
(підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний цифровий простір характеризується експоненційним зростанням обсягів візуального контенту, що створює значне навантаження на інфраструктуру зберігання та передачі даних. Традиційні формати растрових зображень, такі як JPEG та PNG, попри їхню поширеність, дедалі рідше відповідають актуальним вимогам до ефективності стиснення. Використання застарілих алгоритмів стиснення спричиняє витрати дискового простору та збільшене навантаження на пропускну здатність мереж.

Попри появу досконаліших кодеків (AVIF, JPEG XL), їх впровадження відбувається повільно. Це зумовлено інерцією ринку та відсутністю доступних інструментів, які б дозволили розробникам і дизайнерам об'єктивно оцінити переваги нових стандартів. Як наслідок, потенціал сучасних технологій залишається нереалізованим, що призводить до суттєвого зниження продуктивності веб-ресурсів та погіршення користувацького досвіду через ігнорування можливостей оптимізації.

Зазначені фактори актуалізують потребу у проведенні комплексного аналізу ефективності сучасних форматів та створенні спеціалізованого програмного забезпечення, що дозволить користувачам здійснювати обґрунтований вибір формату на основі об'єктивних метрик.

Метою магістерської роботи є проведення комплексного порівняльного аналізу продуктивності та можливостей провідних форматів растрових зображень та розробка програмного засобу, що слугуватиме інструментом для об'єктивного оцінювання їх ефективності та вибору оптимального формату залежно від сценарію використання.

Для досягнення поставленої мети було визначено наступні завдання:

1. Проаналізувати теоретичні основи та алгоритмічні принципи, що лежать в основі класичних та сучасних форматів растрових зображень;

2. Провести системний аналіз функціональних можливостей, переваг та обмежень форматів BMP, GIF, JPEG, PNG, JPEG 2000, WebP, HEIC, AVIF та JPEG XL;
3. Розробити концептуальну архітектуру програмного засобу для аналізу форматів, що забезпечує гнучкість, модульність та можливість інтеграції зовнішніх інструментів кодування;
4. Реалізувати програмний засіб з інтуїтивним графічним інтерфейсом, що дозволяє виконувати одночасне кодування зображень у декілька форматів та розраховувати ключові метрики ефективності;
5. Провести комплексне експериментальне дослідження ефективності форматів PNG, JPEG, WebP, AVIF та JPEG XL на репрезентативних наборах даних;
6. Проаналізувати отримані експериментальні дані та сформулювати практичні рекомендації щодо вибору оптимального формату для різних типів візуального контенту та технічних завдань.

Об'єктом дослідження є властивості, функціональні можливості та характеристики продуктивності сучасних форматів файлів растрових зображень.

Предметом дослідження є методи та метрики оцінки ефективності алгоритмів стиснення, зокрема співвідношення ступеня компресії, оцінки візуальної якості та часу кодування; архітектура та програмна реалізація системи для автоматизованого порівняльного аналізу форматів.

Методи дослідження. Для огляду технологій стиснення застосовуються методи системного та порівняльного аналізу; для розробки системи: методи об'єктно-орієнтованого проєктування та UML-моделювання; для розробки архітектури системи: методи бізнес-аналізу та методи оцінки якості програмних систем; для отримання емпіричних даних про продуктивність кодеків: засоби експериментальних досліджень; для аналізу результатів експерименту: методи статистичної обробки та засоби візуалізації даних.

1 СИСТЕМНИЙ АНАЛІЗ ТЕХНОЛОГІЙ СТИСНЕННЯ РАСТРОВИХ ЗОБРАЖЕНЬ

1.1. Огляд поширених форматів растрових зображень

Історія форматів цифрових зображень тісно пов'язана з розвитком обчислювальної техніки та комунікаційних мереж. Перші формати були розроблені в умовах жорстких обмежень апаратного забезпечення, що диктувало прості, часто нестиснені, способи зберігання піксельних даних. Зі зростанням обчислювальних ресурсів потужності процесорів, обсягів пам'яті та, найголовніше, з появою Інтернету, виникла потреба в ефективних методах стиснення для зменшення розміру файлів. Цей підрозділ розглядає хронологічний перелік растрових форматів, від ранніх стандартів, до сучасних форматів нового покоління. Буде проведено їх класифікацію за ключовими критеріями, що дозволить систематизувати знання про їхнє призначення та функціональні можливості.

1.1.1. BMP (Bitmap Image File). BMP (також відомий як Device Independent Bitmap, DIB) – це формат растрової графіки, переважно пов'язаний з операційними системами Microsoft Windows. Формат був представлений Microsoft у 1987 році разом з першими версіями Windows. Він був розроблений як простий, незалежний від пристроїв спосіб зберігання цифрових зображень, які могли відображатися однаково на різних апаратних конфігураціях. У той час, коли багато форматів зображень були прив'язані до конкретних пристроїв або адаптерів дисплея, формат BMP відрізнявся тим, що забезпечував послідовне представлення растрових зображень у графічній підсистемі Windows (GDI).

Технічні особливості

Стиснення: Формат має дуже обмежені можливості стиснення, тому зазвичай зберігає дані пікселів лінійно без змін. Пізніші версії додали підтримку RLE (Run-Length Encoding), але лише для зображень з індексованими палітрами

у форматі 4-біт та 8-біт. Вбудованої підтримки стиснення з втратами формат не має.

Модель та глибина кольору: BMP підтримує як індексовані кольори, так і True Color. Піксельний формат залежить від обраної кількості бітів на піксель:

- 1-біт: 2 можливих кольори, вказані у палітрі;
- 4-біт: 16 можливих кольорів, вказані у палітрі;
- 8-біт: 256 можливих кольорів, вказані у палітрі;
- 16-біт: 65’536 можливих кольорів, зазвичай збережні у форматі RGB555 або RGB565;
- 24-біт: 16’777’216 можливих кольорів у форматі RGB888, по 8 байт на канал;
- 32-біт: 4’294’967’296 можливих кольори у форматі RGBA8888, по 8 байт на канал включно з прозорістю.

Тобто якщо глибина кольору становить 8-біт або менше, то у файл записується палітра налічуючи від 2 до 2^n кольорів (де n – кількість біт), а пікселі зображення зберігаються у вигляді індексів кольорів палітри. Якщо глибина кольору становить 16-біт або більше, то значення колірних каналів пікселів записується напряму (після відсічення та квантування за потреби).

Підтримка прозорості: Формат має повну підтримку прозорості у 32-бітному піксельному форматі. Специфікація формату має опціональні маски кольорових каналів у заголовку файлу, що дозволяють визначити довільні призначення бітів пікселя до каналу, але хоча це й дозволяє реалізувати альфа канал для піксельних форматів 24-біт та 16-біт, практично такі формати використовувались рідко та не мають поширеної підтримки у графічних редакторах.

Структура файлу та метадані. Файл BMP складається з кількох блоків:

- Заголовок файлу – ідентифікує файл як BMP, зберігає його загальний розмір та зміщення до початку піксельних даних;
- Заголовок DIB – визначає розміри, глибину кольору, тип стиснення та іншу інформацію про колір;
- Таблиця кольорів – за необхідності, використовується в зображеннях з індексованими кольорами;
- Масив пікселів – безпосередньо растрові дані можуть бути стисненими або нестисненими;
- Додаткові бітові маски каналів та профілі кольору.

Підтримка метаданих мінімальна, обмежується технічними параметрами та, за необхідності, вбудованими колірними профілями ICC. Для BMP не існує стандарту розширених метаданих, подібного до EXIF.

Прийняття та поточна актуальність. Формат BMP швидко став основним для зображень у програмах Windows наприкінці 1980-х та протягом 1990-х років. Завдяки простоті та легкості декодування він широко використовувався для іконок, шпалер та графіки у програмному забезпеченні. Однак через відсутність ефективного стиснення та обмежений набір функцій, він виявився непрактичним для збереження фотографій або використання в інтернеті. Згодом такі формати як JPEG, PNG та GIF, витіснили його для більшості повсякденних потреб.

Сьогодні, хоча більшість переглядачів та графічних редакторів все ще підтримують основну специфікацію BMP, практично формат застосовується рідко, за винятком застарілих систем, низькорівневого програмування під Windows або простих тестових завдань. Його історична роль була значною, але попит значно знизився на користь растрових форматів, які є більш ефективними у зберіганні даних та мають ширший функціонал.

1.1.2. GIF (Graphics Interchange Format). GIF (Graphics Interchange Format) – це растровий графічний формат, розроблений командою розробників американського провайдера CompuServe на чолі зі Стівом Вілхайтом у 1987 році. Його метою було створення формату для передачі кольорових зображень через повільні мережеві з'єднання. GIF швидко здобув популярність завдяки ефективному (на той час) стисненню без втрат та унікальній можливості зберігати кілька зображень в одному файлі, що стало основою для ранньої веб-анімації.

Технічні особливості

Стиснення: Формат використовує алгоритм стиснення LZW (Lempel-Ziv-Welch), який створює словник повторюваних послідовностей інформації та замінює їх коротким кодом із словника, тим самим зменшуючи розмір файлу без втрат.

Модель та глибина кольору: Формат GIF використовує виключно індексовану палітру кольорів. Кожен файл містить палітру налічуючи до 256 унікальних кольорів, обраних із 24-бітного простору RGB (16.7 мільйонів кольорів). Кожен піксель у зображенні зберігає лише індекс кольору у цій палітрі. Це робить формат непридатним для фотографій з плавними градієнтами, але ідеальним для простої графіки з обмеженою кількістю кольорів.

Підтримка прозорості: GIF підтримує бінарну прозорість. Це означає, що один із 256 кольорів у палітрі можна визначити як повністю прозорий. Часткова прозорість (окремий альфа-канал), яка дозволяє створювати напівпрозорі ефекти, форматом GIF не підтримується.

Анімація: Це ключова особливість, яка забезпечила формату GIF довготривалу актуальність. Формат дозволяє зберігати послідовність кадрів (окремих зображень) в одному файлі разом з інформацією про тривалість відображення кожного кадру. Веб-браузери та переглядачі зображень автоматично відтворюють ці кадри по черзі, створюючи ефект анімації.

Структура файлу та метадані. Файл GIF має блокову структуру, яка включає:

- Заголовок файлу – ідентифікує файл як GIF та вказує версію (87a або 89a);
- Логічний дескриптор екрана (Logical Screen Descriptor) – визначає розміри зображення та наявність глобальної таблиці кольорів;
- Глобальна таблиця кольорів (Global Color Table) – палітра, спільна для всіх кадрів, (необов'язкова).

Для кожного кадру зображення додаються повторювані блоки:

- Розширення керування графіки (Graphic Control Extension) – тип блоку розширення, визначає параметри кожного кадру, такі як тривалість кадру в анімації (визначена в сотих секунди);
- Дескриптор зображення – визначає позицію та розмір зображення на логічному екрані, а також може містити локальну палітру кадру;
- Блоки даних зображення – кожен кадр може мати власний дескриптор, локальну таблицю кольорів та стиснені піксельні дані.

Підтримка метаданих у GIF дуже обмежена. Окрім GCE (Graphic Control Extension), формат також визначає такий блок розширення як Comment Extension, в якому можна зберегти додаткову інформацію як простий текст. Блок розширення Application Extension може використовуватись програмами редакторами для збереження власної інформації за необхідності. За допомогою таких блоків є можливість включити дані формату XMP (Extensible Metadata Platform) у файлі зображення, проте офіційної підтримки ані XMP, ні EXIF, формат GIF не має.

Прийняття та поточна актуальність. У 1990-х та на початку 2000-х років GIF був одним із основних форматів в Інтернеті завдяки своїй універсальній підтримці браузерами та малим розмірам файлів. Однак для статичних зображень його значною мірою витіснили JPEG та PNG, адже вони пропонують краще стиснення та якість кольору.

Незважаючи на це, GIF зберіг свою нішу завдяки підтримці анімації. Сьогодні він досі є де-факто стандартом для коротких, зациклених, беззвучних анімацій у соціальних мережах, месенджерах та на веб-сайтах. Хоча існують технічно досконаліші формати для анімації (APNG, WebP, WebM), простота створення та універсальна підтримка забезпечують GIF стабільну популярність. Його роль перетворилася з універсального формату зображень в інструмент для створення анімованого візуального контенту.

1.1.3. JPEG (Joint Photographic Experts Group). JPEG (Joint Photographic Experts Group) – формат цифрових зображень що використовує метод стиснення на основі дискретного косинусного перетворення. Розроблений об'єднаною групою експертів з фотографії та офіційно представлений у 1992 році, його основною метою було створення ефективного методу стиснення деталізованих зображень, таких як фотографії, зі значним зменшенням розміру файлу ціною певної втрати якості.

Технічні особливості

Стиснення: JPEG використовує алгоритм стиснення з втратами, який є його визначальною характеристикою. Процес стиснення складається з кількох етапів:

- Перетворення колірного простору: Зображення перетворюється з RGB у колірний простір YCbCr, який поділяє інформацію про яскравість (luma, Y) та колір (chroma, Cb та Cr);
- Зниження колірної дискретизації (Chroma Subsampling): Оскільки людське око менш чутливе до змін кольору, ніж до змін яскравості, для зменшення розміру файлу роздільна здатність колірної інформації може бути зменшена у 2 або 4 рази;
- Дискретне косинусне перетворення (DCT): Зображення розбивається на блоки у 8x8 пікселів, і до кожного блоку застосовується DCT, яке перетворює просторові дані в частотні коефіцієнти;

- Квантування: Це ключовий етап, де відбувається втрата даних. Коефіцієнти DCT діляться на значення таблиці квантування та округлюються. Високочастотні компоненти, які менш помітні для ока, квантуються більш агресивно. Кількість втрат при квантуванні безпосередньо контролюється рівнем якості (від 0 до 100);
- Ентропійне кодування: Отримані коефіцієнти стискаються без втрат за допомогою RLE (Run-Length Encoding), після чого повторно стискаються за допомогою кодування Хаффмана.

Хоча стандарт JPEG визначає режим стиснення без втрат, він не набув поширення та не підтримується більшістю програмного забезпечення.

Модель та глибина кольору: Стандартний JPEG підтримує 8 біт на колірний канал, що забезпечує 24-бітну глибину кольору (True Color), або 16.7 мільйонів кольорів. Формат також підтримує 8-бітні зображення у відтінках сірого.

Підтримка прозорості: Формат JPEG не зберігає інформації про альфа-канал, тому підтримка прозорості повністю відсутня. При збереженні прозорих зображень у JPEG прозорі області замінюються суцільним кольором (зазвичай білим).

Прогресивне завантаження: Окрім стандартного режиму завантаження, де дані зображення поступають лінійно рядок за рядком, JPEG підтримує прогресивний режим. У такому режимі спочатку завантажується низькоякісна версія повного зображення, яка поступово деталізується в міру надходження даних. Це було особливо корисно в епоху повільного Інтернету.

Структура файлу та метадані. Файл JPEG (зазвичай у контейнері JFIF або EXIF) складається із послідовності сегментів позначених ідентифікованими маркерами:

- Заголовок файлу (SOI – Start Of Image) – початкова сигнатура файлу, яка ідентифікує його як JPEG;
- APPx (де x – число від 0 до 15) – сегмент метаданих (наприклад, інформація у форматі JFIF або Exif);
- COM – поле для простих текстових коментарів.
- DQT (Define Quantization Table) – таблиці квантування, що використовуються при стисканні;
- DHT (Define Huffman Table) – таблиці Хаффмана для ентропійного кодування;
- SOF (Start Of Frame) – інформація про розмір зображення, кількість компонентів (наприклад, YCbCr) та глибину кольору;
- SOS (Start Of Scan) – вказує на початок закодованих піксельних даних.
- Стиснені піксельні дані – основна частина зображення, закодована методом JPEG;
- Кінцевий маркер (EOI – End Of Image) – позначає завершення основних даних файлу.

Формат JPEG має розширену підтримку метаданих, перш за все через стандарт EXIF (Exchangeable Image File Format). Майже кожна цифрова камера зберігає фотографії у форматі JPEG з вбудованим блоком EXIF, який може містити:

- Технічні параметри зйомки (модель камери, витримку, діафрагму, ISO, фокусну відстань);
- Дату та час створення фотографії;
- GPS-координати місця зйомки;
- Невелике зображення-мініатюра для попереднього перегляду.

Окрім EXIF, JPEG може містити метадані стандартів IPTC та XMP, що використовуються для додавання описів, ключових слів та інформації про авторське право.

Прийняття та поточна актуальність. JPEG став абсолютним стандартом для цифрової фотографії та веб-графіки. Здатність значно зменшувати розмір файлів зробила можливою передачу фотографій через Інтернет та їх зберігання на пристроях з обмеженою пам'яттю. JPEG є форматом за замовчуванням для переважної більшості цифрових камер, смартфонів та веб-сайтів.

Сьогодні, незважаючи на появу більш сучасних форматів, таких як WebP, AVIF та HEIC, які пропонують кращий ступінь стиснення за тої ж візуальної якості, JPEG залишається найпоширенішим форматом для фотографій завдяки своїй універсальній сумісності. Практично будь-яке прикладне та апаратне програмне забезпечення, що працює із зображеннями, може відкрити файл JPEG. Однак його недоліки, такі як помітні артефакти стиснення (блочність, шум) за низьких налаштувань якості та відсутність підтримки прозорості, роблять його непридатним для іконок, логотипів та іншої графіки, що вимагає чітких ліній.

1.1.4. PNG (Portable Network Graphics). PNG (Portable Network Graphics) – растровий графічний формат, що використовує стиснення без втрат. Розроблений у середині 1990-х років як відкрита та вільна від патентів альтернатива формату GIF, PNG був створений для вирішення його технічних обмежень, зокрема, щодо глибини кольору та підтримки прозорості. Формат був офіційно затверджений як стандарт W3C у 1996 році і швидко став основним для передачі якісної графіки через Інтернет.

Технічні особливості

Стиснення: PNG використовує стиснення без втрат на основі алгоритму DEFLATE, який є комбінацією алгоритму LZ77 та кодування Хаффмана. Перед стисненням до даних застосовується процес фільтрації, який перетворює дані пікселів таким чином, щоб збільшити ефективність їх стиснення. Для кожного рядка пікселів обирається тип фільтру (наприклад, Sub, Up, Average) замінюють абсолютні значення пікселів на різницю між сусідніми, що створює більш однорідні та повторювані послідовності даних, ідеальні для DEFLATE. Це

дозволяє досягти значного зменшення розміру файлу без будь-якої втрати якості зображення.

Модель та глибина кольору: PNG підтримує широкий вибір типу кольору та кількості біт на канал:

- Індексована палітра (PNG-8): Зберігає індекси кольорів з палітри, обраних з 24-бітного простору. Зображення може використовувати від 1 до 8 біт на піксель, тобто від 2 до 256 унікальних кольорів;
- Відтінки сірого: Монохромне зображення де кожен піксель може займати від 1 до 16 біт, тобто від 2 до 65536 відтінків сірого. Також є тип відтінків сірого з прозорістю, що підтримує 16 та 32 біт на піксель, порівну поділені на два канали;
- True Color (PNG-24): Підтримує 24-бітний колір (по 8 біт на канали R, G, B), що забезпечує 16.7 мільйонів кольорів;
- True Color з альфа-каналом (PNG-32): Додає 8-бітний альфа-канал до 24-бітного кольору, що забезпечує повну підтримку прозорості;
- Висока глибина кольору: Формат також підтримує 48-бітний RGB та 64-бітний RGBA, де кожний канал має глибину в 16 біт.

Підтримка прозорості: Формат підтримує повноцінний 8-бітний альфа-канал. Можливість створення плавних переходів та напівпрозорих ефектів, разом з ефективним стисненням, є значною перевагою над бінарною прозорістю GIF та повною її відсутністю у JPEG.

Анімація: Офіційний стандарт PNG історично не підтримував анімацію. Проте, існує розширення формату APNG (Animated PNG), публіковане розробниками з Mozilla у 2008 році, що було офіційно включено до специфікації PNG (Третьої редакції) лише у 2024 році. APNG дозволяє зберігати послідовність кадрів, та є зворотно сумісним, відображаючи перший кадр в програмах що не підтримують цей формат. Хоча APNG довгий час не був частиною офіційної

специфікації PNG, його підтримка рушіями Gecko та Chromium забезпечили сумісність з більшістю популярних браузерів.

Структура файлу та метадані. Файл PNG складається з 8-байтової сигнатури яка ідентифікує файл, після якої слідує послідовність блоків файлу. Кожен блок має назву, дані та контрольну суму. Блоки поділяються на критичні та допоміжні, що дозволяють розширювати формат зберігаючи сумісність.

Основні критичні блоки:

- IHDR (Image Header) – перший блок у файлі, містить інформацію про розміри зображення, глибину кольору, метод стиснення та фільтрування, а також наявність черезрядкової розгортки;
- PLTE (Palette) – містить таблицю кольорів для зображень з індексованою палітрою;
- IDAT (Image Data) – містить стиснені дані пікселів зображення. Зображення може бути розбите на декілька таких блоків;
- IEND (Image End) – маркер, що позначає кінець файлу.

PNG має гнучку систему для зберігання метаданих у допоміжних блоках, таких як tEXt (текстові дані у форматі ключ-значення), dSIG (цифрові підписи), gAMA (інформація про гамма-корекцію) та iCCP (вбудований колірний профіль ICC). Це дозволяє зберігати інформацію про автора, опис зображення та параметри для коректного відображення кольорів на різних пристроях. Також можливе вбудовування метаданих формату EXIF та XMP (у блоках eXIf та iTXt відповідно).

Прийняття та поточна актуальність. PNG став де-факто стандартом для веб-графіки, яка вимагає чіткості та прозорості. Він ідеально підходить для логотипів, іконок, діаграм, скріншотів та будь-яких зображень з текстом або чіткими межами, де артефакти стиснення JPEG є неприйнятними. PNG, використовуючи стиснення без втрат, також не втрачає якості за редагування та повторного збереження.

Сьогодні PNG досі є одним з найпоширеніших форматів зображень для розповсюдження в Інтернеті та графічного дизайну. Його універсальна підтримка браузерами та графічними редакторами, а також висока якість зображення та підтримка прозорості, забезпечують актуальність формату, співіснуючи із JPEG та GIF.

1.1.5. JPEG 2000. JPEG 2000 – це стандарт стиснення растрових зображень, розроблений комітетом Joint Photographic Experts Group наприкінці 1990-х років і фіналізований у 2000 році. Його метою було створення наступника популярного формату JPEG, який би пропонував вищу якість зображення при кращому ступені стиснення та мав ширший набір функцій. На відміну від свого попередника, що базується на дискретному косинусному перетворенні (DCT), JPEG 2000 використовує технологію дискретного вейвлет-перетворення (DWT).

Технічні особливості

Стиснення: Основою JPEG 2000 є вейвлет-перетворення, яке аналізує зображення в цілому, а не розбиває його на блоки 8x8 пікселів, як це робить JPEG. Цей підхід дозволяє уникнути характерних для JPEG блочних та кільцевих артефактів стиснення, замінюючи їх більш плавним розмиттям при високих коефіцієнтах стиснення. Формат підтримує як стиснення з втратами, так і стиснення без втрат в рамках єдиного кодового потоку. Це досягається завдяки використанню оборотних цілочисельних вейвлет-трансформацій для режиму без втрат. Завдяки цьому, JPEG 2000 забезпечує на 20-30% краще стиснення, ніж JPEG, за тієї ж самої візуальній якості.

Модель та глибина кольору: JPEG 2000 має гнучку підтримку кольорових моделей. На відміну від JPEG, який зазвичай обмежується 8-бітним представленням RGB, JPEG 2000 підтримує глибину кольору до 38 біт на компонент та може обробляти до 16384 компонентів (каналів) в одному файлі. Це робить його ідеальним для професійної фотографії, медичних зображень

(наприклад, рентгенівських знімків) та інших застосувань, що вимагають високої точності передачі кольору та динамічного діапазону.

Підтримка прозорості: Формат повністю підтримує альфа-канал для створення зображень з різними рівнями прозорості.

Додаткові можливості: JPEG 2000 впровадив низку інноваційних функцій, відсутніх у його попередника:

Масштабованість: З одного файлу JPEG 2000 можна отримати зображення різної роздільної здатності та якості без необхідності перекодування.

Області інтересу (Region of Interest, ROI): Формат дозволяє визначати певні ділянки зображення як важливіші та кодувати їх з вищою якістю, ніж решту зображення, оптимізуючи таким чином розмір файлу.

Стійкість до помилок: Структура кодового потоку робить JPEG 2000 більш стійким до помилок при передачі даних у порівнянні з JPEG.

Структура файлу та метадані. Файл JPEG 2000 (зазвичай з розширенням .jp2 або .j2k) має складну структуру, що базується на концепції "боксів" (boxes), подібну до формату MP4. Основні частини файлу включають:

- Сигнатура JPEG 2000: Ідентифікує файл як JP2;
- Заголовок файлу (File Type Box): Визначає специфікації, яким відповідає файл;
- Заголовок зображення (Image Header Box): Містить основну інформацію про зображення, таку як висота, ширина, кількість компонентів та глибина кольору;
- Кодовий потік (Contiguous Codestream Box): Містить власне стиснені дані зображення, організовані в пакети, що дозволяє гнучке декодування;
- Формат має розширену підтримку метаданих. Він може включати стандартні метадані, такі як EXIF та XMP, а також геопросторову

інформацію (GeoJP2) та інші специфічні дані, що робить його популярним у картографії та архівній справі.

Прийняття та поточна актуальність. Незважаючи на свої значні технічні переваги над JPEG, формат JPEG 2000 не зміг завоювати масову популярність і витіснити свого попередника. Алгоритми на основі вейвлет-перетворення вимагали значно більше обчислювальних ресурсів (пам'яті та потужності процесора), ніж JPEG, що було суттєвою перешкодою на момент його появи на початку 2000-х років. JPEG 2000 також не був зворотно сумісним з JPEG, що вимагало від розробників програмного забезпечення імплементувати абсолютно новий кодек. Найбільшою перешкодою для поширення в Інтернеті стала відсутність нативної підтримки у більшості веб-браузерів, на сьогодні лише Safari підтримує JPEG 2000.

Через ці фактори JPEG 2000 залишився нішевим форматом, але знайшов застосування у сферах, де його переваги були критичними:

- Цифрове кіно: Стандарт Motion JPEG 2000 (MJ2) став стандартом для кодування цифрових кінофільмів;
- Медична візуалізація: Висока глибина кольору та стиснення без втрат роблять його ідеальним для зберігання медичних знімків (DICOM);
- Архівування та геопросторові дані: Бібліотеки, архіви та картографічні сервіси використовують JPEG 2000 для довготривалого зберігання високоякісних цифрових копій.

1.1.6. WebP. WebP – сучасний растровий графічний формат, розроблений компанією Google та вперше представлений у 2010 році. Його основною метою було створення універсального формату для розповсюдження зображень через Інтернет, який би забезпечував значно кращий ступінь стиснення як для фотографій, так і для графіки з прозорістю, прискорюючи завантаження веб-сторінок. WebP поєднує в собі риси JPEG, PNG та GIF, пропонуючи режими стиснення з втратами та без втрат, а також підтримку анімації.

Технічні особливості

Стиснення: WebP підтримує два основні режими стиснення:

- Стиснення з втратами: Оснований на методі внутрішньокадрового кодування (intra-frame coding) з відеокодека VP8. Як і JPEG, він використовує блокове перетворення та квантування, але застосовує більш розвинуті методи прогнозування блоків (block prediction), що дозволяє значно ефективніше кодувати зображення;
- Стиснення без втрат: Окрім скорочення повторюваних фрагментів за допомогою LZ77 та кодування Хаффмана, цей режим також використовує просторове прогнозування (spatial prediction) та колірне кешування (color caching) для перетворення даних пікселів у більш стисливу форму. Альфа канал завжди стискається без втрат незалежно від режиму.

Модель та глибина кольору: WebP працює з 8-бітним колірним простором YUV420 для режиму з втратами та 8-бітним RGBA для режиму без втрат. Це забезпечує підтримку True Color та повноцінний альфа-канал, але означає що стиснення з втратами завжди використовує субдискретизацію кольору (chroma subsampling).

Підтримка прозорості: Формат WebP має повну підтримку 8-бітного альфа-каналу як у режимі без втрат, так і в режимі з втратами. Це є значною перевагою над JPEG, який не підтримує прозорість, та дозволяє створювати файли з прозорістю значно меншого розміру, ніж PNG.

Анімація: WebP підтримує анімацію, дозволяючи зберігати послідовність кадрів в одному файлі. Кадри WebP можуть використовувати як стиснення з втратами, так і без втрат, що дозволяє зберігати анімації без обмежень кольору та повною підтримкою прозорості ефективно за розміром, що є перевагою над GIF та APNG.

Структура файлу та метадані. Файл WebP базується на контейнері RIFF (Resource Interchange File Format). Структура файлу складається з блоків (chunks), які містять різні типи даних:

- Заголовок RIFF – ідентифікує файл як WebP;
- Блок VP8X – містить метаінформацію про можливості, що використовуються у файлі (наприклад, наявність анімації, альфа-каналу, ICC-профілю);
- Блоки даних зображення – позначається як VP8 (для стиснення з втратами) або VP8L (для стиснення без втрат);
- Блок ALPH – містить стиснені дані альфа-каналу (використовується в режимі з втратами);
- Блоки ANIM та ANMF – використовуються для зберігання глобальних параметрів анімації та інформації про окремі кадри відповідно.

WebP підтримує зберігання метаданих у форматах EXIF та XMP, а також колірних профілів ICC. Ця інформація зберігається у відповідно позначених спеціалізованих блоках всередині RIFF-контейнера.

Прийняття та поточна актуальність. З моменту своєї появи WebP поступово здобував популярність, особливо після того, як його підтримку додали найбільш поширені веб-браузери (Chrome, Firefox, Edge, Safari). Сьогодні він широко використовується багатьма IT-компаніями для оптимізації веб-контенту та зменшення навантаження на мережу.

WebP є хорошою заміною для JPEG, PNG та GIF в Інтернеті, вважаючи краще стиснення для всіх типів зображень в одному форматі, однак його прийняття за межами вебу залишається обмеженим. Багато графічних редакторів та операційних систем досі не мають вбудованої підтримки WebP, що іноді вимагає використання розширень або спеціалізованого програмного забезпечення для роботи з ним.

1.1.7. HEIC (High Efficiency Image Container). HEIC – реалізація формату-контейнера HEIF (High Efficiency Image File Format), що використовує для стиснення зображень відеокодек HEVC (High Efficiency Video Coding, також відомий як H.265). Формат HEIC був розроблений Moving Picture Experts Group (MPEG) та стандартизований у 2015 році як частина MPEG-H Part 12. Основною метою HEIC є забезпечити значно кращий ступінь стиснення порівняно з JPEG, зберігаючи при цьому високу якість зображення. HEIC набув широкого поширення після того, як компанія Apple прийняла його як стандартний формат для фотографій на пристроях iOS починаючи з версії 11 у 2017 році.

Технічні особливості

Стиснення: HEIC використовує алгоритм стиснення з втратами на основі високоефективного відеокодека HEVC (High Efficiency Video Coding), також відомого як H.265. На відміну від JPEG, який використовує дискретне косинусне перетворення на фіксованих блоках 8x8, HEVC розбиває зображення на блоки кодування (Coding Tree Units), розмір яких може варіюватися, та використовує значно більшу кількість режимів внутрішньокадрового прогнозування (intra-prediction modes). Це дозволяє набагато точніше відтворювати текстури та деталі там де це необхідно, та ефективно стискати області однорідного кольору, що призводить до значного зменшення розміру файлу порівняно з JPEG за тої ж візуальної якості. Хоча зазвичай HEIC налаштований на стиснення з втратами, кодек HEVC можна використовувати без квантування та трансформації даних, що дозволяє зберігати піксельні дані без втрат.

Модель та глибина кольору: HEIC, окрім стандартної 8-бітної глибини кольору, також підтримує 10-біт та 12-біт на канал. Це забезпечує більш точне відтворення кольорів та плавних градієнтів, що особливо важливо для професійної фотографії та контенту HDR (High Dynamic Range). При кодуванні колірні канали RGB перетворюються у простір YCbCr, що допомагає ефективно стискати колірну інформацію, а також дозволяє зменшувати колірну дискретизацію.

Підтримка прозорості: Формат HEIC підтримує альфа-канал, кодуючи його у окремий HEVC потік, що дозволяє зберігати зображення з різними рівнями прозорості.

Додаткові можливості: Контейнер HEIF, на якому базується HEIC, підтримує зберігання не тільки окремих зображень, але й:

- Послідовності зображень (для серійної зйомки або анімацій);
- Допоміжні зображення, такі як карти глибини (depth map), що дозволяє змінювати фокус або застосовувати ефекти боке після зйомки;
- Зображення з різними експозиціями, для створення HDR-фотографій;
- Мініатюри для попереднього перегляду;
- Аудіофайл, пов’язаний із зображенням;
- Послідовність недеструктивного редагування, що дозволяє застосовувати деякі операції (такі як обрізка, поворот, експозиція) з можливістю скасування без втрати даних.

Структура файлу та метадані. Файл HEIC базується на форматі ISOBMFF (ISO Base Media File Format), який також є основою для контейнерів MP4. Файл складається з ієрархічної структури блоків, кожен з яких містить певний тип даних:

- ftyp (File Type Box) – ідентифікує файл як HEIF/HEIC;
- meta (Meta Box) – містить метадані про зображення;
- mdat (Media Data Box) – стиснені дані пікселів зображення;
- iloc (Item Location Box) – вказує розташування даних зображень та інших елементів у файлі;
- iinf (Item Info Box) – надає інформацію про кожен елемент у файлі;
- iprp (Item Properties Box) – містить властивості зображень, такі як розміри, колірний простір, інформація про прозорість.

HEIC має розширену підтримку метаданих, з можливістю зберігати метадані EXIF, XMP та IPTC, а також колірні профілі ICC. Завдяки гнучкості ISOBMFF, HEIC може включати будь-які додаткові метадані, необхідні для розширених функцій (наприклад, інформацію про глибину сцени для портретного режиму).

Прийняття та поточна актуальність. Прийняття HEIC значно зросло після його інтеграції в екосистему Apple, що зробило його одним з найпоширеніших форматів для фотографій, зроблених на iPhone. Однак за межами пристроїв Apple його підтримка все ще неоднорідна. Хоча Windows 10 та 11 підтримують HEIC за допомогою додаткових кодеків, а деякі графічні редактори та веб-браузери поступово додають підтримку, його поширення за межами цих екосистем, особливо в Інтернеті, залишається обмеженим. Основною перешкодою є патентні обмеження та ліцензійні відрахування, пов'язані з кодеком HEVC.

На сьогодні HEIC є потужним та ефективним форматом для зберігання фотографій на персональних пристроях, але використання його як універсального веб-стандарту для широкого обміну зображеннями залишається під питанням через ліцензійну політику, обчислювально ресурсоємне декодування, менш підтримувану прозорість та відсутність таких функцій як прогресивне завантаження.

1.1.8. AVIF (AV1 Image File Format). AVIF – растровий графічний формат, що використовує для стиснення зображень відкритий та вільний від роялті відеокодек AV1. Формат був розроблений Альянсом за відкриті медіа (Alliance for Open Media), до якого входять такі компанії, як Google, Mozilla, Microsoft, Amazon та Netflix. AVIF був представлений у 2019 році за потребу в сучасному, високоефективному форматі зображень, який би перевершував існуючі стандарти (JPEG, WebP) і водночас не мав патентних обмежень, на відміну від HEIC.

Технічні особливості

Стиснення: Формат підтримує як стиснення з втратами, так і без втрат. AVIF використовує метод внутрішньокадрового кодування з відеокодека AV1. Цей кодек застосовує значно складніші та ефективніші алгоритми, ніж його попередники. AV1 використовує гнучкі блоки кодування та передові методи прогнозування, адаптивні фільтри (Constrained Directional Enhancement Filter, Loop Restoration Filter) та ентропійне кодування. Це дозволяє досягти значно кращого співвідношення "якість/розмір" – файли AVIF можуть бути на 50% меншими за JPEG та на 20-30% меншими за WebP за аналогічної візуальної якості, особливо за низьких бітрейтів. Стиснення без втрат використовує той же алгоритм, але пропускає операції квантування та фільтрації, зберігаючи оригінальні значення пікселів.

Модель та глибина кольору: AVIF має розширену підтримку кольору:

- Високу глибину кольору – 8, 10 та 12 біт на колірний канал;
- High Dynamic Range (HDR) – Повну підтримку HDR-контенту з метаданими PQ (Perceptual Quantizer) та HLG (Hybrid Log-Gamma);
- Wide Color Gamut (WCG) – Підтримка широких колірних просторів, таких як BT.2020 та BT.2100;
- YCbCr, що дозволяє застосовувати субдискретизацію кольору (chroma subsampling) у різних конфігураціях (4:4:4, 4:2:2, 4:2:0);
- Підтримка монохромних зображень та зображень з альфа-каналом.

Підтримка прозорості: Формат має повну підтримку альфа-каналу з такою ж глибиною, як і основні колірні канали (8, 10 або 12 біт), що дозволяє створювати високоякісні зображення з плавною прозорістю.

Додаткові можливості: Оскільки AVIF використовує контейнер HEIF, він успадковує багато його можливостей, зокрема:

- Анімація – Підтримка зберігання послідовності кадрів для створення анімацій;
- Допоміжні дані – Можливість зберігати карти глибини, мініатюри та послідовності зображень;
- Синтез плівкового зерна (Film Grain Synthesis) – Унікальна особливість кодека AV1, яка дозволяє видалити плівкове зерно перед стисненням, а потім алгоритмічно відтворити його під час декодування. Це значно підвищує ефективність стиснення для зображень із зернистою текстурою, адже шум важко стиснути та відносно легко відтворити.

Структура файлу та метадані. Як і HEIC, AVIF базується на форматі-контейнері HEIF, який, у свою чергу, є реалізацією ISOBMFF. Структура файлу є блоковою та ієрархічною, що дозволяє гнучко зберігати різні типи даних. Ключова відмінність від HEIC полягає в тому, що стиснені дані зображення в блоці mdat закодовані за допомогою кодека AV1, а не HEVC.

AVIF має повну підтримку метаданих, включаючи EXIF, XMP та колірні профілі ICC. Уся ця інформація зберігається у відповідних блоках контейнера HEIF.

Прийняття та поточна актуальність. AVIF демонструє одні з найшвидших темпів прийняття серед сучасних форматів зображень, в першу чергу завдяки своїй відкритості та відсутності ліцензійних відрахувань. Його підтримка вже реалізована в більшості сучасних веб-браузерів, а великі контент-платформи та CDN (наприклад, Cloudflare, Netflix) активно використовують AVIF для зменшення трафіку та прискорення завантаження контенту.

Сьогодні AVIF позиціонується як головний конкурент HEIC та довгоочікувана заміна для JPEG в Інтернеті, його технічні переваги та відкрита модель ліцензування роблять його провідним кандидатом на роль універсального формату зображень. Але незважаючи на вищу ефективність стиснення, головним недоліком AVIF залишається висока обчислювальна

складність процесу кодування, що вимагає значно більше часу та ресурсів порівняно з JPEG або WebP, та відсутність таких функцій як прогресивне завантаження.

1.1.9. JPEG XL. JPEG XL – растровий графічний формат нового покоління, розроблений Об'єднаною групою експертів з фотографії (Joint Photographic Experts Group), тією ж організацією, що створила JPEG-1. Стандартизований у 2022 році, JPEG XL був створений з амбітною метою стати універсальним форматом на заміну таких форматів як JPEG, PNG та GIF. Його ключові особливості – висока ефективність стиснення, широкий набір функцій та, що найважливіше, можливість неруйнівного перекодування існуючих файлів JPEG для зменшення їх розміру.

Технічні особливості

Стиснення: JPEG XL використовує два взаємодоповнюючі режими кодування, що дозволяє йому бути однаково ефективним як для фотографій, так і для синтетичної графіки:

- Режим VarDCT (Variable-blocksize DCT): Використовується для стиснення з втратами. Це вдосконалена версія алгоритму DCT з JPEG, але замість фіксованих блоків 8x8, VarDCT використовує блоки змінного розміру (від 2x2 до 256x256) та форми, що дозволяє краще адаптуватися до вмісту зображення, мінімізуючи артефакти;
- Модульний режим (Modular): Використовується в першу чергу для стиснення без втрат. Цей режим використовує методи прогнозування, контекстного моделювання та колірних трансформацій, частково запозичених з PNG, але вдосконалених та застосованих на окремі канали та блоки. Даний режим також може використовувати стиснення з втратами, що іноді підходить краще до деяких типів зображень, а також використовується для окремих каналів таких як альфа, для кращого стиснення та уникання DCT артефактів.

Неруйнівне перекодування JPEG (Lossless JPEG Recompression): Це унікальна та ключова функція JPEG XL. Формат може напряму перекодувати існуючі файли JPEG, зменшуючи їх розмір приблизно на 20% без будь-якої втрати якості. При цьому процес є повністю оборотним, що дозволяє відновити оригінальний JPEG файл за потреби. Це робить JPEG XL ідеальним для архівування та оптимізації великих колекцій JPEG зображень.

Модель та глибина кольору: Формат підтримує високу глибину кольору до 32 біт на канал, повний HDR (з метаданими PQ та HLG) та широкі колірні простори (Wide Color Gamut), що відповідає та перевершує можливості AVIF та HEIC.

Підтримка прозорості: JPEG XL має повну підтримку альфа-каналу, який ефективно стискається за допомогою модульного режиму.

Прогресивне завантаження: Формат має вбудовану підтримку прогресивного рендерингу, що є значно досконалішим, ніж у JPEG. JPEG XL дозволяє декодувати зображення у низькій якості лише з невеликої частини стиснених даних, це означає що при завантаженні на веб-сайтах зображення з’являється майже одразу та поступово деталізується. Формати що не підтримують прогресивний рендеринг потребують завантаження всіх даних зображення перед відображенням, що може погіршити користувацький досвід, особливо у повільних мережах.

Анімація: JPEG XL має підтримку анімації, що дозволяє йому замінити анімовані GIF, APNG та WebP.

Підтримка шарів: JPEG XL нативно підтримує зберігання кількох шарів в одному файлі. Це дозволяє реалізовувати функціонал подібний до форматів PSD або TIFF, зберігаючи окремі елементи зображення (наприклад, текст, графіку, маски) без їх злиття. Кожен шар може мати власні розміри, положення та режим злиття, що робить формат придатним для складних графічних проєктів та недеструктивного редагування.

Велика кількість каналів: На відміну від більшості веб-форматів, що обмежуються трьома (RGB) або чотирма (RGBA) каналами, JPEG XL підтримує до 4099 додаткових каналів. Це дозволяє зберігати дані для професійного друку (CMYK, плашкові кольори), наукової візуалізації (інфрачервоні канали, карти глибини) та 3D-графіки.

Структура файлу та метадані. JPEG XL використовує власний контейнер, який базується на принципах ISOBMFF (схожий на HEIF), що забезпечує гнучкість та розширюваність. Файл складається з заголовка, що ідентифікує його як .jxl, та послідовності блоків, які містять стиснені дані, метадані та іншу інформацію.

Формат має повну підтримку метаданих EXIF, XMP та колірних профілів ICC. При неруйнівному перекодуванні з JPEG усі оригінальні метадані зберігаються без змін.

Прийняття та поточна актуальність. JPEG XL позиціонується як технічно досконалий формат для потреб сьогодення та майбутнього, що вирішує проблеми майже всіх попередників. Його підтримують багато професійних графічних редакторів (Adobe Camera Raw, GIMP, Krita) та інструментів командного рядка. Відкритий код та відсутність ліцензійних відрахувань є його значною перевагою.

Однак головною перешкодою для його масового прийняття стала позиція розробників веб-браузерів. Наприкінці 2022 року команда Google Chrome ухвалила рішення видалити експериментальну підтримку JPEG XL, мотивуючи це недостатнім інтересом з боку екосистеми та перевагою існуючих форматів. Це рішення значно сповільнило його інтеграцію в веб.

На сьогодні JPEG XL є форматом з величезним потенціалом, особливо для професійних фотографів, дизайнерів та для архівування цифрових активів. Проте його майбутнє як переважного веб-стандарту залишається невизначеним, доки він не отримає нативної підтримки у провідних браузерах.

1.2. Теоретичні основи алгоритмів стиснення

В основі кожного формату растрових зображень лежить набір математичних алгоритмів, які визначають його здатність зменшувати надлишковість даних. Вибір та реалізація цих алгоритмів є ключовим фактором, що впливає на баланс між ступенем стиснення, візуальною якістю та обчислювальною складністю формату. Розуміння принципів їхньої роботи є фундаментальною передумовою для об'єктивного аналізу та порівняння ефективності форматів. У даному підрозділі проводиться аналіз основних парадигм стиснення: методів без втрат, що гарантують повну ідентичність відновлених даних, та методів з керованими втратами, які використовують особливості людського зорового сприйняття для досягнення значно вищих коефіцієнтів компресії.

1.2.1. RLE. RLE (Run-Length Encoding, англ. Кодування довжин серій) – Один із найпростіших алгоритмів стиснення без втрат, що базується на ідентифікації та компактному представленні послідовностей однакових значень. Його концепція застосовувалася ще в ранніх системах передачі даних, зокрема у факсимільних апаратах.

Алгоритм RLE оперує на принципі ідентифікації та компактного представлення послідовних повторень ідентичних значень (серій) у вхідному потоці даних. Процес кодування замінює таку серію на кортеж, що зазвичай складається з довжини серії (кількості повторень) та самого значення. Таким чином, алгоритм трансформує дані, зменшуючи їх обсяг за рахунок усунення локальної надлишковості. Декодування є симетричною операцією, що відтворює вихідну послідовність на основі цих кортежів, гарантуючи повну відсутність втрат інформації. Критичним недоліком є низька ефективність на даних з високою ентропією (наприклад, фотографічний шум), де відсутні довгі серії, що може призвести до збільшення розміру файлу (негативне стиснення).

Оптимальним застосуванням даного алгоритму є зображення з великими областями суцільного кольору, такі як іконки, логотипи, діаграми та проста комп'ютерна графіка. Алгоритм RLE використовується у таких форматах як BMP (для зображень з індексованою палітрою), TGA та PCX.

1.2.2. Кодування Хаффмана. Huffman Coding (англ. Кодування Хаффмана) – це фундаментальний статистичний алгоритм стиснення без втрат, який присвоює символам вхідного потоку префіксні двійкові коди змінної довжини. Основний принцип полягає в тому, що символи, які зустрічаються найчастіше, отримують найкоротші коди, а рідкісні – найдовші.

Процес кодування складається з кількох етапів:

- Аналіз частот: На першому етапі виконується повний прохід по вхідних даних для побудови таблиці частот появи кожного унікального символу;
- Побудова дерева: На основі таблиці частот будується двійкове дерево (дерево Хаффмана). Процес починається з того, що кожен символ розглядається як вузол-листок з вагою, що дорівнює його частоті. На кожній ітерації два вузли з найменшими вагами об'єднуються в новий батьківський вузол, вага якого дорівнює сумі ваг його дочірніх вузлів. Цей процес повторюється, доки всі вузли не будуть об'єднані в єдине дерево з одним коренем;
- Призначення кодів: Двійкові коди для кожного символу генеруються шляхом обходу дерева від кореня до відповідного листка. Зазвичай, рух до лівого дочірнього вузла позначається як ‘0’, а до правого – як ‘1’. Отримана система кодів є префіксною, що означає, що жоден код не є початком іншого, і це дозволяє однозначно декодувати послідовний потік бітів.

Кодування Хаффмана широко використовується як фінальний етап ентропійного кодування в багатьох алгоритмах, зокрема в класичному JPEG та як частина гібридного алгоритму DEFLATE, що є основою форматів PNG та ZIP. Його перевагою є простота та оптимальність для відомого статичного розподілу

ймовірностей. Обмеженням є необхідність двох проходів по даних або передачі таблиці частот разом із закодованими даними, що створює додаткові накладні витрати.

1.2.3. Словникові алгоритми: LZ77, LZW та DEFLATE. Словникові алгоритми відносяться до сімейства алгоритмів стиснення без втрат, що працюють за принципом заміни повторюваних послідовностей даних посиланнями на їх попередні входження у динамічно формованому "словнику".

Принципи роботи кожного з алгоритмів:

- LZ77 (Лемпель-Зів 1977): В основі алгоритму лежить концепція "ковзного вікна", яке поділене на дві частини: "словник" (буфер з нещодавно обробленими даними) та "буфер попереднього перегляду" (дані, що підлягають кодуванню). Кодер шукає найдовший збіг початку буфера перегляду з будь-якою підпослідовністю у словнику. Якщо збіг знайдено, він замінюється кортежем (відстань, довжина, наступний символ), де відстань – це зсув назад у словнику, довжина – довжина збігу, а наступний символ – перший символ після збігу. Якщо збіг не знайдено, кодується лише сам символ;
- LZW (Лемпель-Зів-Велч): На відміну від LZ77, LZW будує явний словник рядків під час процесу кодування. Словник ініціалізується всіма можливими символами. Алгоритм зчитує з вхідного потоку найдовший рядок, що вже існує в словнику. Потім у вихідний потік записується код цього рядка, а до словника додається новий запис, що складається з цього рядка та наступного символу з вхідного потоку. Декодер може реконструювати ідентичний словник паралельно, не вимагаючи його передачі;
- DEFLATE: Це гібридний алгоритм, що поєднує в собі LZ77 для усунення надлишковості та кодування Хаффмана для подальшого стиснення. На першому етапі LZ77 знаходить повторювані послідовності та замінює їх посиланнями (відстань, довжина). Дані, для яких збігів не знайдено,

залишаються як є (літерали). На другому етапі отриманий потік літералів та посилань стискається за допомогою кодування Хаффмана, для чого будуються два окремі дерева: одне для літералів та довжин, інше – для відстаней.

Алгоритм LZW був основою раннього веб-формату GIF. DEFLATE є стандартом де-факто для стиснення даних і є ключовим компонентом форматів PNG, ZIP та GZIP. Ці алгоритми є надзвичайно ефективними для даних з високим рівнем повторень, але їх ефективність знижується на даних, що нагадують шум.

1.2.4. DCT. DCT (Discrete Cosine Transform, англ. Дискретне косинусне перетворення) – це ортогональне перетворення, що є математичним ядром стандарту стиснення JPEG. Його основна функція – перетворення просторової інформації про пікселі в частотну область, що дозволяє ефективно відокремити візуально значущу інформацію від менш значущої.

Принцип роботи алгоритму:

- Сегментація на блоки: Зображення поділяється на непересічні блоки розміром 8x8 пікселів. Усі подальші операції виконуються над кожним блоком незалежно;
- Пряме перетворення: До кожного блоку застосовується двовимірне DCT-II, яке перетворює 64 значення інтенсивності пікселів у 64 частотні коефіцієнти. Цей процес розкладає просторовий сигнал блоку на суму косинусоїдних функцій з різними просторовими частотами;
- Інтерпретація коефіцієнтів: Результатом є матриця 8x8 коефіцієнтів. Верхній лівий коефіцієнт (0,0) називається DC-коефіцієнтом і представляє середню інтенсивність всього блоку. Решта 63 коефіцієнти називаються AC-коефіцієнтами і представляють амплітуди вищих просторових частот (деталей) у горизонтальному та вертикальному напрямках. Ключовою властивістю DCT є концентрація енергії, коли більша частина візуальної інформації міститься у невеликій кількості низькочастотних коефіцієнтів.

DCT є фундаментальним для формату JPEG та багатьох відеокодеків. Його головна перевага полягає у високій ефективності стиснення та наявності швидких алгоритмів для його обчислення. Однак його блокова основа є і головним недоліком, оскільки за високих коефіцієнтів стиснення на межах блоків виникають помітні візуальні артефакти ("блочність").

1.2.5. DWT. DWT (Discrete Wavelet Transform, англ. Дискретне вейвлет-перетворення) – це математичний апарат, що лежить в основі стандарту стиснення JPEG 2000. На відміну від DCT, що аналізує зображення у фіксованих блоках, DWT розглядає зображення цілісно, розкладаючи його на набір сигналів різної частоти та просторової роздільної здатності. Це дозволяє уникнути блокових артефактів та досягти кращої якості зображення за низьких бітрейтів.

Процес DWT є ітеративним і базується на застосуванні пари фільтрів – низькочастотного та високочастотного – до даних зображення.

- Одновимірне перетворення: Спочатку фільтри застосовуються до кожного рядка пікселів. Низькочастотний фільтр усереднює значення, створюючи зменшену, більш розмиту версію рядка (апроксимація). Високочастотний фільтр, навпаки, фіксує різницю між пікселями, зберігаючи інформацію про деталі. Після цього кількість коефіцієнтів зменшується вдвічі (субдискретизація);
- Двовимірне перетворення: Та ж сама операція застосовується до стовпців отриманих на попередньому етапі даних. В результаті зображення розкладається на чотири субдіапазони:
    - LL (Low-Low): Зменшена версія оригінального зображення (апроксимація);
    - LH (Low-High): Горизонтальні деталі (вертикальні межі);
    - HL (High-Low): Вертикальні деталі (горизонтальні межі);
    - HH (High-High): Діагональні деталі.
- Багаторівневе розкладання: Процес повторюється рекурсивно для LL-діапазону, щоразу створюючи новий набір з чотирьох субдіапазонів

меншого розміру. Це створює пірамідальну структуру, де зображення представлено на різних рівнях деталізації.

Формат JPEG 2000 використовує два типи вейвлет-перетворень: незворотне 9/7 (на основі чисел з рухомою комою) для стиснення з втратами та зворотне 5/3 (цілочисельне) для стиснення без втрат. Завдяки своїй природі, DWT забезпечує ефективне прогресивне завантаження як за якістю, так і за роздільною здатністю, що є його значною перевагою. Головним обмеженням є вища обчислювальна складність порівняно з DCT.

1.2.6. Блокове стиснення. Блокове стиснення (Block-based Compression) – це переважна парадигма в сучасних методах стиснення зображень та відео з втратами, що полягає у поділі зображення на сітку блоків, які обробляються відносно незалежно, дозволяючи локалізувати та паралелізувати обчислення.

Загальний процес для кожного блоку виглядає наступним чином:

- Розбиття (Partitioning): Зображення поділяється на блоки. Якщо в JPEG це фіксовані блоки 8x8, то в сучасних кодеках, як HEVC та AV1, використовуються ієрархічні структури (Coding Tree Units), які можуть рекурсивно поділятися на менші блоки різного розміру та форми. Це дозволяє краще адаптуватися до вмісту: великі блоки для гладких областей, а менші для деталей;
- Прогнозування: Для блоку генерується прогноз на основі вже закодованих сусідніх блоків (внутрішньокадрове прогнозування);
- Трансформація та Квантування: Кодується не сам блок, а залишковий сигнал (різниця між оригіналом та прогнозом). Цей залишок піддається частотному перетворенню (DCT або його варіації) та квантуванню;
- Внутрішньоциклова фільтрація: Сучасні кодеки застосовують фільтри (наприклад, deblocking filter, SAO) до реконструйованого зображення всередині циклу кодування. Це зменшує артефакти на межах блоків і

покращує якість зображення, яке буде використовуватись для прогнозування наступних блоків.

Ця парадигма є фундаментальною для JPEG, WebP, HEIC, AVIF та практично всіх сучасних відеокодеків. Вона дозволяє досягти високого ступеня стиснення та ефективно використовувати обчислювальні ресурси. Головним викликом, який вирішують сучасні формати, є мінімізація артефактів на межах блоків.

1.2.7. Алгоритми прогнозування. Прогнозування (Prediction) є ключовою технікою для зменшення надлишковості, за якою значення пікселя або блоку передбачається на основі вже закодованих сусідніх даних. Кодується лише помилка прогнозування (залишок), яка зазвичай має значно меншу ентропію, ніж вихідні дані.

Принцип роботи:

- Прогностична фільтрація (у стисненні без втрат): Формат PNG використовує набір з п'яти простих фільтрів (None, Sub, Up, Average, Paeth), які застосовуються до кожного рядка пікселів. Фільтр Sub прогнозує піксель як значення лівого сусіда, Up – як значення верхнього. Кодер для кожного рядка обирає той фільтр, який мінімізує суму абсолютних значень залишків, що робить дані більш "стисливими" для подальшого етапу DEFLATE;
- Внутрішньокадрове прогнозування (у стисненні з втратами): Сучасні кодеки (WebP, HEVC, AV1) використовують значно складніший підхід. Для кожного блоку тестується велика кількість "режимів прогнозування". Це включає DC-прогнозування (заповнення середнім значенням сусідів), планарне прогнозування (створення плавного градієнта) та велику кількість напрямлених режимів, які екстраполюють значення пікселів вздовж певного кута. Кодер виконує процес оптимізації "швидкість-

спотворення" (rate-distortion optimization), щоб обрати режим, який забезпечує найкращу якість при найменшій кількості біт.

Ця техніка є критично важливою для ефективності як PNG, так і всіх сучасних форматів з втратами. Вона дозволяє суттєво зменшити енергію залишкового сигналу перед трансформацією та квантуванням. Основний компроміс полягає у значній обчислювальній складності кодера, який повинен перевіряти безліч режимів для кожного блоку.

1.2.8. Квантування. Квантування (Quantization) – це центральний і незворотний процес у стисненні з втратами, на якому відбувається контрольована втрата інформації. Дана операція зменшує точність частотних коефіцієнтів, отриманих після таких перетворень як DCT або DWT, відповідно до психовізуальних моделей людського сприйняття.

Процес полягає в поелементному діленні матриці частотних коефіцієнтів C на матрицю кроків квантування Q, з подальшим округленням до найближчого цілого:

$$C_q(u,v) = \left[\frac{C(u,v)}{Q(u,v)}\right] \tag{1.1}$$

Матриця Q розроблена таким чином, що для низькочастотних коефіцієнтів (до яких око чутливе) використовуються малі значення, що зберігає їх точність. Для високочастотних коефіцієнтів (які відповідають за дрібні деталі та шум) використовуються значно більші значення, що призводить до їх сильного огрубіння або повного анулювання. Загальний рівень якості контролюється єдиним параметром, який масштабує всю матрицю квантування.

Квантування є невід'ємною частиною всіх форматів з втратами таких як JPEG, WebP, HEIC та AVIF. Це найпотужніший інструмент для керування балансом між розміром файлу та візуальною якістю. Водночас, саме агресивне квантування є основним джерелом артефактів стиснення, таких як втрата текстур, блочність та кільця (ringing) навколо контурів.

1.2.9. Колірна субдискретизація. Колірна субдискретизація (Chroma Subsampling) – це техніка стиснення з втратами, що базується на психовізуальній властивості людського зору, який має значно вищу чутливість до змін яскравості, ніж до змін кольору.

Принцип роботи:

- Перетворення колірного простору: Зображення перетворюється з адитивного простору RGB у простір, що розділяє яскравість та колір, наприклад, YCbCr. Канал Y представляє яскравість, а канали Cb та Cr – колірні відмінності (синій та червоний відповідно);
- Зменшення роздільної здатності: Канал яскравості (Y) зберігається з повною роздільною здатністю, тоді як просторова роздільна здатність колірних каналів (Cb та Cr) зменшується. Найпоширенішою схемою є 4:2:0, де для блоку з 2x2 пікселів зберігаються всі 4 значення яскравості, але лише по одному значенню для Cb та для Cr. Таким чином, кількість інформації про колір зменшується в 4 рази. Такі формати як JPEG підтримують різні схеми, такі як 4:2:2 (зменшення лише по горизонталі) або 4:4:4 (без зменшення), тоді як кодек WebP у режимі з втратами може працювати лише в режимі YCbCr 4:2:0.

Ця техніка широко застосовується у JPEG, WebP, HEIC, AVIF та більшості стандартів цифрового відео. Головна перевага даної техніки – значне зменшення обсягу даних (до 50% для схеми 4:2:0) з мінімальним візуальним впливом на фотографічні зображення. Обмеженням є можлива поява артефактів, таких як розмиття кольору на різких межах, що робить її менш придатною для синтетичної графіки або тексту, де точність передачі кольору є критичною.

1.2.10. Ентропійне кодування. Сучасні методи ентропійного кодування, до яких входить арифметичне кодування та ANS, відносяться до класу алгоритмів стиснення без втрат, які є більш ефективними за кодування

Хаффмана, оскільки вони здатні представляти символи дробовою кількістю біт, наближаючись до теоретичної межі стиснення, визначеної ентропією Шеннона.

Принцип роботи:

- Арифметичне кодування: На відміну від кодування Хаффмана, яке замінює кожен символ цілим кодом, арифметичне кодування представляє весь потік даних як єдине дійсне число в інтервалі [0, 1). Цей інтервал рекурсивно звужується з кожним наступним символом. Ширина під-інтервалу, що виділяється для символу, пропорційна його ймовірності. Чим вища ймовірність, тим менше звужується інтервал, і тим менше біт потрібно для кодування. Сучасні контекстно-адаптивні версії, як CABAC (використовується в HEVC), динамічно оновлюють ймовірності символів на основі вже закодованого контексту, що значно підвищує ефективність;
- ANS (Asymmetric Numeral Systems): Це новітнє сімейство ентропійних кодерів, що досягає ефективності стиснення, порівнянної з арифметичним кодуванням, але при значно вищій швидкодії, особливо при декодуванні. ANS працює як стекова операція над одним натуральним числом (станом), де кодування символу відповідає додаванню інформації до стану, а декодування – її вилученню.

Арифметичне кодування та його варіації є стандартом для JPEG 2000 та HEVC (що використовується у форматі HEIC). Новіший та швидший ANS є ключовим компонентом таких сучасних форматів, як JPEG XL. Головною перевагою цих методів є вищий коефіцієнт стиснення. Історично, недоліками арифметичного кодування були патенти та вища обчислювальна складність, однак ці проблеми значною мірою вирішені завдяки сучасним реалізаціям та появі вільних альтернатив, як ANS.

1.2.11. VarDCT. VarDCT (Variable-blocksize Discrete Cosine Transform, англ. Дискретне косинусне перетворення зі змінним розміром блоку) – удосконалення класичного алгоритму DCT, розроблений для подолання його

фундаментальних обмежень, пов'язаних з фіксованим розміром блоків. Цей режим, що є одним із ключових у стандарті JPEG XL, використовує адаптивну структуру блоків різного розміру та форми, що дозволяє значно краще пристосовуватися до локальних характеристик зображення, мінімізуючи артефакти та підвищуючи ефективність стиснення.

На відміну від жорсткої сітки 8x8 у JPEG, VarDCT застосовує значно гнучкіший підхід до обробки зображення:

- Адаптивне розбиття на блоки: Зображення поділяється на групи блоків (зазвичай 256x256 пікселів), які, у свою чергу, рекурсивно розбиваються за допомогою дерева квадрантів (quad-tree). Кодер аналізує вміст кожного блоку: для гладких, низькочастотних областей використовуються великі блоки (до 256x256), що дозволяє ефективно кодувати їх з мінімальними накладними витратами. В областях з високою деталізацією, текстурами або різкими межами блок рекурсивно поділяється на менші підблоки (аж до 2x2). Таке адаптивне розбиття дозволяє "концентрувати" біти там, де вони найбільш потрібні для збереження візуальної інформації;
- Використання блоків різної форми: Окрім квадратних блоків, VarDCT у JPEG XL підтримує прямокутні форми (наприклад, 16x8, 8x32, 64x16). Це дозволяє ще точніше адаптуватися до анізотропних структур у зображенні, що додатково покращує енергетичну концентрацію та зменшує артефакти;
- Вибір адаптивного перетворення: Для кожного блоку , окрім DCT-II, кодер може обирати інші види ортогональних перетворень, такі як AFV (Adaptive Frequency Vectorization) або перетворення Хорнуса. Це дозволяє обрати математичний апарат, який найкраще відповідає статистичним властивостям сигналу всередині конкретного блоку, що призводить до ще кращої концентрації енергії в меншій кількості коефіцієнтів;
- Квантування та ентропійне кодування: Після етапу перетворення отримані коефіцієнти піддаються квантуванню та ентропійному кодуванню. Однак, завдяки значно ефективнішому представленню сигналу, досягнутому на

попередніх етапах, для досягнення тієї ж візуальної якості потрібне менш агресивне квантування. Для фінального стиснення коефіцієнтів використовується високопродуктивне асиметричне кодування (ANS).

Режим VarDCT є основним інструментом для стиснення з втратами фотографічних зображень у форматі JPEG XL. Його ключовою перевагою є практично повне усунення блокових артефактів, що є головним недоліком класичного JPEG. Завдяки адаптивній структурі блоків, межі між ними стають візуально непомітними, а деталі та текстури зберігаються значно точніше за аналогічних коефіцієнтів стиснення. Це забезпечує суттєво вищу візуальну якість при тому ж розмірі файлу. Головним компромісом є значно вища обчислювальна складність кодера, який повинен виконувати складний процес оптимізації "швидкість-спотворення" (rate-distortion optimization) для знаходження оптимальної структури розбиття для всього зображення.

1.2.12. Modular Mode JPEG XL. Модульний режим (Modular) є гнучкою та потужною системою кодування, що лежить в основі формату JPEG XL. Він призначений в першу чергу для ефективного стиснення без втрат та з майже непомітними втратами (near-lossless), що робить його ідеальним для синтетичної графіки (логотипів, діаграм, тексту). Однак його роль у форматі значно ширша, адже він використовується для кодування будь-яких нефотографічних даних, включаючи альфа-канали, шари, карти глибини, а також метадані, необхідні для роботи режиму VarDCT (наприклад, карти квантування).

В основі модульного режиму лежить удосконалена система прогностичного кодування, що значно перевершує фіксовані фільтри PNG за рахунок динамічної адаптації та потужнішого контекстного моделювання. Процес кодування включає кілька інноваційних етапів:

- Оборотне колірне перетворення: За замовчуванням, для багатоканальних зображень (наприклад, RGB) застосовується оборотне колірне перетворення YCoCg-R. Воно перетворює канали в простір, що розділяє

яскравість та колір, що значно зменшує кореляцію між ними і робить дані більш передбачуваними та сприятливими до стиснення. Цей процес є повністю оборотним, що гарантує відсутність втрат;

- Адаптивне прогностичне кодування: Це ключова перевага над PNG. Замість вибору одного з п'яти простих фільтрів для цілого рядка, JPEG XL використовує динамічний набір з 13 параметризованих предикторів, включаючи градієнтні та самокоректуючі. Вибір оптимального предиктора відбувається локально для кожного пікселя (або невеликої групи пікселів), що дозволяє значно точніше моделювати як плавні градієнти, так і різкі межі на зображенні;
- Мета-адаптивне контекстне моделювання: Для ентропійного кодування залишків (помилок прогнозування) кожного пікселя алгоритм аналізує контекст – тобто значення та властивості вже закодованих сусідніх пікселів. На основі цього контексту, за допомогою спеціальної структури – мета-адаптивного дерева (MA-tree) – обирається найбільш відповідний статистичний розподіл для поточного залишкового значення;
- Ентропійне кодування: Обраний за допомогою MA-tree статистичний розподіл передається до асиметричної системи числення (ANS), яка виконує фінальне стиснення;
- Оборотне перетворення Хаара ("Squeeze"): Додатково, модульний режим може застосовувати оборотне перетворення Хаара – операцію, що перегруповує дані, зменшуючи кореляцію між сусідніми пікселями та розділяючи низькочастотні та високочастотні компоненти. Це підвищує ефективність стиснення, а також є основою прогресивного завантаження, дозволяючи відображати зображення з низькою роздільною здатністю до повного завантаження даних.

Модульний режим є основним вибором для стиснення без втрат у JPEG XL, забезпечуючи значно кращі результати (часто на 20-50% менший розмір файлу) порівняно з файлами PNG. Його гнучкість дозволяє також реалізовувати

стиснення з контрольованими втратами, де помилки прогнозування квантуються перед кодуванням. Завдяки своїй універсальності він є фундаментальним компонентом стиснення, як звичайної графіки, так і складних багатошарових зображень. Його головним компромісом є вища обчислювальна складність порівняно з простішими алгоритмами.

1.3. Порівняльний аналіз можливостей та обмежень форматів

У цьому підрозділі проведено структуроване порівняння попередньо розглянутих форматів за єдиним набором технічних та функціональних критеріїв. До уваги взяті такі характеристики, як типи стиснення, підтримка глибини кольору та колірних просторів, прозорості та анімації, гнучкість роботи з метаданими. Результати систематизовані у вигляді порівняльних таблиць для наочного представлення відмінностей.

Таблиця 1.1

Основні можливості

	Стиснення з втратами	Стиснення без втрат	Підтримка прозорості	Підтримка анімації
BMP	Ні	Так	Так	Ні
GIF	Ні	Так	Бінарна	Так
JPEG	Так	Ні*	Ні	Ні
PNG	Ні	Так	Так	Ні**
JPEG 2000	Так	Так	Так	Ні
WebP	Так	Так	Так	Так
HEIC	Так	Так	Так	Так
AVIF	Так	Так	Так	Так
JPEG XL	Так	Так	Так	Так

*Хоча специфікація JPEG-1 визначає режим стиснення без втрат (JPEG-LS), імплементація даного функціоналу у декодерах зустрічається вкрай рідко.

**Підтримується розширенням стандарту APNG.

Таблиця 1.2

Обмеження кольору та розміру

	Макс. глибина кольору на канал (біт)	Колірна суб-дискретизація	Максимальна роздільна здатність (n×n)	Максимальна кількість каналів
BMP	8	-	30’000*	4
GIF	-	-	65’535	1
JPEG	8	Присутня	65’535	4
PNG	16	-	2’147’483’647	4
JPEG 2000	38	Присутня	4’294’967’295	16384
WebP	8	Тільки 4:2:0	16’383	4
HEIC	12	Зазвичай 4:2:0	65’535**	5
AVIF	12	Присутня	65’535**	5
JPEG XL	32	Присутня	1’073’741’823	4099

*Хоча теоретичний ліміт ширини/висоти BMP зображення становить 2 млрд. пікселів, практичний ліміт є значно меншим оскільки заголовок формату не підтримує файли більше 4ГБ.

**Відеокодеки що лежать в основі HEIC та AVIF мають обмеження розміру 7680×4320 (8K), тому зображення розміром більше 8K використовують плитку з декількох окремо закодованих ділянок, що призводить до візуальних артефактів на межах плиток.

Таблиця 1.3

Розширені функції

	Підтримка HDR	Підтримка багатошарових зображення	Синтез шуму	Підтримка метаданих
BMP	Ні	Ні	Ні	ICC
GIF	Ні	Ні	Ні	Comment, XMP
JPEG	Ні	Ні	Ні	EXIF, IPTC, XMP
PNG	Так	Ні	Ні	tEXt, iCCP, EXIF, XMP
JPEG 2000	Так	Ні	Ні	EXIF, XMP, GeoJP2
WebP	Ні	Ні	Ні	EXIF, XMP, ICC
HEIC	Так	Ні	Ні	EXIF, XMP, IPTC, ICC
AVIF	Так	Ні	Плівкове зерно	EXIF, XMP, ICC
JPEG XL	Так	Так	Цифровий шум	EXIF, XMP, ICC

Таблиця 1.4

Технології кодування та обробки

	Алгоритм стиснення з втратами	Алгоритм стиснення без втрат	Фільтри після декодування
BMP	-	RLE	-
GIF	-	LZW	-
JPEG	DCT	-	-
PNG	-	DEFLATE	-
JPEG 2000	DWT	DWT	-
WebP	VP8	WebP Lossless	DF
HEIC	HEVC/H.265	HEVC/H.265	DF, SAO
AVIF	AV1	AV1	DF, CDEF, LRF
JPEG XL	VarDCT	Modular (predictive)	EPF, GF

- DF – Deblocking Filter (Фільтр зняття блочності)

- SAO – Sample Adaptive Offset (Зміщення адаптивне до зразків)
- CDEF – Constrained Directional Extrapolation Filter (Обмежений фільтр направленої екстраполяції)
- LRF – Loop Restoration Filter (Фільтр циклічного відновлення)
- EPF – Edge Preserving Filter (Краєзберігаючий фільтр)
- GF – Gaborish Filter (Габороподібний Фільтр)

Таблиця 1.5

Впровадження та продуктивність

	Обчислювальна складність	Підтримка у браузерах	Підтримка у ПЗ
BMP	Дуже низька	Широка	Широка
GIF	Низька	Широка	Широка
JPEG	Низька	Широка	Широка
PNG	Середня	Широка	Широка
JPEG 2000	Висока	Дуже обмежена	Дуже обмежена
WebP	Середня	Широка	Доволі широка
HEIC	Висока	Обмежена	Обмежена
AVIF	Дуже висока	Широка	Зростаюча
JPEG XL	Висока	Обмежена	Зростаюча

1.4. Постановка задачі дослідження

Проведений у попередніх підрозділах аналіз демонструє складний та динамічний ландшафт технологій стиснення растрових зображень. Сучасні формати, зокрема AVIF та JPEG XL, пропонують значні переваги в ефективності стиснення порівняно з традиційними JPEG та PNG, однак їхнє впровадження у практичну діяльність розробників, дизайнерів та звичайних користувачів відбувається повільно. Основною перешкодою є не лише інерція ринку, а й відсутність доступних, простих та водночас об'єктивних інструментів для проведення прямого порівняльного аналізу.

Існуючі програмні засоби часто є або надто складними для пересічного користувача (консольні утиліти), або не підтримують сучасні формати, або ж не надають об'єктивних числових метрик для оцінки якості, покладаючись лише на візуальне сприйняття. Крім того, пряме порівняння форматів ускладнюється тим, що шкали налаштувань якості (quality) не є еквівалентними між різними кодеками. Це призводить до того, що вибір формату часто ґрунтується на застарілих рекомендаціях, суб'єктивних оцінках або методі спроб і помилок, що не гарантує оптимального результату. Неоптимальний вибір формату призводить до збільшення обсягу веб-трафіку, уповільнення завантаження сайтів, зростання витрат на зберігання даних та потенційного погіршення візуальної якості контенту.

Таким чином, на основі проведеного аналізу виникає науково-практична задача, яка полягає у подоланні зазначеного розриву між теоретичними перевагами сучасних форматів та складністю їх практичного застосування. Для сприянню її вирішення пропонується:

1. Розробити програмне забезпечення, яке слугуватиме інструментом для проведення експериментальних досліджень. Цей інструмент повинен мати інтуїтивно зрозумілий графічний інтерфейс та реалізовувати функціонал одночасного кодування вихідного зображення в декілька цільових форматів з гнучкими налаштуваннями, з подальшим розрахунком ключових метрик (розмір файлу, об'єктивна візуальна якість, час кодування) та наданням засобів для зручного візуального порівняння результатів;
2. Використовуючи основу розробленого інструменту, провести комплексне експериментальне дослідження ефективності провідних растрових форматів (PNG, JPEG, WebP, AVIF, JPEG XL) на репрезентативних наборах даних, що включають різні типи візуального контенту.

Вирішення цієї задачі дозволить отримати нові, об'єктивні дані про продуктивність сучасних кодеків та створити програмний продукт, що має практичну цінність для широкого кола користувачів.

2 ПРОЄКТУВАННЯ СИСТЕМИ АНАЛІЗУ ЕФЕКТИВНОСТІ ФОРМАТІВ

2.1. Розробка концептуальної архітектури програмного засобу

Проєктування архітектури є ключовим етапом у розробці програмного забезпечення, що визначає його високорівневу структуру та принципи взаємодії основних компонентів. Для розроблюваної системи аналізу ефективності форматів була обрана багаторівнева архітектура, що базується на принципі розподілу відповідальності. Такий підхід дозволяє чітко розмежувати логіку користувацького інтерфейсу, основні обчислювальні процеси та взаємодію із зовнішніми залежностями, що підвищує гнучкість та спрощує подальший супровід системи.

Концептуально архітектура системи поділяється на три основні логічні рівні:

1. Рівень представлення: Реалізований за допомогою фреймворку WinUI 3, цей рівень відповідає виключно за візуалізацію даних та обробку вводу користувача. Він не містить бізнес-логіки, а лише делегує команди користувача на нижчий рівень та відображає отримані результати;
2. Рівень бізнес-логіки: Ядро системи, що координує всі операції. Цей рівень отримує запити від рівня представлення, реалізує алгоритми (зокрема, бінарний пошук для досягнення цільових параметрів), керує процесом конвертації та агрегує отримані дані для подальшого відображення;
3. Рівень взаємодії із зовнішніми інструментами: Цей рівень інкапсулює всю логіку взаємодії із зовнішніми консольними утилітами (cwebp.exe, avifenc.exe, cjxl.exe та ін.). Він відповідає за формування командних рядків, асинхронний запуск зовнішніх процесів та парсинг їх результатів.

Така архітектурна декомпозиція забезпечує слабку зв'язаність компонентів. Модифікація або заміна зовнішнього кодера потребуватиме змін лише на рівні взаємодії, не змінюючи логіку інтерфейсу чи основні алгоритми.

2.2. Моделювання функціональності та структури системи за допомогою UML-діаграм

2.2.1. Діаграма прецедентів. Діаграма прецедентів (Use case diagram) визначає функціональні вимоги до системи аналізу ефективності форматів, моделюючи її поведінку з точки зору зовнішніх акторів та варіантів використання. Вона формалізує межі системи та ідентифікує всі зовнішні сутності, що взаємодіють з нею, а також ключові функції, які система надає. На рис. 2.1 наведена діаграма прецедентів проєктованої програмної системи.

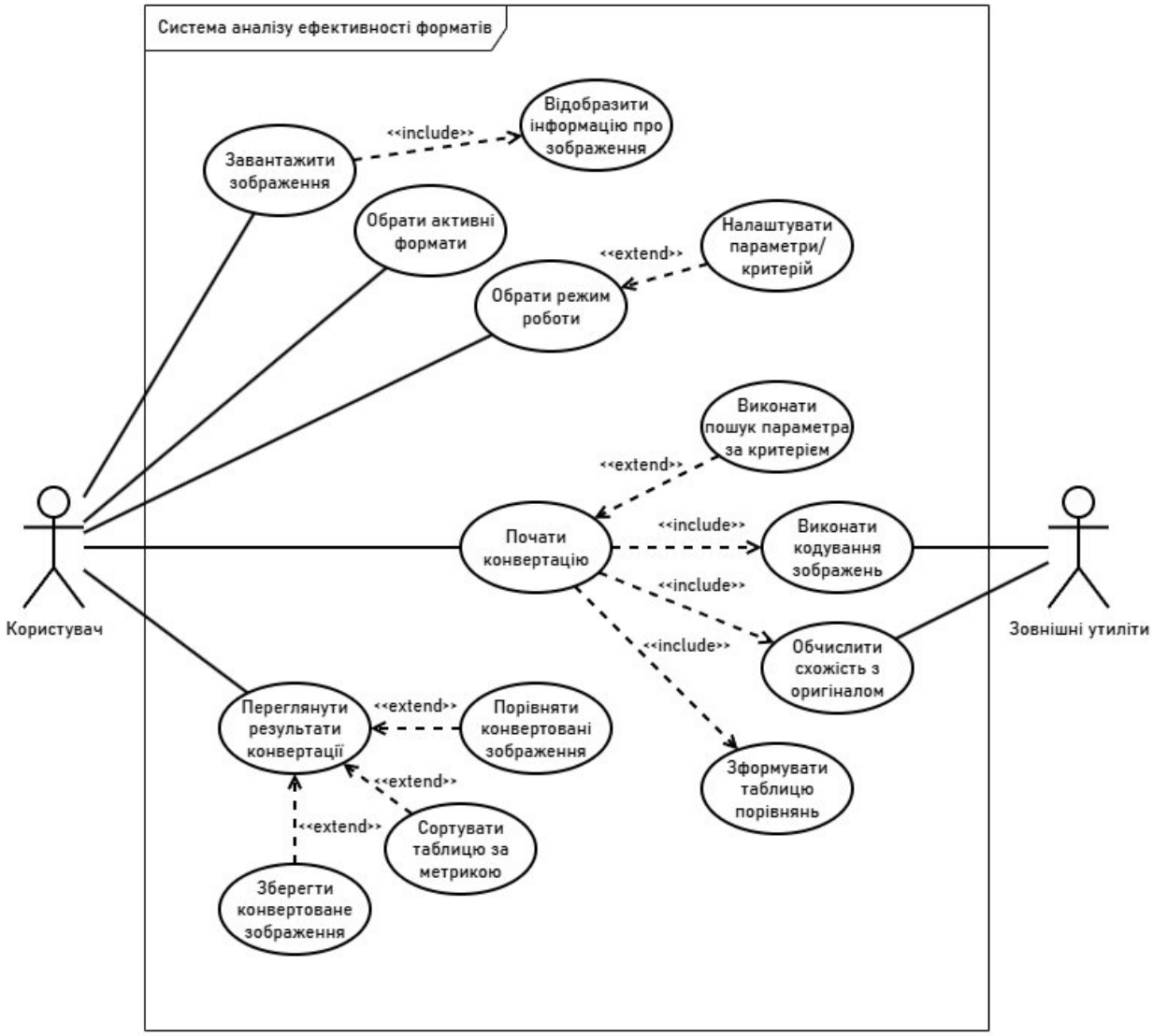

Рис. 2.1 Діаграма прецедентів

Основними акторами, що взаємодіють із системою, є:

- Користувач – первинний актор, що ініціює всі основні процеси в системі, від завантаження вихідних даних до аналізу отриманих результатів;
- Зовнішні утиліти – вторинний актор, що представляє собою сукупність зовнішніх консольних програм. Цей актор не ініціює взаємодію, а надає сервіси (кодування, обчислення метрик) у відповідь на запити системи.

Діаграма визначає такі ключові прецеденти та їхні взаємозв'язки:

- Прецедент "Завантажити зображення" представляє початкову точку взаємодії, що дозволяє користувачу обрати вихідний файл для аналізу. Цей

прецедент завжди включає "Відобразити інформацію про зображення", що забезпечує автоматичне вилучення та відображення базових метаданих файлу;

- Конфігурація аналізу представлена прецедентами "Обрати активні формати" та "Обрати режим роботи". Останній може бути розширений прецедентом "Налаштувати параметри/критерії" для введення специфічних значень (наприклад, цільового розміру файлу або рівня якості);
- Прецедент "Почати конвертацію" є центральним у системі та інкапсулює основний робочий процес. Він реалізується через обов'язкове включення трьох базових прецедентів:
    - "Виконати кодування зображень" – перетворення вихідного зображення у цільові формати за допомогою зовнішніх кодеків.
    - "Обчислити схожість з оригіналом" – розрахунок метрики візуальної якості за допомогою утиліти SSIMULACRA2;
    - "Сформувати таблицю порівнянь" – агрегація та відображення всіх отриманих метрик;
    - "Виконати пошук параметра за критерієм" – імплементує алгоритм бінарного пошуку параметру, якщо користувач обрав режим роботи з цільовим критерієм.
- Взаємодія з результатами моделюється прецедентом "Переглянути результати конвертації". Його функціональність може бути розширена прецедентами "Порівняти конвертовані зображення" (для детального візуального аналізу) та "Сортувати таблицю за метрикою" (для впорядкування результатів);
- "Зберегти конвертоване зображення" – дозволяє користувачу експортувати результат кодування зображення у файлову систему.

2.2.2. Діаграма компонентів. Діаграма компонентів (Component diagram) відображає фізичну архітектуру розробленої системи, декомпозуючи її на

модульні програмні компоненти та демонструючи залежності між ними. Компонент у даному контексті представляє собою модуль коду, що інкапсулює певну функціональність та має чітко визначений інтерфейс взаємодії. Діаграма компонентів даної системи зображена на рис 2.2.

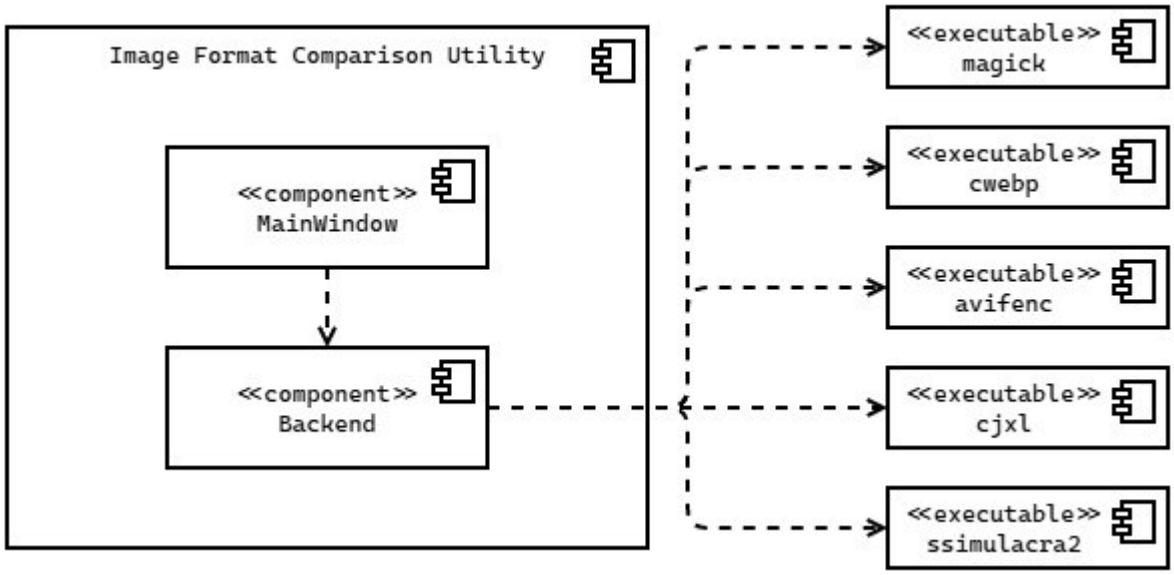

Рис. 2.2 Діаграма компонентів

Архітектура системи складається з основного програмного компонента та набору зовнішніх виконуваних утиліт.

1. Компонент-контейнер Image Format Comparison Utility представляє головний виконуваний модуль додатку, що інкапсулює внутрішню логіку системи. Він містить у собі два основні логічні підкомпоненти, що реалізують багаторівневу архітектуру:
    - Компонент MainWindow: Реалізує рівень представлення (Presentation Layer). Він інкапсулює всю логіку, пов'язану з графічним інтерфейсом користувача (GUI), включаючи обробку подій від елементів керування, валідацію вводу та візуалізацію даних. Даний компонент не містить бізнес-логіки, а делегує виконання обчислювальних операцій компоненту Backend;
    - Компонент Backend: Реалізує рівень бізнес-логіки (Business Logic Layer) та рівень взаємодії із зовнішніми інструментами. Він є ядром

системи та відповідає за координацію всіх операцій конвертації та аналізу. Його функціональність включає реалізацію алгоритмів пошуку оптимальних параметрів, формування командних рядків, керування асинхронним запуском зовнішніх процесів та агрегацію отриманих результатів.

2. Зовнішні виконувані компоненти (<<executable>>) є набором незалежних консольних утиліт, що надають спеціалізовану функціональність для кодування та аналізу зображень. Вони є фізично відокремленими від основного додатку та викликаються компонентом Backend для виконання конкретних завдань:
    - magick: Компонент, що надає інтерфейс до бібліотеки ImageMagick, використовується для уніфікації вхідних даних та виконання базових операцій з зображеннями, зокрема кодування у формат JPEG;
    - cwebp, avifenc, cjxl: Спеціалізовані компоненти-кодери, що реалізують алгоритми стиснення для форматів WebP, AVIF та JPEG XL відповідно;
    - ssimulacra2: Компонент, що реалізує алгоритм обчислення метрики візуальної якості SSIMULACRA2 для об'єктивної оцінки схожості між оригінальним та стисненим зображеннями.

Взаємозв'язки між компонентами:

- Залежність між MainWindow та Backend реалізована через прямі виклики методів (method calls) програмного інтерфейсу Backend. Компонент MainWindow ініціює операції та передає параметри, а Backend повертає оброблені дані для візуалізації;
- Залежність Backend від зовнішніх утиліт реалізована через механізм запуску окремих системних процесів. Backend формує відповідні командні рядки з параметрами, асинхронно запускає виконувані файли та обробляє їхні вихідні потоки для отримання результатів (розмір файлу, час виконання) або згенерованих файлів.

Така компонентна декомпозиція забезпечує слабку зв'язаність (loose coupling) та високу модульність системи. Інкапсуляція взаємодії із зовнішніми інструментами у компоненті Backend дозволяє легко оновлювати версії кодерів або додавати підтримку нових форматів, вимагаючи змін лише на цьому рівні архітектури.

2.2.3. Діаграма активності. Для візуалізації динамічних аспектів поведінки системи, потоків керування та послідовності виконання операцій була побудована діаграма активності (Activity Diagram). Така діаграма детально моделює основний робочий процес системи, починаючи з ініціації користувачем і закінчуючи отриманням конвертованого зображення. На рисунку 2.3 представлена діаграма активності, що описує логіку функціонування розроблюваного програмного засобу.

Діаграма структурно розділена на дві доріжки (swimlanes), що розмежовують відповідальність між двома основними акторами: "Користувач" та "Програмна система".

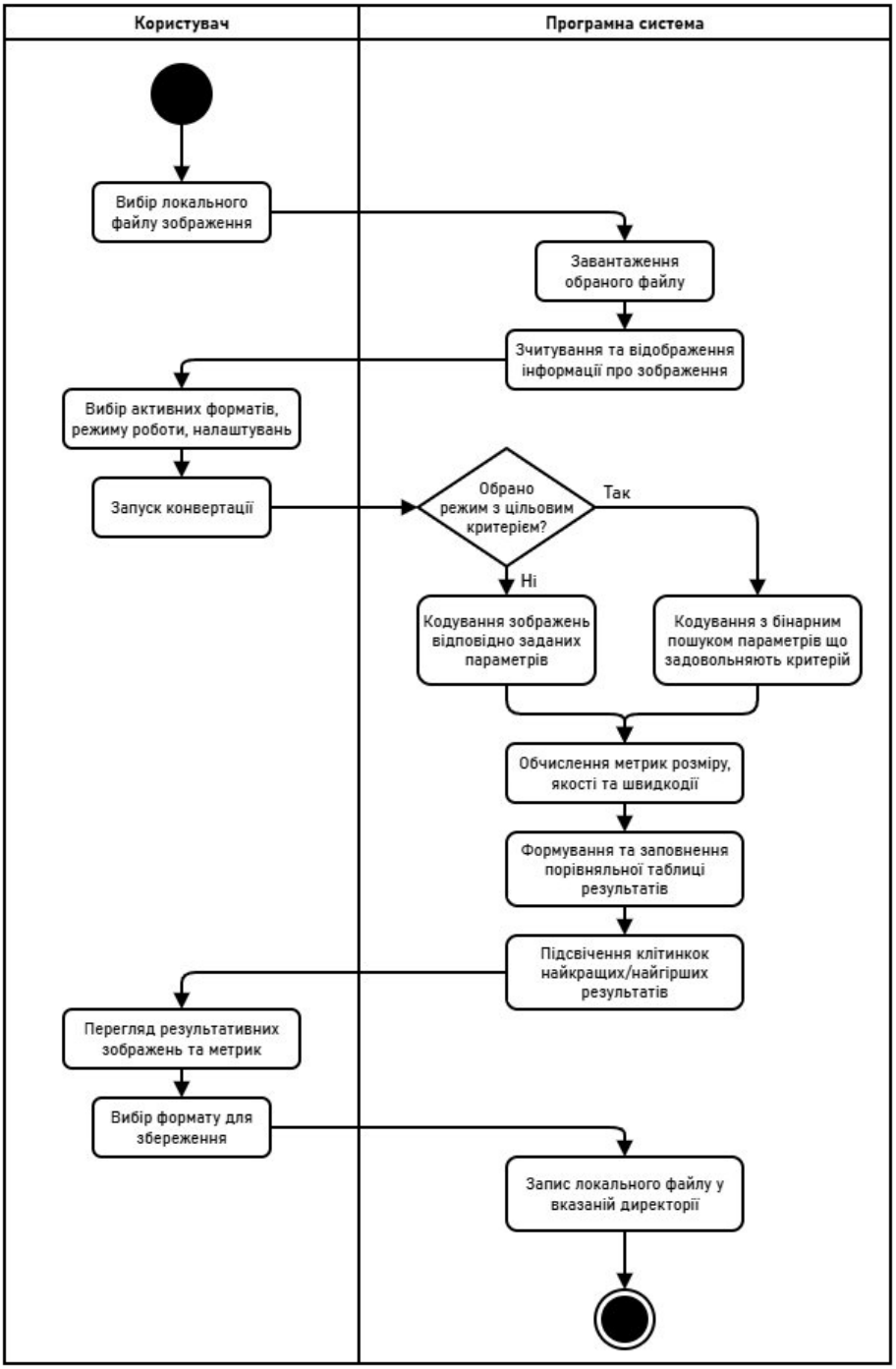

Рис. 2.3 Діаграма активності системи аналізу форматів зображень

Процес, змодельований на діаграмі, складається з таких послідовних етапів:

1. Ініціація процесу. Робочий процес починається з дії користувача. "Вибір локального файлу зображення" ініціює відповідь з боку програмної системи, яка виконує "Завантаження обраного файлу" та "Зчитування та

відображення інформації про зображення", що включає аналіз метаданих (інформація про файл, роздільну здатність, формат);

2. Конфігурація аналізу. Користувач визначає параметри кодування: "Вибір активних форматів, режиму роботи, налаштувань". Ця дія включає взаємодію з інтерфейсом та оновлення внутрішніх змінних;
3. Запуск основного процесу. Користувач ініціює основний обчислювальний процес дією "Запуск конвертації". Це є тригером для переходу до основної логіки системи;
4. Розгалуження логіки кодування. Вузол рішення "Обрано режим з цільовим критерієм?" розгалужує потік керування залежно від обраного користувачем режиму:
    - Шлях "Ні" (Ручний режим): Якщо користувач обрав режим ручних налаштувань, система виконує пряме "Кодування зображень відповідно заданих параметрів", використовуючи фіксовані значення якості та швидкості, надані користувачем;
    - Шлях "Так" (Режим з цільовим критерієм): Якщо обрано режим досягнення цільової візуальної якості або розміру файлу, система ініціює ітераційний процес "Кодування з бінарним пошуком параметрів, що задовольняють критерії". Цей алгоритм багаторазово кодує зображення з різними параметрами якості, наближаючись до заданого користувачем цільового значення.
5. Агрегація та обробка результатів. Після завершення кодування (незалежно від обраного шляху) система виконує послідовність дій:
    - "Обчислення метрик розміру, якості та швидкодії": для кожного згенерованого файлу розраховується його розмір, час витрачений на кодування, та відносна візуальна якість за допомогою алгоритму SSIMULACRA2;
    - "Формування та заповнення порівняльної таблиці результатів": агреговані дані систематизуються та відображаються у табличному вигляді;

- "Підсвічення клітинок найкращих/найгірших результатів": застосовується умовне форматування для візуальної ідентифікації найбільш та найменш ефективних результатів за кожною метрикою.

6. Взаємодія з результатами. Після відображення результатів керування повертається до користувача. Користувач може виконати "Перегляд результативних зображень та метрик", що включає візуальне порівняння та сортування таблиці. На основі аналізу користувач може виконати дію "Вибір формату для збереження", обравши конвертований формат, папку та назву файлу, після чого система виконує "Запис локального файлу у вказаній директорії".

2.3. Проєктування користувацького інтерфейсу. Проєктування графічного користувацького інтерфейсу (GUI) є важливим етапом у розробці програмного додатку, оскільки від його ефективності та інтуїтивності безпосередньо залежить зручність використання програмного продукту та доступність його функціоналу для кінцевого користувача. Основною метою при проєктуванні інтерфейсу було створення логічно структурованого, інформативного та візуально зрозумілого середовища, яке мінімізує когнітивне навантаження та дозволяє користувачеві ефективно виконувати порівняльний аналіз форматів.

Інтерфейс спроєктовано як єдине вікно, що реалізується на базі фреймворку WinUI 3. Воно умовно поділене на три основні функціональні зони, які відповідають логіці робочого процесу: завантаження вихідних даних, конфігурація параметрів та аналіз результатів. Такий підхід забезпечує послідовну та передбачувану взаємодію з системою. Макет описаного інтерфейсу наведено на рис. 2.4.

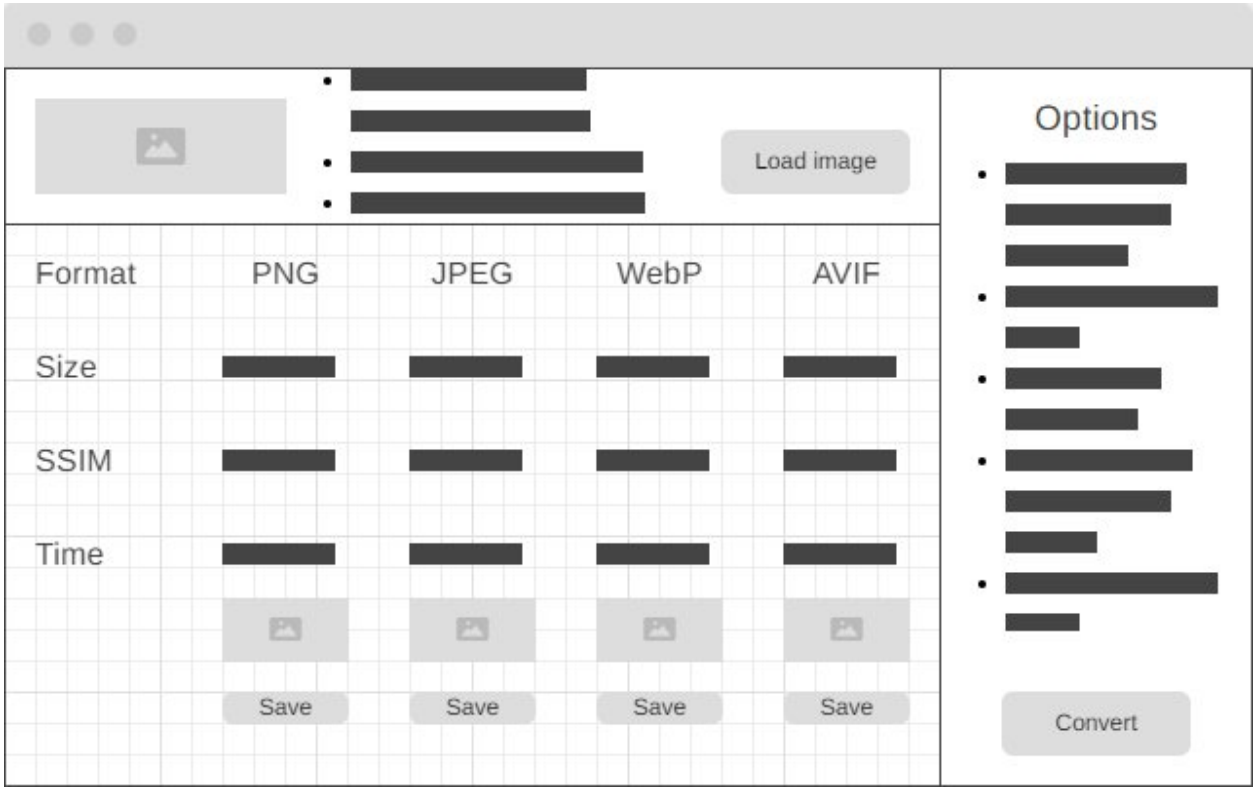

Рис 2.4 Макет користувацького інтерфейсу

Основні функціональні області, передбачені макетом:

1. Область вихідного зображення (ліва верхня частина). Ця зона є початковою точкою взаємодії та призначена для представлення об'єкта аналізу. Вона включає:
    - Кнопка для ініціації вибору та завантаження локального файлу;
    - Область мініатюри завантаженого зображення;
    - Блок з ключовими характеристиками вихідного файлу (назва, розмір, роздільна здатність, вихідний формат), що надає користувачеві контекст для подальшого порівняння.
2. Область налаштувань (права частина). Ця панель дозволяє керувати режимами конвертації та визначати їх параметри, передбачаючи наявність таких груп елементів:
    - Елемент вибору типу стиснення: дозволяє фільтрувати набір доступних для тестування форматів (наприклад, тільки формати з втратами або без втрат);

- Група перемикачів для вибору форматів: надає користувачеві можливість обирати, які саме кодеки будуть залучені до порівняння.
- Елемент для вибору режиму роботи: визначає основну логіку аналізу (ручне налаштування параметрів, досягнення цільової якості або цільового розміру);
- Динамічні елементи введення параметрів: набір полів та повзунків, видимість та призначення яких залежить від обраного режиму роботи (наприклад, введення рівня якості або цільового розміру у кілобайтах);
- Кнопка для старту процесу аналізу.

3. Область відображення результатів порівняння (центральна частина). Це основна область, де агрегуються та візуалізуються результати обчислень. Вона спроєктована у вигляді таблиці, де:
   - Стовпці представляють кожен з обраних для аналізу форматів;
   - Рядки відповідають ключовим метрикам ефективності, таким як фінальний розмір файлу, візуальна схожість, час кодування;
   - Внизу кожного стовпця міститься мініатюра відповідного конвертованого зображення та кнопка для збереження файлу зображення.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ АНАЛІЗУ ФОРМАТІВ

3.1. Вибір засобів та технологій розробки

Вибір технологічного стека для розробки системи аналізу форматів базувався на вимогах до створення нативного десктопного додатку для ОС Windows, необхідності реалізації асинхронних обчислень та забезпечення гнучкої інтеграції зі сторонніми інструментами командного рядка.

3.1.1. Платформа розробки. Для реалізації програмного ядра та логіки користувацького інтерфейсу було обрано мову програмування C# та платформу .NET. Обрана мова є статично типізованою, що мінімізує помилки на етапі компіляції, а її синтаксичні можливості, зокрема LINQ (Language-Integrated Query), оптимізують операції з колекціями даних при агрегації результатів аналізу. Модель асинхронного програмування з ключовими словами async/await є невід'ємною для реалізації, оскільки дозволяє виконувати тривалі операції кодування зображень у фоновому режимі, не блокуючи потік користувацького інтерфейсу (UI thread). Стандартна бібліотека класів (BCL) платформи .NET містить необхідний інструментарій для роботи з файловою системою (System.IO) та управління зовнішніми процесами (System.Diagnostics.Process), що дозволило реалізувати логіку взаємодії з утилітами без залучення сторонніх залежностей.

3.1.2. Технологія користувацького інтерфейсу. Для реалізації рівня представлення було обрано фреймворк WinUI 3, що входить до складу Windows App SDK. Дана технологія забезпечує створення нативного користувацького інтерфейсу згідно з принципами Fluent Design System, а використання декларативної мови розмітки XAML дозволяє реалізувати патерн розділення логіки представлення та бізнес-логіки (Code-Behind), що підвищує модульність та спрощує супровід коду. Нативність фреймворку гарантує високу продуктивність рендерингу графічного інтерфейсу, що забезпечує плавну візуалізацію та оновлення таблиці результатів.

3.1.3. Підхід до інтеграції кодеків. Архітектурно було прийнято рішення інкапсулювати логіку кодування шляхом взаємодії з зовнішніми консольними утилітами (cwebp.exe, avifenc.exe, cjxl.exe) замість прямої інтеграції нативних бібліотек. Такий підхід забезпечує слабку зв'язаність компонентів та має низку переваг: використання референтних імплементацій кодеків, спрощення процесу оновлення їх версій шляхом заміни виконуваних файлів, а також підвищення стабільності системи за рахунок ізоляції процесів кодування.

3.1.4. Вибір метрики візуальної якості. Незважаючи на широке розповсюдження, класичні метрики, такі як PSNR та SSIM, мають суттєві обмеження, що робить їх недостатньо релевантними для об'єктивного порівняння сучасних алгоритмів стиснення.

Метрика PSNR, що базується на середньоквадратичній помилці (Mean Squared Error, MSE), оцінює виключно поелементну різницю значень пікселів, повністю ігноруючи структурний контекст та особливості людської зорової системи. Тобто такі руйнівні структурні артефакти як розмиття, блочність (blocking) чи кільця (ringing), що мають дуже різний візуальний ефект, можуть мати однакове середньоквадратичне відхилення.

Метрика SSIM є значним кроком уперед, оскільки аналізує локальні структурні подібності, враховуючи яскравість, контрастність та структуру. Проте її головним обмеженням є аналіз на основі локальних вікон (local patches), що робить її менш чутливою до артефактів корельованих на великих відстанях. Крім того, SSIM може недостатньо точно оцінювати специфічні для сучасних кодеків спотворення, такі як втрата дрібних текстур (texture smearing) або колірні артефакти (color bleeding).

Тому для об'єктивної оцінки візуальної якості зображень після стиснення з втратами було обрано метрику SSIMULACRA2. На відміну від традиційних метрик, даний алгоритм розроблений спеціально для порівняння артефактів стиснення і демонструє значно вищу кореляцію з суб'єктивним сприйняттям

якості людиною (Human Visual System, HVS). Дана метрика є доречним інструментом для проведення науково-обґрунтованого порівняння продуктивності кодеків, враховуючи її валідацію на великих наборах даних з оцінками людських спостерігачів. Використання саме цієї метрики дозволяє отримати результати, що близько відповідають оцінці реальної візуальної різниці між форматами.

3.2. Реалізація ключових програмних модулів та алгоритмів

Програмна реалізація системи базується на архітектурних рішеннях, визначених у попередньому розділі. Основна логіка інкапсульована у двох ключових класах: статичному класі Backend, що відповідає за взаємодію із зовнішніми процесами та обчислення, та класі MainWindow, що реалізує логіку представлення та керування життєвим циклом додатку.

3.2.1. Реалізація модуля управління процесами кодування та аналізу

Даний модуль реалізовано у вигляді статичного класу Backend, що виконує роль сервісного шару, який абстрагує всю взаємодію із зовнішніми консольними утилітами. Такий підхід дозволяє ізолювати основну логіку додатку від специфіки виклику кожного окремого інструменту. Повний лістинг програмного коду класу представлений у Додатку А.

Ключовим механізмом взаємодії є асинхронний запуск зовнішніх процесів. Для цього використовується клас System.Diagnostics.Process та структура System.Diagnostics.ProcessStartInfo. Кожен виклик зовнішньої утиліти конфігурується з параметрами, що пригнічують створення вікна (CreateNoWindow) та перенаправляють стандартні потоки виводу (RedirectStandardOutput), що забезпечує повністю фонове виконання. Усі методи-обгортки (наприклад, WEBP, AVIF, JPEGXL) реалізовані як асинхронні (async Task<string>), а очікування завершення зовнішнього процесу виконується

за допомогою виклику await process.WaitForExitAsync(), що запобігає блокуванню потоку користувацького інтерфейсу.

Кожен такий метод формує командний рядок на основі вхідних параметрів (шлях до файлу, якість, швидкість кодування) та повертає шлях до згенерованого тимчасового файлу. Для управління проміжними даними використовується системна тимчасова директорія, шлях до якої отримується методом Path.GetTempPath(), а для уникнення колізій імен файлів генеруються унікальні імена за допомогою Path.GetRandomFileName(). Життєвий цикл цих файлів контролюється додатком, і вони видаляються після завершення сесії аналізу.

3.2.2. Реалізація алгоритмів конвертації та оптимізації параметрів. Головна логіка програми визначена методами описаними у модулі MainWindow.xaml.cs, повний лістинг коду якого представлений у Додатку Б. Основний робочий процес системи реалізовано в методі ExecuteConversion, який координує операції кодування та аналізу. Залежно від обраного користувачем режиму, застосовуються різні стратегії визначення параметрів стиснення.

У режимі ручних налаштувань (convertMode.MANUAL) параметри якості та швидкості, задані користувачем, безпосередньо передаються у відповідні методи класу Backend.

У режимах досягнення цільових показників (convertMode.SIZE та convertMode.QUALITY) імплементовано алгоритм бінарного пошуку для ефективного знаходження оптимального значення параметра якості (quality). Алгоритм функціонує наступним чином:

1. Ініціалізується діапазон пошуку для параметра якості, зазвичай [0, 100];
2. На кожній ітерації виконується тестове кодування зображення зі значенням якості, що відповідає середині поточного діапазону;
3. Отриманий результат (розмір файлу або показник SSIMULACRA2) порівнюється з цільовим значенням;

4. Залежно від результату порівняння, діапазон пошуку звужується вдвічі (зміщується верхня або нижня межа);
5. Процес повторюється доти, доки розмір діапазону не стане мінімальним (різниця між межами дорівнює 1), що свідчить про знаходження найближчого можливого значення.

Такий підхід дозволяє знаходити оптимальний параметр за логарифмічний час (O(log N), де N – діапазон значень якості), що є значно ефективнішим за лінійний перебір.

Після визначення фінальних параметрів для кожного формату виконується остаточне кодування, під час якого вимірюються ключові метрики: час виконання за допомогою класу System.Diagnostics.Stopwatch та розмір отриманого файлу через System.IO.FileInfo.Length.

3.3. Реалізація користувацького інтерфейсу та логіки взаємодії.

Рівень представлення системи реалізовано з використанням технології WinUI 3, де статична структура та вигляд користувацького інтерфейсу визначаються декларативно за допомогою мови розмітки XAML у файлі MainWindow.xaml. Такий підхід дозволяє чітко відокремити візуальну складову від програмної логіки. Для забезпечення візуальної узгодженості з операційною системою та покращення користувацького досвіду, в XAML-розмітці активно використовуються системні тематичні ресурси (ThemeResource) – це дозволяє інтерфейсу додатку автоматично адаптуватися до налаштувань теми Windows, змінюючи кольорову палітру (світла/темна тема) та акцентний колір відповідно до персональних налаштувань користувача. Початковий вигляд вікна програмного додатку представлений на рис. 3.1.

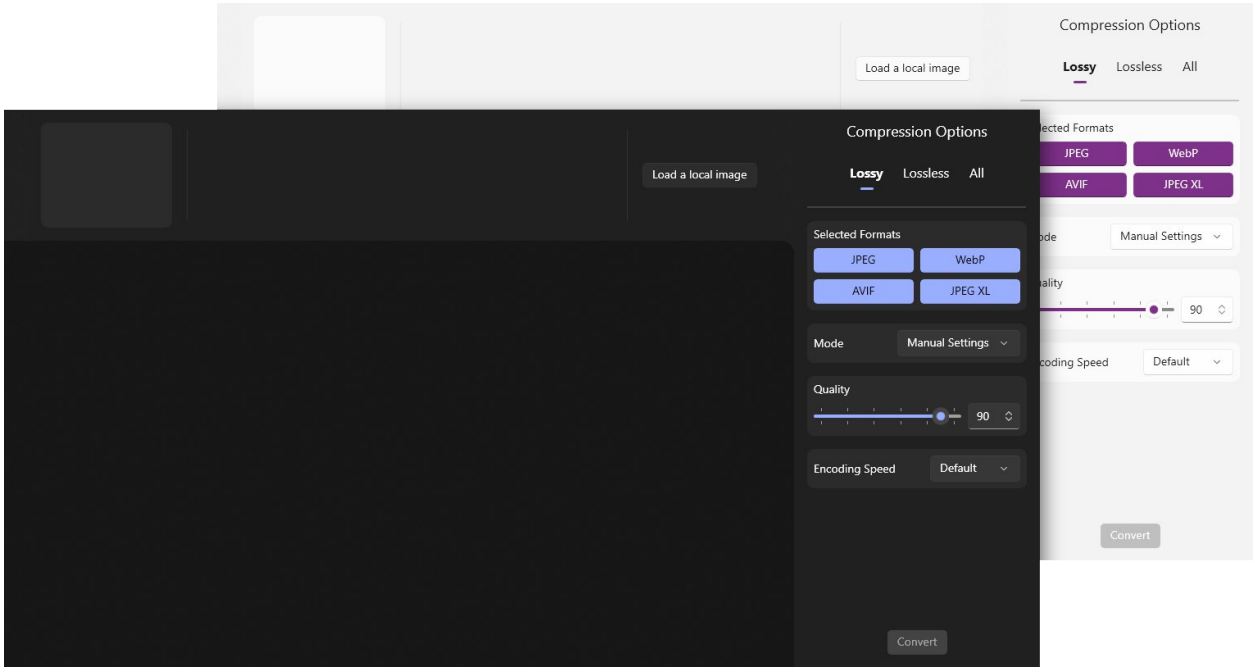

Рис. 3.1 Інтерфейс програми з різними темами та акцентами

Програмна логіка, що керує поведінкою цих елементів, інкапсульована у відповідному класі MainWindow.xaml.cs. Цей клас відповідає за обробку подій від елементів керування, ініціацію операцій у бізнес-логіці (Backend), а також за динамічне відображення та оновлення даних в інтерфейсі користувача.

3.3.1. Ініціалізація процесу аналізу та відображення вихідних даних. Взаємодія користувача з системою починається з вибору локального файлу зображення. Ця функціональність реалізована через обробник події ButtonLoad_Click, який ініціює системний діалог FileOpenPicker. Для інтеграції діалогового вікна з вікном додатку використовується механізм ініціалізації через дескриптор вікна (InitializeWithWindow.Initialize), що є стандартною практикою для додатків WinUI 3.

Після успішного вибору файлу виконується асинхронний метод DisplayExifInfo. Цей метод викликає утиліту exiftool.exe для вилучення метаданих зображення та динамічно заповнює елемент GridFileInfo відповідними значеннями. Ця інформаційна панель надає користувачеві контекст про вихідний файл, відображаючи наступні параметри:

- Name: Ім'я файлу без розширення;
- File size: Загальний розмір файлу на диску, відформатований для читабельності (наприклад, "2.54 MB");
- Resolution: Роздільна здатність зображення – ширина та висота у пікселях (наприклад, "3072 × 2304") та загальна кількість у мегапікселях;
- Image format: Формат файлу за MIME-типом, тобто визначений його вмістом, незалежно від його розширення;
- Додаткова інформація: Специфічний для формату параметр, що надає додаткові технічні деталі:
    - Для JPEG: Відображається приблизна оцінка якості збереження, отримана за допомогою опції JpegQualityEstimate утиліти ExifTool, а також інформація про колірну субдискретизацію (наприклад, "90 / 100, quarter color resolution (YCbCr 4:2:0)"). Функція JpegQualityEstimate аналізує таблиці квантування JPEG файлу для визначення налаштування якості, за яким зображення було збережене;
    - Для PNG: Відображається колірна модель та глибина кольору (наприклад, "RGBA 32-bit" або "Indexed 8-bit"), що характеризує спосіб представлення колірних даних;
    - Для WebP: Відображається тип стиснення ("Lossless" або "Lossy") та, у випадку стиснення з втратами, використовуваний колірний простір;
    - Для BMP: Відображається тип стиснення, якщо він застосовувався (наприклад, RLE), або його відсутність.

Приклади вигляду верхньої панелі після завантаження файлу зображення представлені на рис. 3.2.

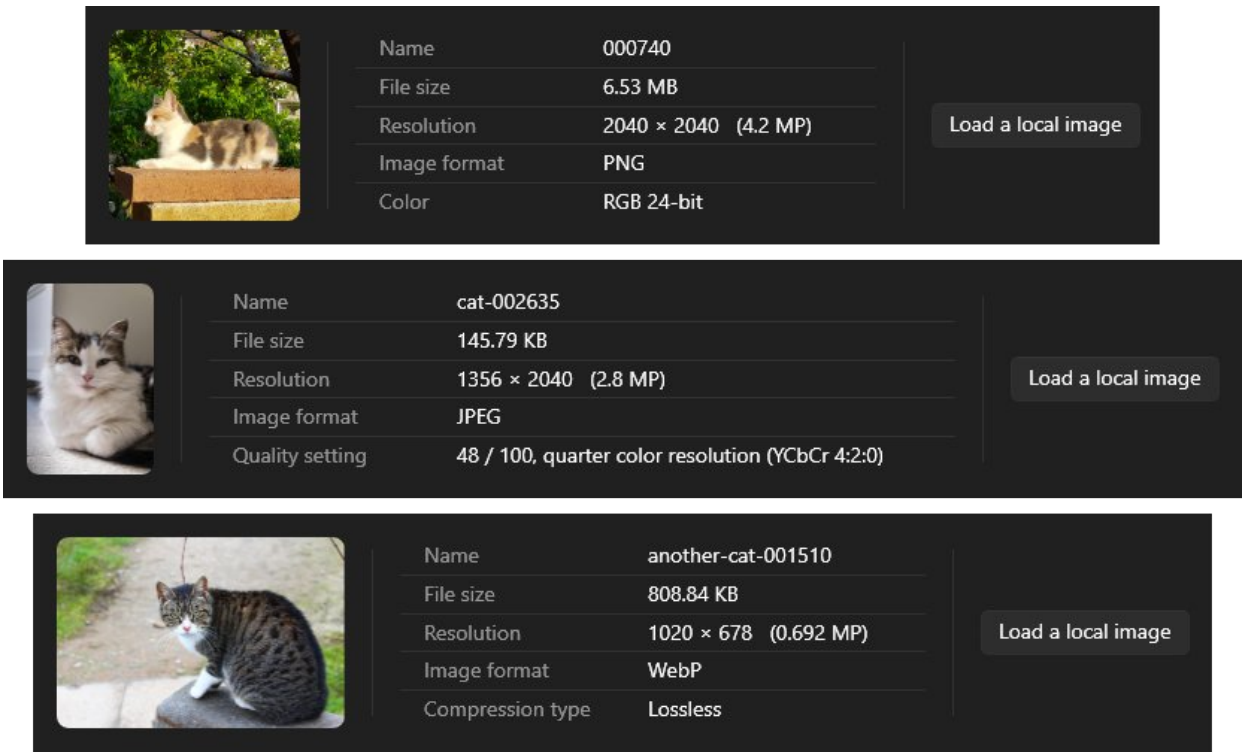

Рис. 3.2 Панель вихідного зображення з інформацією

Для подальших операцій порівняння та обчислення якості вихідне зображення, незалежно від його початкового формату, конвертується у еталонний формат PNG без втрат за допомогою утиліти ImageMagick. Це створює уніфіковану, нестиснену базу для всіх подальших операцій кодування, забезпечуючи коректність та об'єктивність порівняння результатів.

3.3.2. Конфігурація параметрів аналізу. Панель налаштувань реалізована як набір взаємопов'язаних елементів керування, що дозволяють користувачеві гнучко конфігурувати параметри аналізу.

Фільтрація форматів: Елемент SelectorBar (SelectorBarLoss) дозволяє користувачеві фільтрувати набір доступних для тестування форматів за бажаним типом стиснення ("Lossy", "Lossless", "All"). Обробник події SelectorBar_SelectionChanged динамічно перебудовує сітку GridFormatSelection, заповнюючи її елементами ToggleButton для відповідних форматів. Стан кожної кнопки зберігається у масиві selectedFormats, що дозволяє зберігати вибір користувача при перемиканні режимів.

Вибір режиму роботи та адаптивний інтерфейс: Вибір основного режиму роботи ("Manual Settings", "Target Quality", "Target Size") здійснюється через елемент ComboBox (ComboBoxMode). Обробник події ComboBoxMode_SelectionChanged реалізує логіку адаптивного інтерфейсу: він керує властивістю Visibility контейнерів з елементами налаштувань (GridSettingQuality, GridSettingTargetSize, GridSettingTargetQuality), відображаючи лише ті, що є релевантними для поточного режиму.

Введення параметрів: Введення числових параметрів (якість, цільовий розмір) реалізовано за допомогою зв'язаних елементів Slider та NumberBox, що забезпечує як інтуїтивне візуальне, так і точне введення значень. Їхні значення синхронізуються через обробники подій ValueChanged. Для налаштування швидкості кодування використовується ComboBox (ComboBoxEffort). Обраний індекс ("Fast", "Default", "Slow", "Very slow") трансформується у конкретне числове значення effort для кожного кодека за допомогою попередньо визначеної матриці відповідності effortMap, що враховує різні діапазони параметрів для утиліт cwebp, avifenc та cjxl.

Зразки вигляду бокової панелі налаштувань показані на рис. 3.3.

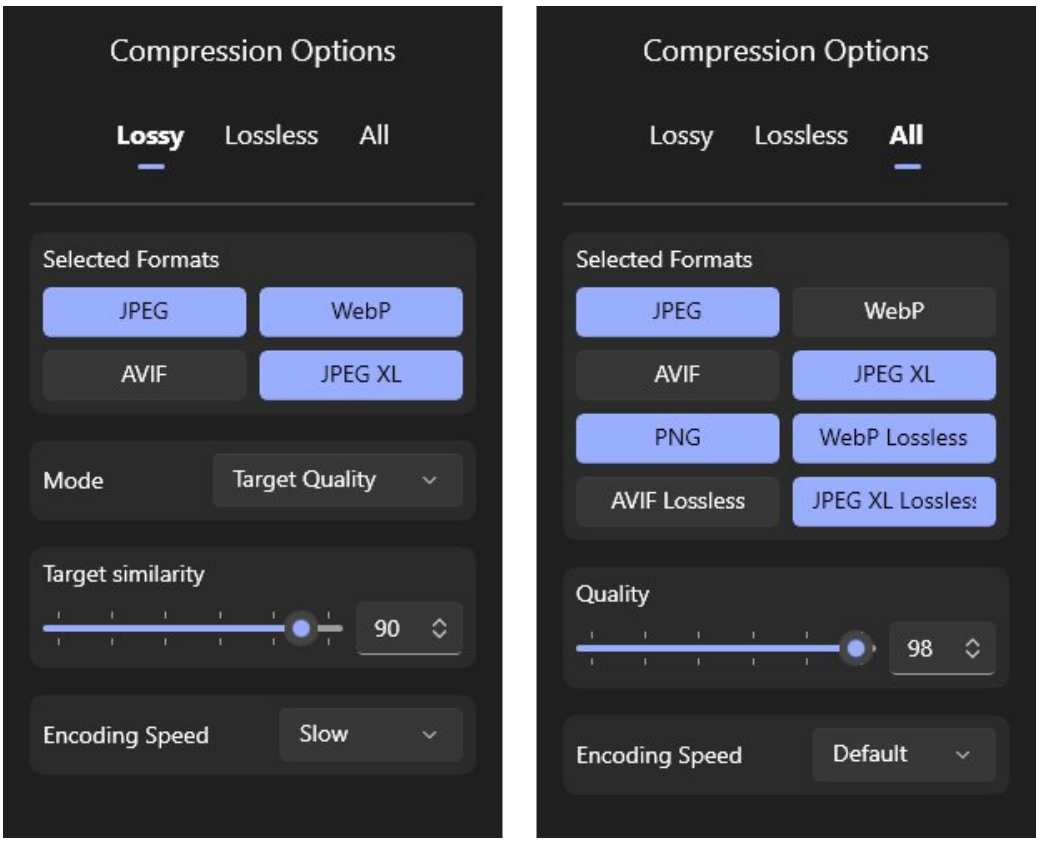

Рис. 3.3 Бокова панель налаштувань

3.3.3. Формування та візуалізація порівняльної таблиці результатів. Інтерфейс таблиці результатів є динамічним та генерується програмно. Перед кожним новим аналізом вміст та структура елемента Grid повністю очищуються. Далі, на основі кількості обраних для порівняння форматів, програмно створюється відповідна кількість стовпців (ColumnDefinition).

Для кожного формату створюється окремий дочірній контейнер Grid, який заповнюється елементами TextBlock для відображення розрахованих метрик (розмір, схожість, час), елементом Image для мініатюри та кнопкою Button для збереження.

Після завершення всіх операцій кодування та обчислень застосовується умовне форматування для візуалізації результатів. Система аналізує масиви отриманих метрик, визначаючи мінімальні та максимальні значення в кожній категорії. На основі позиції конкретного значення в цьому діапазоні, елемент Border що знаходиться під текстовим блоком метрики, присвоюється

відповідний колірний ресурс (SystemFillColorSuccessBrush, SystemFillColorCautionBrush або SystemFillColorCriticalBrush). Додатково, система визначає та м'яко підсвічує стовпець з пропонованим форматом, що представляє найліпше співвідношення якість/розмір, яке вираховується евристичним алгоритмом, представленим на формулі нижче.

$$score - k \times log_n size \tag{3.1}$$

де score – оцінка візуальної якості SSIMULACRA2;

k – підібраний коефіцієнт;

size – розмір файлу, [байт].

Зразок вигляду таблиці порівняння після конвертації наведений на рис. 3.4.

Рис. 3.4 Порівняльна таблиця результатів конвертування

3.3.4. Взаємодія з результатами конвертації. Сортування таблиці: Для кожного рядка що містить метрику у таблиці результатів передбачено кнопку сортування, а обробник події SortColumns реалізує логіку перевпорядкування стовпців. Алгоритм отримує поточний порядок стовпців, зчитує відповідні

метрики, сортує масив індексів стовпців за допомогою алгоритму бульбашкового сортування, а потім застосовує новий порядок, змінюючи властивість Grid.Column для кожного стовпця.

Деталізований перегляд та порівняння зображень: Кожна мініатюра у таблиці результатів є інтерактивною. Обробник події Tapped (OpenFullPreview) активує модальне вікно перегляду, завантажуючи в елемент BorderPopup повнорозмірне зображення з кешу (fullImages). Внизу цього вікна розташований StackPanel, який динамічно заповнюється кнопками, що відповідають усім успішно конвертованим форматам та вихідному зображенню. Обробник події SelectFormatImage дозволяє користувачеві миттєво перемикати відображуване зображення, не закриваючи вікно. Елемент ScrollViewer забезпечує функціонал масштабування та панорамування, причому поточний стан трансформації (ZoomFactor, HorizontalOffset, VerticalOffset) зберігається при перемиканні між зображеннями, дозволяючи легко помічати візуальні відмінності між конвертованими зображеннями різних форматів за однакового масштабу та позиції.

Відображення повновіконного перегляду конвертованих зображень показано на рис. 3.5.

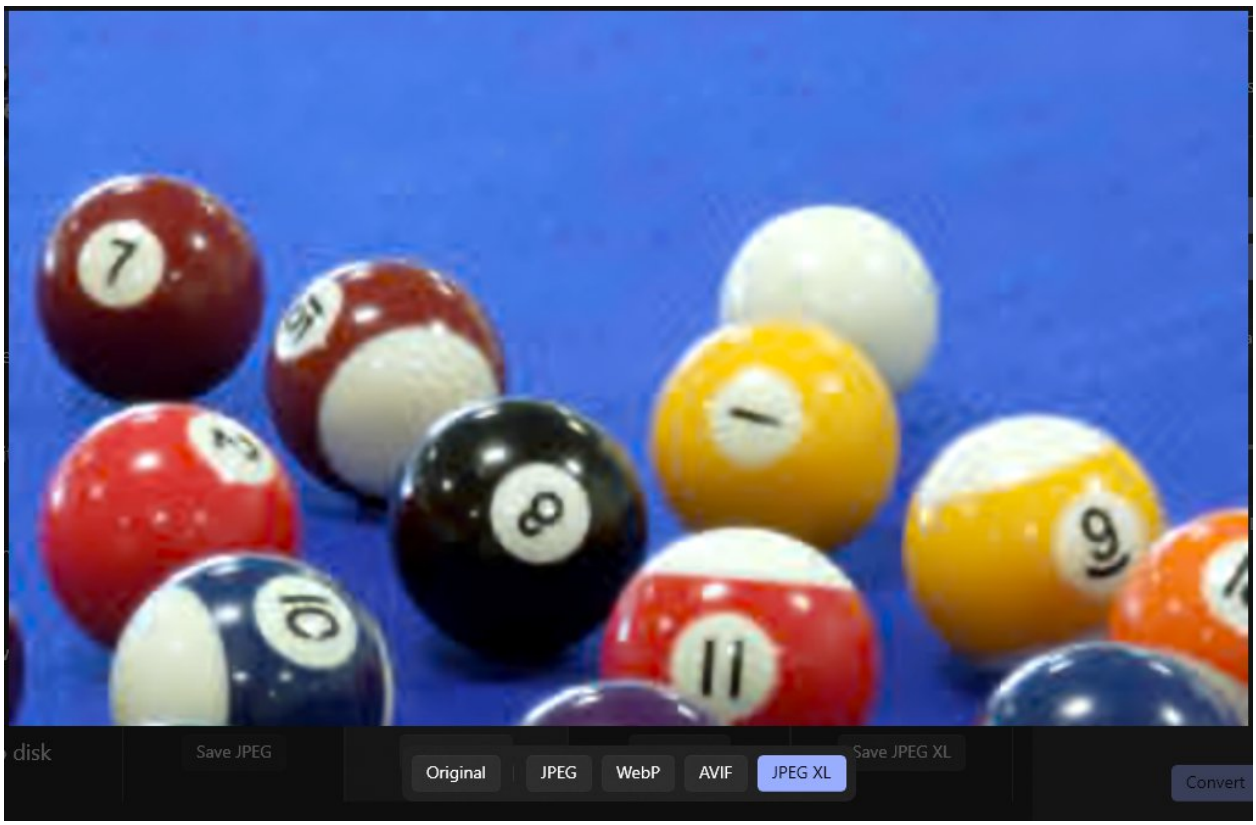

Рис. 3.5 Модальне вікно перегляду зображень.

Збереження результатів на диск: Для фіксації результатів конвертації кожен стовпець таблиці містить кнопку збереження, функціонал яких реалізовано в обробнику події SaveFormatImage. При активації кнопки система використовує клас FileSavePicker з Windows App SDK для ініціалізації системного діалогу збереження файлу. Після вибору користувачем кінцевого шляху збереження, програма виконує операцію копіювання, переміщуючи відповідний тимчасовий файл, шлях до якого зберігається у словнику convertedImagePaths, до вказаної користувачем директорії за допомогою методу StorageFile.CopyAndReplaceAsync.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ФОРМАТІВ

4.1. Організація та методика проведення експерименту

Метою експерименту є отримання кількісних даних щодо ефективності алгоритмів стиснення, що лежать в основі форматів PNG, JPEG, WebP, AVIF та JPEG XL, у різних сценаріях використання. Для забезпечення об'єктивності результатів було стандартизовано апаратно-програмне середовище та визначено чіткі критерії оцінювання.

4.1.1. Апаратно-програмне середовище. Усі експериментальні обчислення проводились на єдиній апаратній конфігурації для усунення впливу варіацій продуктивності обладнання на результати вимірювань часу кодування.

Апаратна конфігурація тестового стенда:

- Центральний процесор (CPU): AMD Ryzen 5-4500 (6 ядер, 6 потоків);
- Оперативна пам'ять (RAM): 16 ГБ DDR4;
- Операційна система: Microsoft Windows 10 Pro.

Програмне середовище складалося з набору референтних консольних утиліт, що є стандартними реалізаціями відповідних кодеків, та спеціалізованого інструменту для оцінки візуальної якості:

- ImageMagick: Використовувався для кодування у традиційні формати JPEG та PNG;
- cwebp: Офіційна утиліта з бібліотеки libwebp для кодування у формат WebP;
- avifenc: Офіційна утиліта з бібліотеки libavif для кодування формату AVIF;
- cjxl: Офіційна утиліта з бібліотеки libjxl для кодування у формат JPEG XL;
- SSIMULACRA2: Інструмент для оцінки візуальної схожості стиснених зображень з оригіналом.

Використання офіційних референтних реалізацій кодерів гарантує, що отримані результати відображають продуктивність саме базових алгоритмів, мінімізуючи вплив сторонніх оптимізацій або обмежень.

4.1.2. Формування вибірки даних для дослідження. Для забезпечення всебічного аналізу була сформована репрезентативна вибірка зображень, що охоплює два основні типи візуального контенту, які по-різному впливають на ефективну роботу алгоритмів стиснення.

- Фотографічний контент: Для тестування на фотографіях було використано зображення з відкритих наборів даних TESTIMAGES, DIV2K та Flickr2K. Ці набори містять тисячі зображень високої роздільної здатності з широкою різноманітністю сцен, що включають природні ландшафти, портрети, архітектуру та макрозйомку. Така варіативність забезпечує наявність у вибірці зображень зі складними текстурами, плавними градієнтами, дрібними деталями та різними колірними палітрами, що є типовим для фотографічного контенту;
- Синтетичний та графічний контент: Для аналізу ефективності на веб-графіці було використано набір даних WebScreenshots. Цей набір складається зі знімків екрана веб-сторінок і характеризується наявністю чітких меж, текстових елементів, іконок, діаграм та великих областей суцільного кольору. Такі характеристики є типовими для синтетичних зображень і створюють інші виклики для алгоритмів стиснення порівняно з фотографіями.

Всі вхідні зображення мають формат PNG, гарантуючи високу якість та відсутність попередніх артефактів стиснення.

4.1.3. Критерії та метрики оцінювання ефективності. Для комплексного аналізу продуктивності форматів було визначено набір основних та похідних кількісних метрик.

Основні метрики, що фіксувалися безпосередньо під час кодування:

- Розмір файлу (байт): Показник конвертованого файлу, що безпосередньо відображає ефективність стиснення;
- Час кодування (мілісекунди): Метрика, що характеризує обчислювальну складність та ресурсоємність процесу кодування;
- Візуальна якість (оцінка SSIMULACRA2): Оцінка метрики SSIMULACRA2, отримана у порівнянні оригінального та конвертованого файлів зображення після стиснення з втратами, та демонструє структурне відхилення між ними викликане артефактами. Значення метрики має діапазон (-∞;100], де 100 – математична відсутність втрат, 90 – візуальна відсутність втрат якості, 80 – незначні помітні втрати, і т.д.

Похідні метрики, що розраховувались для нормалізації результатів та уможливлення порівняння зображень різної роздільної здатності:

- Ступінь стиснення (біт на піксель, bpp): Нормалізований показник розміру файлу, що розраховується за формулою:

$$\frac{size \times 8}{w \times h} \tag{4.1}$$

де size – розмір файлу конвертованого зображення, [байт];

w – ширина зображення, [пікселі];

h – висота зображення, [пікселі].

Дозволяє об'єктивно порівнювати ефективність стиснення незалежно від розмірів зображення.

- Швидкість кодування (пікселів на секунду, px/s): Нормалізований показник продуктивності кодера, що розраховується як:

$$\frac{w \times h}{t} \tag{4.2}$$

де w – ширина зображення, [пікселі];

h – висота зображення, [пікселі];

t – час витрачений на кодування зображення, [мілісекунди].

Дозволяє порівнювати швидкодію кодерів на зображеннях різного розміру.

4.2 Аналіз результатів експериментального дослідження

У даному підрозділі представлено та проаналізовано емпіричні дані, отримані в ході виконання експериментів з кодування репрезентативної вибірки зображень. Аналіз проводиться окремо для режимів стиснення без втрат та з втратами, з урахуванням типу візуального контенту та інших ключових параметрів. Результати візуалізовані за допомогою діаграм та графіків для наочної демонстрації виявлених закономірностей.

4.2.1. Нееквівалентність параметрів якості та їх вплив на порівняльний аналіз. При проведенні порівняльного аналізу алгоритмів стиснення з втратами наївним способом є пряме зіставлення результатів, отриманих за однакових числових значень параметра якості (quality). Однак, даний параметр не є стандартизованим і імплементація його значень різниться між алгоритмами, що призводить до нееквівалентних результатів як за візуальною якістю, так і за розміром файлу. Демонстрація цієї невідповідності наведена залежністю візуальної якості (за метрикою SSIMULACRA2) від налаштування quality кодера (рис. 4.1).

Рис. 4.1 Залежність візуальної якості від параметра кодека

Як видно з графіка, криві для різних форматів мають різний характер:

- JPEG: Демонструє майже лінійну залежність в діапазоні quality від 100 до 50, після чого відбувається різке падіння якості;
- WebP: Показує більш рівномірну, близьку до лінійної, залежність у всьому діапазоні;
- AVIF: Має криву, що нагадує кореневу функцію, де значні зміни якості відбуваються при низьких значеннях quality, а при високих значеннях крива стає більш пологою;
- JPEG XL: Шкала якості розроблена таким чином, щоб наближатись до традиційної шкали JPEG, що підтверджується схожістю їх кривих.

Невідповідність шкал якості безпосередньо впливає і на отриманий розмір файлу. На рис. 4.2 показано залежність ступеня стиснення (біт на піксель) від

параметра quality. За однакових налаштувань якості (для великих зображень), видно як AVIF нібито показує найменший розмір файлу за налаштувань якості менше 40, та найбільший розмір за якості більше 65 відповідно.

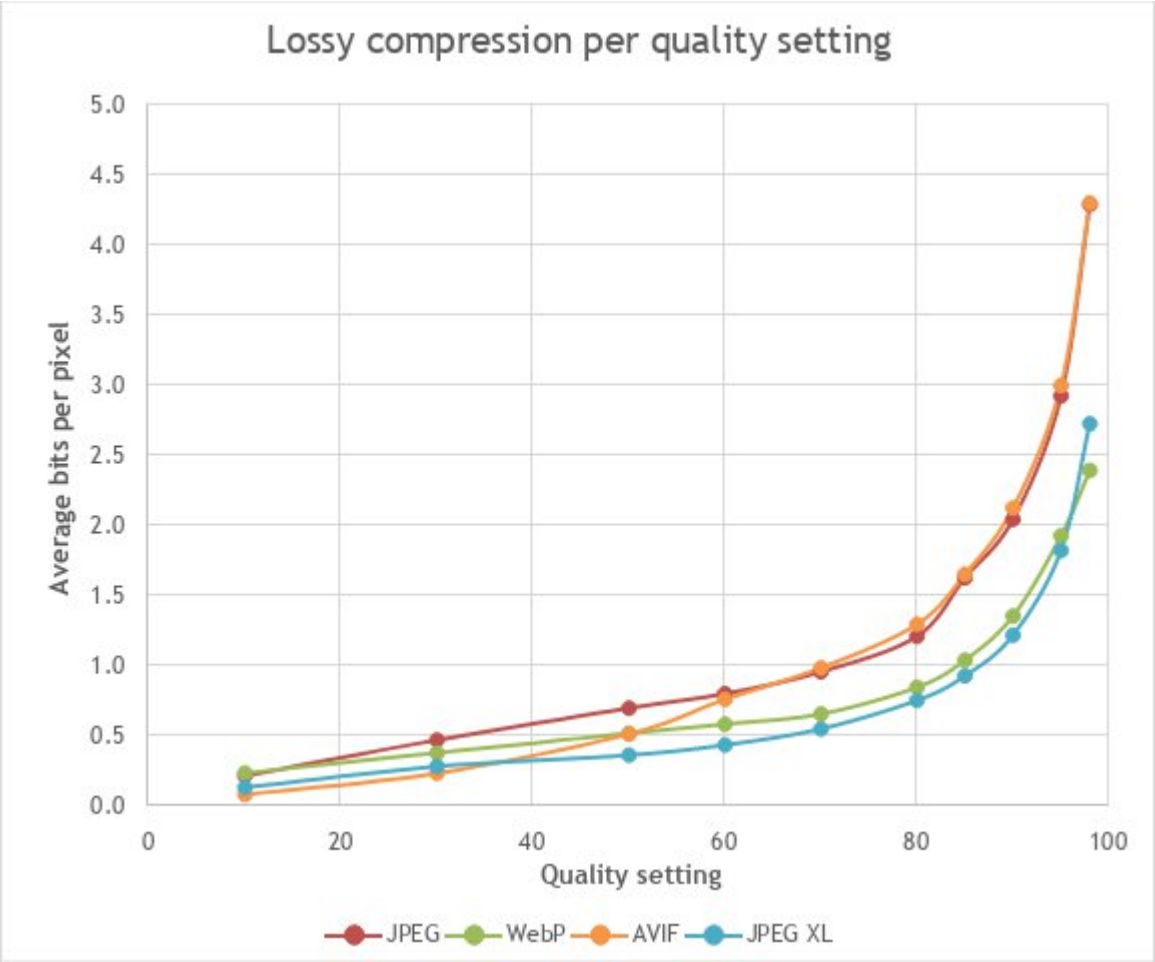

Рис. 4.2 Залежність розміру файлу від параметра кодека

Таким чином, експериментально представлено, що порівняння форматів за фіксованим значенням quality є методологічно некоректним і не відображає їх реальну відносну ефективність. Для отримання об'єктивних та зіставних результатів необхідно оцінювати отриману візуальну якість кожного формату за уніфікованою візуальною метрикою (в даному дослідженні – SSIMULACRA2), та порівнювати інші показники (наприклад, розмір файлу або час кодування) для зображень з еквівалентними рівнями візуальної якості. Саме такий підхід застосовується у подальшому аналізі для коректного зіставлення ефективності форматів.

4.2.2. Ефективність стиснення без втрат (Lossless Compression). Стиснення без втрат є критично важливим для сценаріїв, де вимагається повне збереження вихідної інформації, наприклад, при архівуванні, медичній візуалізації або роботі з майстер-копіями графіки.

Аналіз ступеня стиснення, представлений на гістограмі (рис. 4.3), демонструє значну перевагу сучасних форматів над класичним PNG. Формат JPEG XL показує найвищу ефективність, забезпечуючи зменшення розміру файлу в середньому на ~30% для фотографій та на ~50% для скріншотів порівняно з PNG. WebP посідає друге місце, також суттєво випереджаючи PNG. Водночас, формат AVIF у режимі без втрат демонструє найменшу ефективність, у середньому незначно поступаючись навіть формату PNG, що робить його використання для цього сценарію недоцільним.

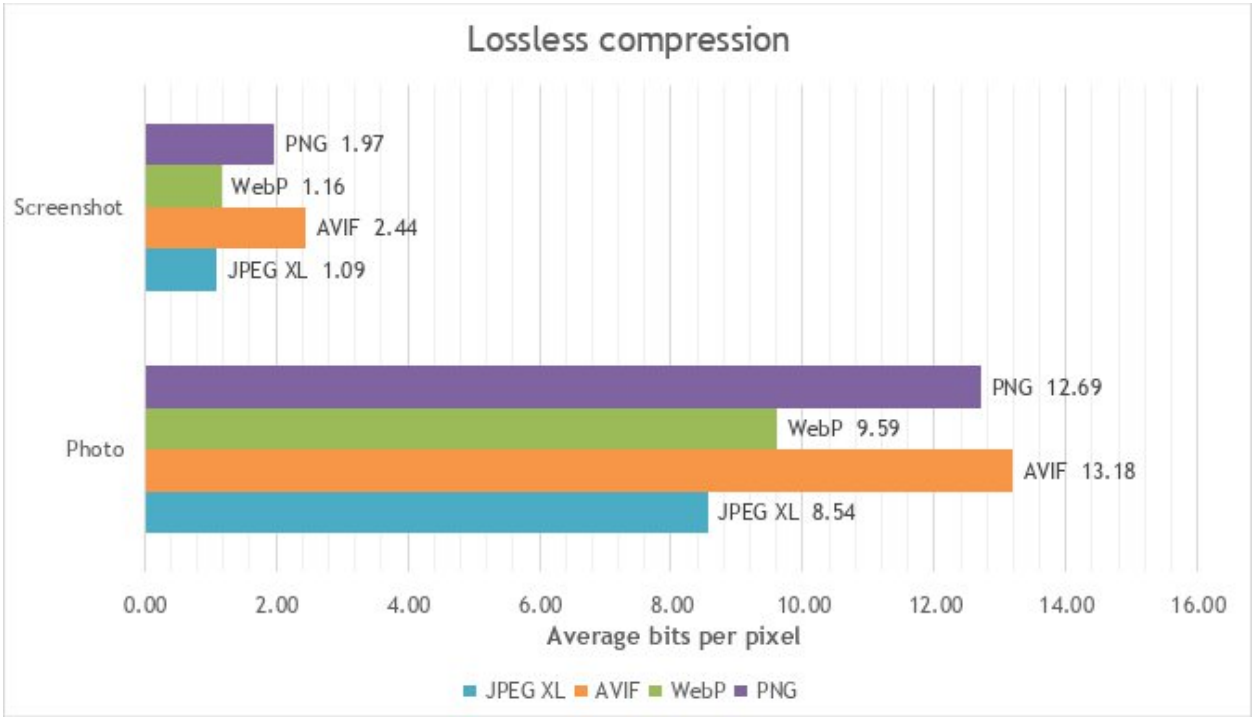

Рис. 4.3 Середня ефективність стиснення без втрат

Окрім ступеня стиснення, важливим параметром є продуктивність кодера. На рисунках 4.4 та 4.5 представлено співвідношення між швидкістю кодування (пікселів на секунду) та досягнутим ступенем стиснення (біт на піксель), середніх для кожного рівня налаштувань швидкості (effort) кодеків.

Для фотографічного контенту (рис. 4.4) JPEG XL демонструє найкращий компроміс, досягаючи найвищого стиснення при конкурентній швидкості на всіх рівнях effort. WebP є хорошою альтернативою, коли пріоритетом є швидкість, а не максимальне стиснення. Ефективність стиснення AVIF значно поступається іншим форматам, особливо на налаштуваннях швидкості вище середнього.

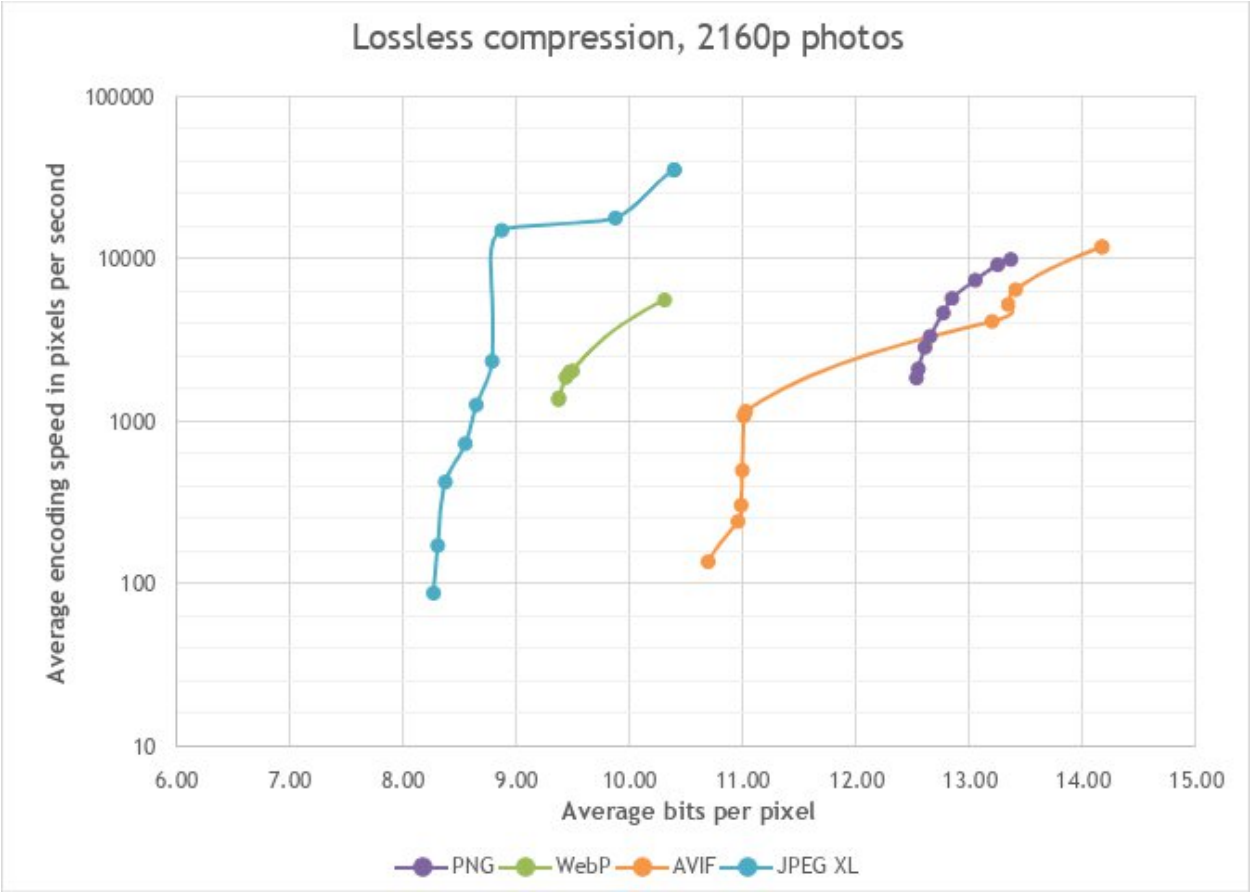

Рис. 4.4 Швидкість відносно ефективності стиснення фото без втрат

Для синтетичних зображень (рис. 4.5) WebP досі показує найкращий баланс швидкості та стиснення. JPEG XL є менш ефективним за тієї ж швидкості, але може досягати кращого стиснення ціною значно більш тривалого кодування. PNG та AVIF, відповідно, демонструють гірші співвідношення цих параметрів.

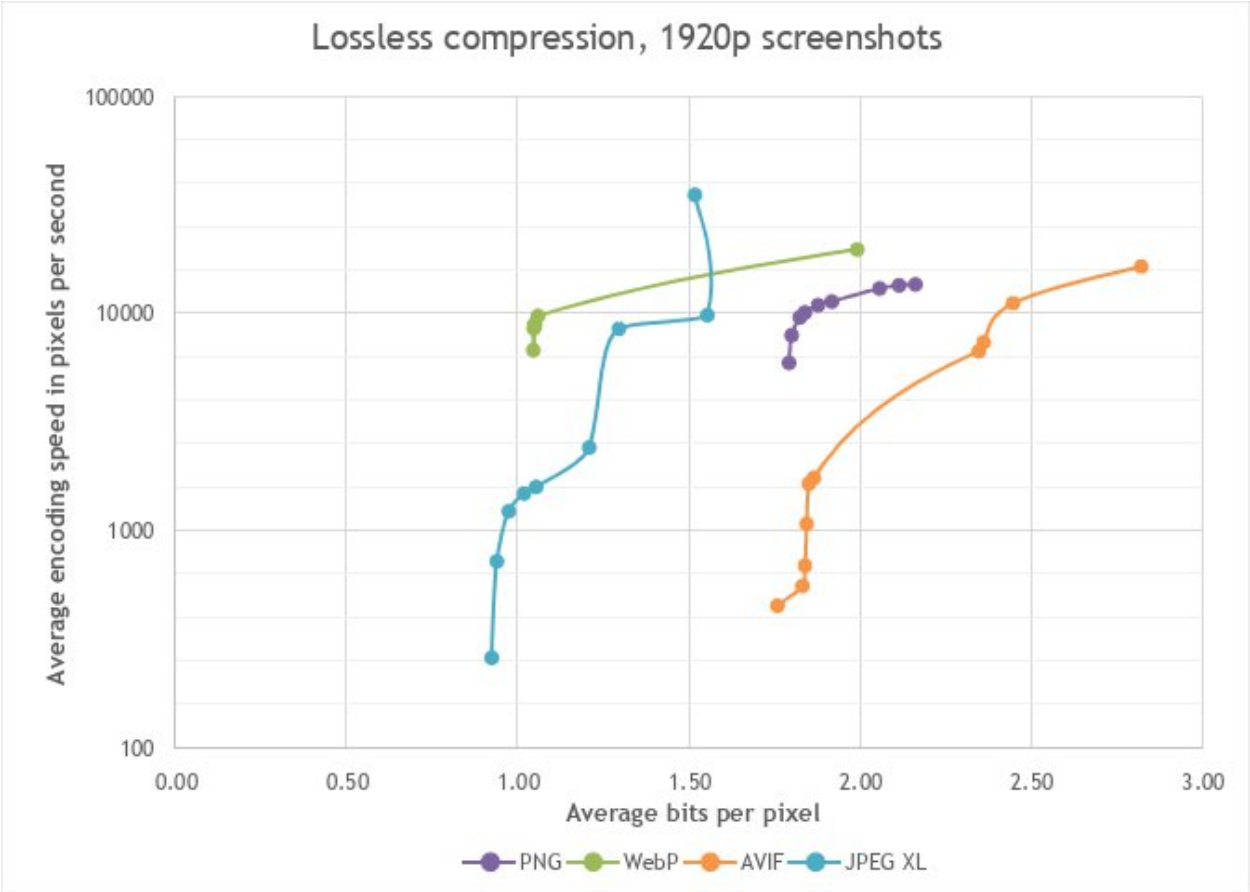

Рис. 4.5 Швидкість відносно ефективності стиснення скріншотів без втрат

4.2.3. Ефективність стиснення з втратами (Lossy Compression). Стиснення з втратами є переважним підходом для розповсюдження візуального контенту в мережі Інтернет. Ефективність у цьому режимі визначається балансом між розміром файлу та збереженням візуальної якості. Аналіз залежності ступеня стиснення (біт на піксель) від об'єктивної візуальної якості (оцінка SSIMULACRA2) виявляє ключові відмінності між кодеками.

Для фотографій (рис. 4.6), при низьких та середніх рівнях якості (оцінка < 65), формат AVIF демонструє найкращу ефективність. Однак при переході до високих та дуже високих рівнів якості (оцінка > 70), його випереджає JPEG XL, забезпечуючи стабільно менший розмір файлу за тієї ж візуальної схожості з оригіналом. WebP стабільно поступається обом сучасним форматам, а оригінальний JPEG, очікувано, займає останнє місце.

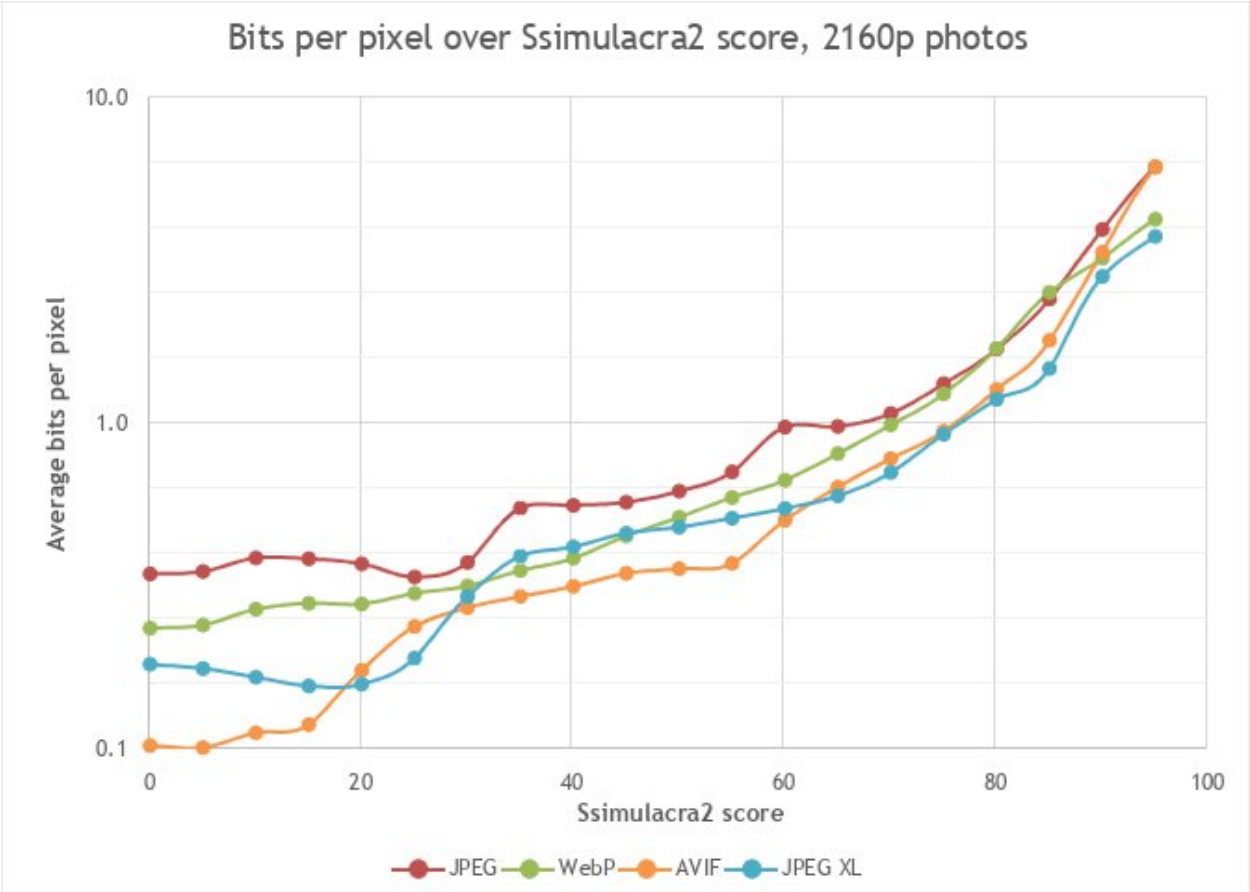

Рис. 4.6 Ефективність стиснення відносно оцінки якості фото

Для скріншотів (рис. 4.7) домінування AVIF є ще більш вираженим у діапазоні від низької до високої якості. JPEG XL зрівнюється з ним та перевершує лише за дуже високих значень якості (оцінка > 85). Це пояснюється походженням кодека AV1 з відео, що робить його ефективним для збереження чітких меж та контурів, характерних для графіки. WebP та JPEG поступаються обом форматам.

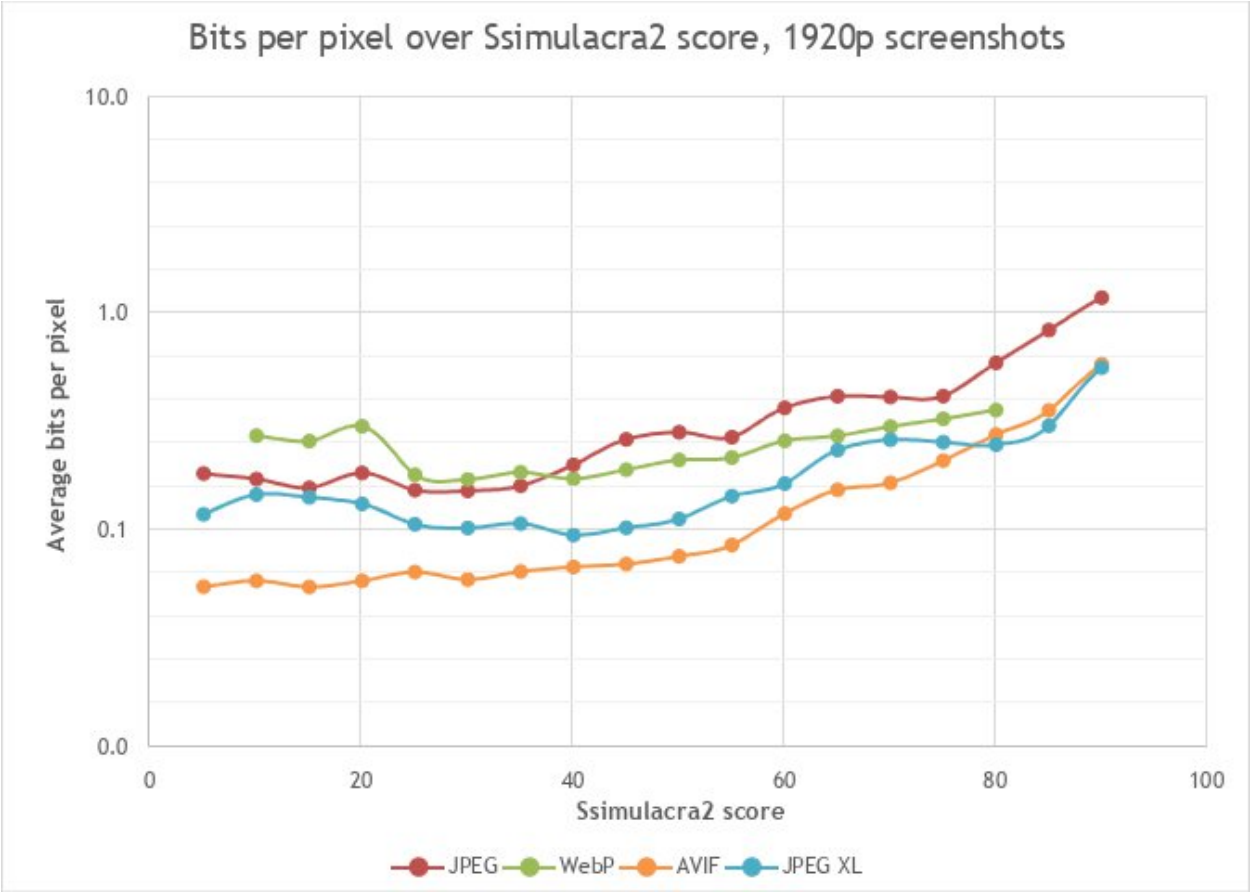

Рис. 4.7 Ефективність стиснення відносно оцінки якості скріншотів

Кореляція переваги ефективності стиснення з втратами також була визначена з розміром зображення. Як показано на графіку (рис. 4.8), для зображень малої роздільної здатності (ширина до ~600 пікселів, загальна кількість ~0.4МП) найкращий ступінь стиснення демонструє AVIF, тоді як для зображень більшого розміру, перевагу займає JPEG XL. Це можна пояснити тим, що зменшені зображення мають вищу щільність інформації та деталей, з чим краще справляються алгоритми прогнозування AVIF, тоді як JPEG XL ефективніше кодує великі плавні області, характерні для фотографій високої роздільної здатності, для яких розробка формату й була націлена. Подібної кореляції у порівнянні стиснення без втрат знайдено не було.

Рис. 4.8 Ефективність стиснення відносно розмірів зображення

4.2.4. Узагальнений порівняльний аналіз стиснення з втратами. Для підсумкової оцінки було проведено агрегацію результатів за трьома категоріями візуальної якості: "хороша" (оцінка SSIMULACRA2 70-79), "висока" (80-89) та "дуже висока" (90+).

При стисненні фотографій (рис. 4.9) JPEG XL демонструє найкращі результати у всіх категоріях якості, причому його перевага над AVIF зростає зі збільшенням вимог до якості.

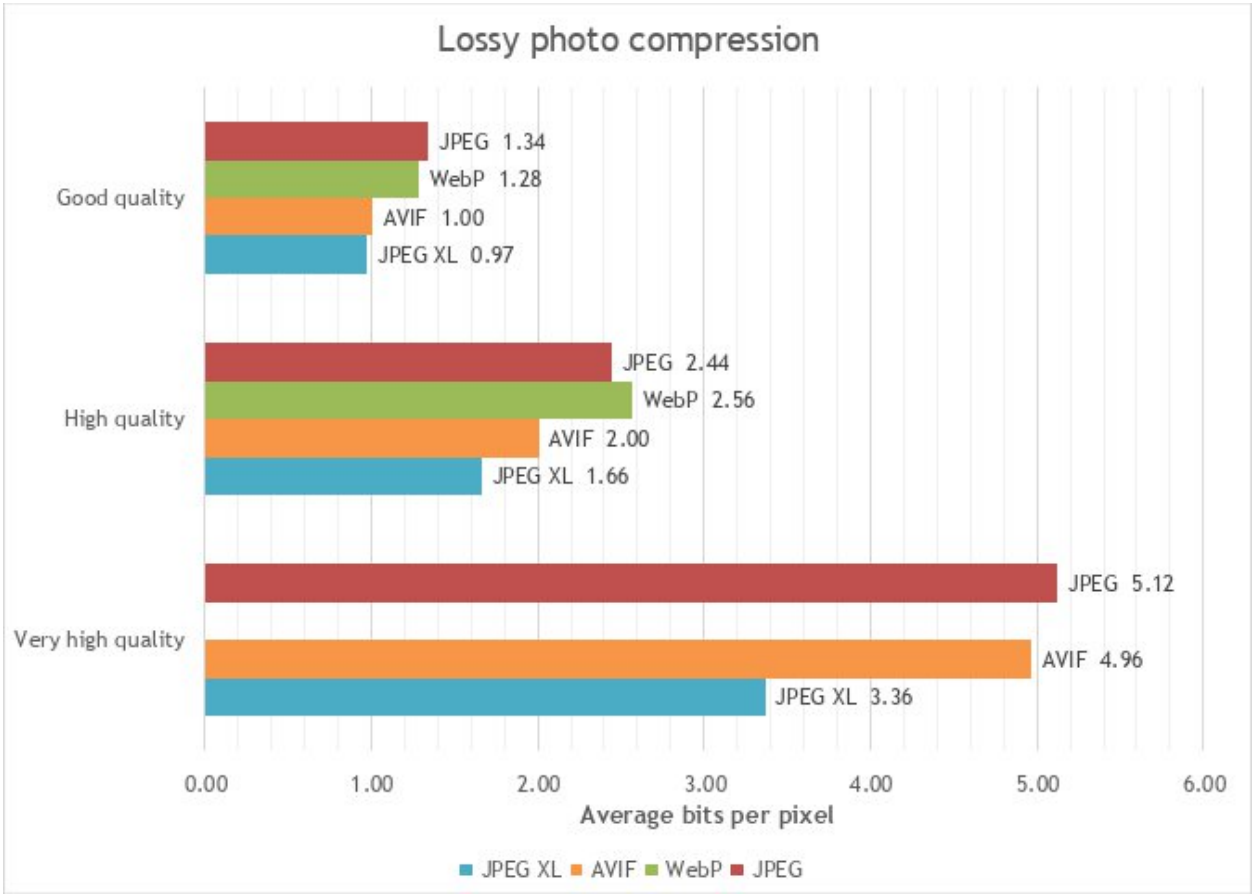

Рис. 4.9 Середня ефективність стиснення фото з втратами

При стисненні скріншотів (рис. 4.10) AVIF є лідером у категорії "хорошої" якості. У категоріях "високої" та "дуже високої" якості JPEG XL показує результати, порівнянні з AVIF, або незначно його перевершує. Формат WebP стабільно посідає третє місце, значно випереджаючи JPEG, але поступаючись AVIF та JPEG XL.

Рис. 4.10 Середня ефективність стиснення скріншотів з втратами

Значення WebP не було включено до категорії дуже високої якості, адже формат рідко досягає значень якості вище 90. Це зумовлено строгим використанням колірної субдискретизації (YCrCb 4:2:0), тобто зниженої роздільної здатності колірних каналів, що унеможливлює точне відтворення різких контурів та змін кольору, призводячи до спотворенню кольорів та артефактів розмиття на краях об’єктів (color bleed).

4.3 Практичні рекомендації щодо вибору формату

Результати, отримані в ході експериментального дослідження, дозволяють сформулювати обґрунтовані рекомендації щодо вибору оптимального формату растрових зображень залежно від специфіки застосування зображень, пріоритетів щодо якості, розміру файлу та обчислювальних витрат.

1. Архівування та зберігання майстер-копій. Для завдань, що вимагають абсолютної цілісності даних, найефективнішим є формат JPEG XL у режимі стиснення без втрат. Він забезпечує найвищий коефіцієнт стиснення серед усіх розглянутих форматів, суттєво перевершуючи WebP та класичний PNG, що дозволяє оптимізувати використання дискового простору без жодної втрати якості;
2. Веб-графіка та синтетичні зображення (логотипи, іконки, скріншоти). Для цього типу контенту, що характеризується чіткими межами та обмеженою палітрою, вибір залежить від вимог до якості:
    - AVIF (з втратами, середня якість): Є оптимальним вибором, коли пріоритетом є мінімальний розмір файлу. Його алгоритми, успадковані з відеокодеків, ефективно кодують чіткі контури та однорідні області;
    - JPEG XL (без втрат або з непомітними втратами): Рекомендовано для випадків, що вимагають ідеальної точності передачі графіки, оскільки він забезпечує краще стиснення, ніж PNG, при повній відсутності артефактів.
3. Фотографії для веб-ресурсів високої якості. При публікації фотографій, де критично важливо зберегти високу візуальну якість (наприклад, у портфоліо, на сторінках продуктів), перевагу слід віддавати формату JPEG XL (з втратами). Експерименти показали його суттєву перевагу в ефективності стиснення за високих значень візуальної якості, що дозволяє досягти меншого розміру файлу порівняно з AVIF та WebP без помітної деградації зображення, а також завантажувати зображення прогресивно;
4. Універсальний формат для Web з широкою сумісністю. Якщо ключовою вимогою є баланс між ефективністю та підтримкою переважною більшістю браузерів, включаючи дещо застарілі версії, WebP залишається надійним вибором. Він пропонує значно краще стиснення, ніж JPEG та PNG, і на сьогодні є універсально підтримуваним стандартом;

5. Забезпечення максимальної сумісності зі застарілими системами. У випадках, коли необхідно гарантувати відображення контенту на будь-яких пристроях та у програмному забезпеченні, що не оновлювалося, формати JPEG (для фотографій) та PNG (для графіки з прозорістю) залишаються де-факто стандартами.

Отже, вибір оптимального формату є задачею, що вимагає аналізу конкретного сценарію використання та типу графічного контенту. Розроблений в рамках даної роботи програмний засіб слугує практичним інструментом, що дозволяє проводити подібний аналіз для кожного конкретного випадку, валідуючи та уточнюючи наведені загальні рекомендації.

ВИСНОВКИ

У ході виконання магістерської кваліфікаційної роботи було успішно досягнуто поставлену мету, яка полягала у комплексному дослідженні ефективності сучасних форматів растрових зображень та створенні програмного інструменту для їх об'єктивного аналізу.

1. Було виконано дослідження ключових технічних особливостей форматів файлів растрових зображень. Проведено системний аналіз алгоритмічних та технологічних основ класичних і новітніх форматів зображень. Це дозволило систематизувати їхні функціональні можливості, переваги та обмеження, що стало ефектною теоретичною базою для подальших досліджень. На основі цього аналізу розроблено концептуальну архітектуру програмного засобу з багаторівневою моделлю, що забезпечила модульність та гнучкість системи, чітко розмежувавши логіку інтерфейсу, обчислювальне ядро та механізми взаємодії із зовнішніми інструментами.
2. Практична частина роботи полягала в реалізації спроєктованої системи з використанням сучасного технологічного стека на платформі .NET. Створено нативне комп’ютерне програмне забезпечення з інтуїтивним графічним інтерфейсом, інкапсуляцію процесів кодування, реалізацією асинхронних операцій для підтримки інтерактивності інтерфейсу та алгоритми оптимізації для автоматичного підбору параметрів кодеків за цільовими критеріями.
3. Ключовим етапом роботи стало проведення кодування набору зображень використовуючи актуальні растрові формати задля отримання продуктивних та якісних метрик автоматизованого експериментального дослідження з використанням розробленого програмного коду. На репрезентативних наборах даних, що включали фотографічний та синтетичний контент, було виконано тисячі операцій кодування, в ході

яких систематично збирались та агрегувались ключові метрики продуктивності.

4. Отримані емпіричні дані були ретельно проаналізовані та візуалізовані, що дозволило виявити складні закономірності в ефективності форматів за різних умов. Результати аналізу стали основою для формулювання обґрунтованих висновків та практичних рекомендацій щодо вибору оптимального формату для конкретного сценарію використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. JPEG – Wikipedia [Електронний ресурс] // en.wikipedia.org – Режим доступу: https://en.wikipedia.org/wiki/JPEG
2. PNG – Wikipedia [Електронний ресурс] // en.wikipedia.org – Режим доступу: https://en.wikipedia.org/wiki/PNG
3. WebP Compression Techniques [Електронний ресурс] // Google Developers – Режим доступу: https://developers.google.com/speed/webp/docs/compression
4. AV1 Image File Format (AVIF) – Alliance for Open Media [Електронний ресурс] // developers.google.com – Режим доступу: https://aomediacodec.github.io/av1-avif/
5. JPEG XL White Paper: JPEG XL Image Coding System [Електронний ресурс] // ds.jpeg.org – Режим доступу: https://ds.jpeg.org/whitepapers/jpeg-xl-whitepaper.pdf
6. Ringing artifacts – Wikipedia [Електронний ресурс] // en.wikipedia.org – Режим доступу: https://en.wikipedia.org/wiki/Ringing_artifacts
7. AV1: The Constrained Directional Enhancement Filter [Електронний ресурс] // hacks.mozilla.org – Режим доступу: https://hacks.mozilla.org/2018/06/av1-next-generation-video-the-constrained-directional-enhancement-filter/
8. Parallel Implementation of Sample Adaptive Offset Filtering Block for Low-Power HEVC Chip – Luis A. Fernández Lara [Електронний ресурс] // dspace.mit.edu – Режим доступу: https://dspace.mit.edu/bitstream/handle/1721.1/100609/932233007-MIT.pdf
9. Microsoft. .NET Documentation [Електронний ресурс] // docs.microsoft.com – Режим доступу: https://docs.microsoft.com/en-us/dotnet/
10. Microsoft. Windows UI Library (WinUI) 3 [Електронний ресурс] // docs.microsoft.com – Режим доступу: https://docs.microsoft.com/en-us/windows/apps/winui/winui3/

11. Microsoft. Process Class (System.Diagnostics) [Електронний ресурс] // docs.microsoft.com – Режим доступу: https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.process
12. ImageMagick. Command-line Options. [Електронний ресурс] // ImageMagick.org. – Режим доступу: https://imagemagick.org/script/command-line-options.php.
13. Google Inc. libwebp: WebP encoder and decoder library [Електронний ресурс] // Режим доступу: https://chromium.googlesource.com/webm/libwebp/
14. AOMedia. libavif: Library for encoding and decoding .avif files [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/AOMediaCodec/libavif
15. JPEG XL Project. libjxl: JPEG XL image format reference implementation [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/libjxl/libjxl
16. Harvey P. ExifTool: Perl library and command-line application for reading, writing and editing meta information [Електронний ресурс] // exiftool.org – Режим доступу: https://exiftool.org/
17. Kornel.Lesiński. How to compare images fairly. [Електронний ресурс] // kornel.ski – Режим доступу: https://kornel.ski/en/faircomparison.
18. Sneyers J. Ssimulacra 2. [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/cloudinary/ssimulacra2.
19. Free archive of digital sampling test images [Електронний ресурс] // testimages.org – Режим доступу: https://testimages.org/sampling/
20. DIV2K dataset [Електронний ресурс] // data.vision.ee.ethz.ch – Режим доступу: https://data.vision.ee.ethz.ch/cvl/DIV2K/
21. Flickr2K dataset – Kaggle [Електронний ресурс] // kaggle.com – Режим доступу: https://www.kaggle.com/datasets/daehoyang/flickr2k

Додаток А

Програмний код файлу Backend.cs

```
using Microsoft.UI.Xaml.Controls;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using Windows.Storage;

namespace AppImageComp
{
  static class Backend
  {
    private static string exiftoolPath = @"utils\exiftool-13.36_64\exiftool.exe";
    private static string imagemagickPath = @"utils\ImageMagick-7.1.2-3-portable-Q8-x64\magick.exe";
    private static string cwebpPath = @"utils\libwebp-1.6.0-windows-x64\cwebp.exe";
    private static string avifencPath = @"utils\libavif-1.3.0\avifenc.exe";
    private static string cjxlPath = @"utils\jxl-x64-windows-static\cjxl.exe";
    private static string ssimPath = @"utils\jxl-x64-windows-static\ssimulacra2.exe";

    public static async Task<string> ExifTool(string args) {
      var processInfo = new ProcessStartInfo {
        FileName = exiftoolPath,
        Arguments = args,
        RedirectStandardOutput = true,
        UseShellExecute = false,
        CreateNoWindow = true
      };
      string output;

      using (var process = new Process()) {
        process.StartInfo = processInfo;
        process.Start();
        output = (await process.StandardOutput.ReadToEndAsync()).TrimEnd();
        await process.WaitForExitAsync();
      }

      return output;
    }

    public static async Task Magick(string args) {
      var processInfo = new ProcessStartInfo {
        FileName = imagemagickPath,
        Arguments = args,
        UseShellExecute = false,
        CreateNoWindow = true
```

```
      };

      using (var process = new Process()) {
        process.StartInfo = processInfo;
        process.Start();
        await process.WaitForExitAsync();
      }
    }

    public static async Task<string> PNG(string file, int effort = 75) {
      effort = Math.Clamp(effort, 1, 100);
      string temp = Path.Combine(Path.GetTempPath(), Path.ChangeExtension(Path.GetRandomFileName(),
"png"));
      await Magick($"-quiet -quality {effort} \"{file}\" -strip \"{temp}\"");
      return temp;
    }

    public static async Task<string> JPEG(string file, int quality = 75, int chroma = -1) {
      quality = Math.Clamp(quality, 1, 100);
      if (chroma == -1) {
        chroma = quality switch { ≥85 ⇒ 0, ≥50 ⇒ 1, _ ⇒ 2 };
      }
      chroma = Math.Clamp(chroma, 0, 2);

      string temp = Path.Combine(Path.GetTempPath(), Path.ChangeExtension(Path.GetRandomFileName(),
"jpg"));
      await Magick($"-quiet -quality {quality} -sampling-factor {chroma switch { 1 ⇒ "4:2:2", 2 ⇒
"4:2:0", _ ⇒ "4:4:4" }} \"{file}\" \"{temp}\"");
      return temp;
    }

    public static async Task<string> WEBP(string file, int quality = 75, int effort = 4, bool lossless
= false) {
      quality = Math.Clamp(quality, 0, 100);
      effort = Math.Clamp(effort, 0, 6);

      string temp = Path.Combine(Path.GetTempPath(), Path.ChangeExtension(Path.GetRandomFileName(),
"webp"));
      var processInfo = new ProcessStartInfo {
        FileName = cwebpPath,
        Arguments = $"-quiet -m {effort} {(lossless ? "-lossless" : $"-q {quality}")} \"{file}\" -o
\"{temp}\"",
        UseShellExecute = false,
        CreateNoWindow = true
      };

      using (var process = new Process()) {
        process.StartInfo = processInfo;
        process.Start();
        await process.WaitForExitAsync();
      }
      return temp;
```

```
    }

    public static async Task<string> AVIF(string file, int quality = 60, int speed = 6, bool lossless
= false) {
      quality = Math.Clamp(quality, 0, 100);
      speed = Math.Clamp(speed, 0, 10);

      string temp = Path.Combine(Path.GetTempPath(), Path.ChangeExtension(Path.GetRandomFileName(),
"avif"));
      var processInfo = new ProcessStartInfo {
        FileName = avifencPath,
        Arguments = $"-s {speed} {(lossless ? "-l" : $"-q {quality}")} \"{file}\" -o \"{temp}\"",
                UseShellExecute = false,
        CreateNoWindow = true
      };
      string output;

      using (var process = new Process()) {
        process.StartInfo = processInfo;
        process.Start();
        await process.WaitForExitAsync();
      }
      return temp;
    }

    public static async Task<string> JPEGXL(string file, int quality = 60, int effort = 7, bool
lossless = false) {
      quality = Math.Clamp(quality, 0, 99);
      effort = Math.Clamp(effort, 1, 10);

      string temp = Path.Combine(Path.GetTempPath(), Path.ChangeExtension(Path.GetRandomFileName(),
"jxl"));
      var processInfo = new ProcessStartInfo {
        FileName = cjxlPath,
        Arguments = $"-e {effort} {(lossless ? (file.EndsWith("jpg") ? "-j 1" : "-d 0") : $"-j 0 -q
{quality}")} \"{file}\" \"{temp}\"",
        UseShellExecute = false,
        CreateNoWindow = true
      };

      using (var process = new Process()) {
        process.StartInfo = processInfo;
        process.Start();
        await process.WaitForExitAsync();
      }
      return temp;
    }

    public static async Task<double> SSIM(string orig, string distort) {
      if (distort.EndsWith("webp") || distort.EndsWith("avif") || distort.EndsWith("jxl")) {
        distort = await PNG(distort, 5);
      }
```

```
  double score = 0;
  var processInfo = new ProcessStartInfo {
    FileName = ssimPath,
    Arguments = $"\"{orig}\" \"{distort}\"",
    RedirectStandardOutput = true,
    RedirectStandardError = true,
    UseShellExecute = false,
    CreateNoWindow = true
  };
  string output;

  using (var process = new Process()) {
    process.StartInfo = processInfo;
    process.Start();
    output = await process.StandardOutput.ReadToEndAsync();
    await process.WaitForExitAsync();
  }
  try {
    score = Convert.ToDouble(output);
  }
  catch(FormatException) {
    return 404;
  }

  return score;
}

public static async Task<Dictionary<string, string>> GetExifInfo(string file) {
  string output = await ExifTool($"-t -FileSize# -all:all \"{file}\"");
  Dictionary<string, string> exif = new();

  using (StringReader reader = new StringReader(output)) {
    string? line = reader.ReadLine();
    exif.Add(line.Split('\t')[0], line.Split('\t')[1]);
    while ((line = reader.ReadLine()) ≠ null) {
      if (!exif.ContainsKey(line.Split('\t')[0]))
        exif.Add(line.Split('\t')[0], line.Split('\t')[1]);
    }
  }

  return exif;
}

public static string sizeMakeReadable(long bytes) {
  double size = bytes;
  int p10 = 0;
  while (size ≥ 1024) {
    size /= 1024f;
    p10++;
  }
  return $"{size:F2} {p10 switch { 1 ⇒ "KB", 2 ⇒ "MB", 3 ⇒ "GB", 4 ⇒ "TB", _ ⇒ "B" }}";
```

```
    }

    private static Task CROP(IProgress<int> progress) {
      string[] webscreens = Directory.GetFiles(@"C:\Users\k\Desktop\DATASET\WebScreenshots");
      return Task.Run(() => {
        int count = 0;
        for (int i = 0; count < 1000; i++) {
          int w = (1 + Random.Shared.Next(80)) * 20;
          int h = (1 + Random.Shared.Next(45)) * 20;
          while (w / h > 8) h += 20;
          while (h / w > 4) w += 20;
          int x = (1 + Random.Shared.Next(96 - w / 20)) * 20;
          int y = (1 + Random.Shared.Next(54 - h / 20)) * 20;
          Magick($"{webscreens[2000 + i]} -crop {w}x{h}+{x}+{y} +repage
C:\\Users\\k\\Desktop\\DATASET\\WebScreenshots-CROP\\{count + 1:D4}.png");
          if (new System.IO.FileInfo($"C:\\Users\\k\\Desktop\\DATASET\\WebScreenshots-CROP\\{count +
1:D4}.png").Length > 2048)
            count++;
          progress.Report(count / 10 + 1);
        }
      });
    }

    public static async Task RunTestsGeneral(IProgress<int> progress) {
      int[] sweep = { 10, 30, 50, 60, 70, 80, 85, 90, 95, 98};
      string[] testimages = Directory.GetFiles(@"C:\Users\k\Desktop\DATASET\TESTIMAGES");
      string[] div2k = Directory.GetFiles(@"C:\Users\k\Desktop\DATASET\DIV2K");
      string[] flickr = Directory.GetFiles(@"C:\Users\k\Desktop\DATASET\Flickr2K");
      string[] webscreens = Directory.GetFiles(@"C:\Users\k\Desktop\DATASET\WebScreenshots");
      string[] webscreensCROP = Directory.GetFiles(@"C:\Users\k\Desktop\DATASET\WebScreenshots-CROP");

      List<string> allFiles = new List<string>();
      for (int i = 0; i < webscreensCROP.Length; i++) allFiles.Add(webscreensCROP[i]);

      string temp = "";
      Stopwatch sw = new Stopwatch();

      await Task.Run(async () => {
        using (StreamWriter csv = new StreamWriter(@$"C:\Users\k\Desktop\STATS\stats-
{DateTime.Now.Month:D2}-{DateTime.Now.Day:D2}-{DateTime.Now.Hour:D2}{DateTime.Now.Minute:D2}.csv")) {

csv.WriteLine("initform,initsize,width,height,imgtype,format,quality,chroma,effort,lossless,time,size,
ssim");

          for (int i = 0; i < allFiles.Count; i++) {
            progress.Report(i * 100 / allFiles.Count);
            csv.Flush();
            StorageFile file = await StorageFile.GetFileFromPathAsync(allFiles[i]);
            Dictionary<string, string> exif = await GetExifInfo(file.Path);

            Debug.WriteLine($"Image ({i + 1} / {allFiles.Count}): {file.Name}");
            Debug.WriteLine("Encoding PNGs ... ");
```

```
            sw.Restart();
            temp = await PNG(file.Path);
            sw.Stop();
            csv.WriteLine($"{exif["MIME Type"].Split("/")[1]},{exif["File Size"]},{exif["Image
Width"]},{exif["Image Height"]},photo,png,,,7,yes,{sw.ElapsedMilliseconds},{new
System.IO.FileInfo(temp).Length},100");
            File.Delete(temp);

            Debug.WriteLine("Encoding JPEGs …");
            foreach (int q in sweep) {
              sw.Restart();
              temp = await JPEG(file.Path, q);
              sw.Stop();
              csv.WriteLine($"{exif["MIME Type"].Split("/")[1]},{exif["File Size"]},{exif["Image
Width"]},{exif["Image Height"]},photo,jpeg,{q},{q switch { ≥ 85 ⇒ "444", ≥ 50 ⇒ "422", _ ⇒
"420" }},,no,{sw.ElapsedMilliseconds},{new System.IO.FileInfo(temp).Length},{SSIM(file.Path,
temp):F2}");
              File.Delete(temp);
            }

            Debug.WriteLine("Encoding WebPs …");
            foreach (int q in sweep) {
              sw.Restart();
              temp = await WEBP(file.Path, q);
              sw.Stop();
              csv.WriteLine($"{exif["MIME Type"].Split("/")[1]},{exif["File Size"]},{exif["Image
Width"]},{exif["Image Height"]},photo,webp,{q},420,4,no,{sw.ElapsedMilliseconds},{new
System.IO.FileInfo(temp).Length},{SSIM(file.Path, temp):F2}");
              File.Delete(temp);
            }
            sw.Restart();
            temp = await WEBP(file.Path, 100, 4, true);
            sw.Stop();
            csv.WriteLine($"{exif["MIME Type"].Split("/")[1]},{exif["File Size"]},{exif["Image
Width"]},{exif["Image Height"]},photo,webp,,,4,yes,{sw.ElapsedMilliseconds},{new
System.IO.FileInfo(temp).Length},100");
            File.Delete(temp);

            Debug.WriteLine("Encoding AVIFs …");
            foreach (int q in sweep) {
              sw.Restart();
              temp = await AVIF(file.Path, q);
              sw.Stop();
              csv.WriteLine($"{exif["MIME Type"].Split("/")[1]},{exif["File Size"]},{exif["Image
Width"]},{exif["Image Height"]},photo,avif,{q},444,4,no,{sw.ElapsedMilliseconds},{new
System.IO.FileInfo(temp).Length},{SSIM(file.Path, temp):F2}");
              File.Delete(temp);
            }
            sw.Restart();
            temp = await AVIF(file.Path, 100, 6, true);
            sw.Stop();
```

```
            csv.WriteLine($"{exif["MIME Type"].Split("/")[1]},{exif["File Size"]},{exif["Image
Width"]},{exif["Image Height"]},photo,avif,,,4,yes,{sw.ElapsedMilliseconds},{new
System.IO.FileInfo(temp).Length},100");
            File.Delete(temp);

            Debug.WriteLine("Encoding JXLs ... ");
            foreach (int q in sweep) {
              sw.Restart();
              temp = await JPEGXL(file.Path, q);
              sw.Stop();
              csv.WriteLine($"{exif["MIME Type"].Split("/")[1]},{exif["File Size"]},{exif["Image
Width"]},{exif["Image Height"]},photo,jpegxl,{q},,7,no,{sw.ElapsedMilliseconds},{new
System.IO.FileInfo(temp).Length},{SSIM(file.Path, temp):F2}");
              File.Delete(temp);
            }
            sw.Restart();
            temp = await JPEGXL(file.Path, 100, 7, true);
            sw.Stop();
            csv.WriteLine($"{exif["MIME Type"].Split("/")[1]},{exif["File Size"]},{exif["Image
Width"]},{exif["Image Height"]},photo,jpegxl,,,7,yes,{sw.ElapsedMilliseconds},{new
System.IO.FileInfo(temp).Length},100");
            File.Delete(temp);
          }
        }
      });
    }

        public static async Task RunTestsLossless(IProgress<int> progress) {
            string[] testimages = Directory.GetFiles(@"C:\Users\k\Desktop\DATASET\TESTIMAGES");
            string[] div2k = Directory.GetFiles(@"C:\Users\k\Desktop\DATASET\DIV2K");
            string[] flickr = Directory.GetFiles(@"C:\Users\k\Desktop\DATASET\Flickr2K");
            string[] webscreens = Directory.GetFiles(@"C:\Users\k\Desktop\DATASET\WebScreenshots");
            string[] webscreensCROP = Directory.GetFiles(@"C:\Users\k\Desktop\DATASET\WebScreenshots-
CROP");

            List<string> allFiles = new List<string>();
            for (int i = 1000; i < flickr.Length; i++) allFiles.Add(flickr[i]);

            string temp = "";
            Stopwatch sw = new Stopwatch();

            await Task.Run(async () => {
                using (StreamWriter csv = new StreamWriter(@$"C:\Users\k\Desktop\STATS\stats-
{DateTime.Now.Month:D2}-{DateTime.Now.Day:D2}-{DateTime.Now.Hour:D2}{DateTime.Now.Minute:D2}.csv")) {

csv.WriteLine("initform,initsize,width,height,imgtype,format,quality,chroma,effort,lossless,time,size,
ssim");

                    for (int i = 0; i < allFiles.Count; i++) {
                        progress.Report(i * 100 / allFiles.Count);
                        csv.Flush();
                        StorageFile file = await StorageFile.GetFileFromPathAsync(allFiles[i]);
```

```
                        Dictionary<string, string> exif = await GetExifInfo(file.Path);

                        Debug.WriteLine($"Image ({i + 1} / {allFiles.Count}): {file.Name}");
                        Debug.WriteLine("Encoding PNGs ... ");
            for (int e = 1; e ≤ 9; e++) {
              sw.Restart();
              temp = await PNG(file.Path, e * 10 + 5);
              sw.Stop();
              csv.WriteLine($"{exif["MIME Type"].Split("/")[1]},{exif["File Size"]},{exif["Image
Width"]},{exif["Image Height"]},photo,png,,,{e},yes,{sw.ElapsedMilliseconds},{new
System.IO.FileInfo(temp).Length},100");
              File.Delete(temp);
            }

            Debug.WriteLine("Encoding WebPs ... ");
            for (int e = 0; e ≤ 6; e++) {
              sw.Restart();
              temp = await WEBP(file.Path, 100, e, true);
              sw.Stop();
              csv.WriteLine($"{exif["MIME Type"].Split("/")[1]},{exif["File Size"]},{exif["Image
Width"]},{exif["Image Height"]},photo,webp,,,{e},yes,{sw.ElapsedMilliseconds},{new
System.IO.FileInfo(temp).Length},100");
              File.Delete(temp);
            }

                        Debug.WriteLine("Encoding AVIFs ... ");
            for (int e = 0; e ≤ 10; e++) {
              sw.Restart();
              temp = await AVIF(file.Path, 100, e, true);
              sw.Stop();
              csv.WriteLine($"{exif["MIME Type"].Split("/")[1]},{exif["File Size"]},{exif["Image
Width"]},{exif["Image Height"]},photo,avif,,,{10 − e},yes,{sw.ElapsedMilliseconds},{new
System.IO.FileInfo(temp).Length},100");
              File.Delete(temp);
            }

                        Debug.WriteLine("Encoding JXLs ... ");
            for (int e = 0; e ≤ 9; e++) {
              sw.Restart();
              temp = await JPEGXL(file.Path, 100, e, true);
              sw.Stop();
              csv.WriteLine($"{exif["MIME Type"].Split("/")[1]},{exif["File Size"]},{exif["Image
Width"]},{exif["Image Height"]},photo,jpegxl,,,{e},yes,{sw.ElapsedMilliseconds},{new
System.IO.FileInfo(temp).Length},100");
              File.Delete(temp);
            }
                    }
                }
            });
        }
    }
}
```

Додаток Б

Програмний код файлу MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.UI;
using Microsoft.UI.Input;
using Microsoft.UI.Text;
using Microsoft.UI.Xaml;
using Microsoft.UI.Xaml.Controls;
using Microsoft.UI.Xaml.Data;
using Microsoft.UI.Xaml.Input;
using Microsoft.UI.Xaml.Media;
using Microsoft.UI.Xaml.Media.Imaging;
using Microsoft.UI.Xaml.Navigation;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.Graphics.Imaging;
using Windows.Storage;
using Windows.Storage.Pickers;
using Windows.Storage.Streams;
using Windows.UI.Popups;
using Windows.UI.Text;
using WinRT.Interop;

namespace AppImageComp
{
  public sealed partial class MainWindow : Window {
    private enum formatOptions { JPEG, WEBP, AVIF, JPEGXL, PNG, LWEBP, LAVIF, LJPEGXL };
    private string[] formatLabels = { "JPEG", "WebP", "AVIF", "JPEG XL", "PNG", "WebP Lossless", "AVIF
Lossless", "JPEG XL Lossless" };
    private enum convertMode { MANUAL, SIZE, QUALITY };
    private struct convertOptions {
      public convertMode mode = convertMode.MANUAL;
      public byte quality = 90;
      public byte effort = 1;
      public long targetSize = 1024 * 1024;
      public double targetQuality = 90;

      public convertOptions() {}
    };
    private byte[,] effortMap = { { 0, 2, 8, 4, 25, 2, 8, 4 }, { 0, 4, 6, 6, 55, 4, 6, 6 }, {0, 5, 3,
8, 75, 5, 3, 8 }, {0, 6, 1, 9, 95, 6, 1, 9 } };

    private string? loadedImagePath;
    private string? loadedImagePathPNG;
```

```
    private long? loadedImageSize;
    Dictionary<string, string> convertedImagePaths = new();
    Dictionary<string, BitmapImage> fullImages = new Dictionary<string, BitmapImage>();
    Dictionary<string, BitmapImage> previewImages = new Dictionary<string, BitmapImage>();

    private bool[] selectedFormats = { true, true, true, true, true, true, true, true };
    private convertOptions conOptions = new();

    public MainWindow() {
      this.InitializeComponent();

      SelectorBarLoss.SelectedItem = SelectorBarLoss.Items[0];
      GridSettingTargetQuality.Visibility = Visibility.Collapsed;
      GridSettingTargetSize.Visibility = Visibility.Collapsed;
    }

        private void MainWindow_Closed(object sender, WindowEventArgs args) {
      foreach (string f in convertedImagePaths.Values) File.Delete(f);
        }

    private async void OpenFullPreview(object sender, RoutedEventArgs e) {
      if (!fullImages.ContainsKey((sender as Image).Tag as string)) return;

      ImagePopup.Source = fullImages[(sender as Image).Tag as string];
      BorderPopup.Visibility = Visibility.Visible;
      BorderPopup.Opacity = 1;
      ScrollViewPopup.ZoomTo(1f, new Vector2((float)App.MainAppWindow.Bounds.Width / 2,
(float)App.MainAppWindow.Bounds.Height / 2), new
ScrollingZoomOptions(ScrollingAnimationMode.Disabled));

      foreach (UIElement b in StackPanelFormatButtons.Children) {
        if (b is Button) {
          if ((string)(b as Button).Tag == (string)(sender as Image).Tag)
            (b as Button).Style = (Style)Application.Current.Resources["AccentButtonStyle"];
          else
            (b as Button).Style = (Style)Application.Current.Resources["DefaultButtonStyle"];
        }
      }
    }

    private async void PopupClose(object sender, RoutedEventArgs e) {
      BorderPopup.Opacity = 0;
      await Task.Delay(200);
      BorderPopup.Visibility = Visibility.Collapsed;
    }

    private async Task DisplayExifInfo(string file) {
      Dictionary<string, string> exif = await Backend.GetExifInfo(file);

      GridFileInfo.Children.Clear();
      Border[] borders = new Border[10];
      for (int i = 0; i < borders.Length; i++) {
```

```
        borders[i] = new Border { BorderBrush =
(Brush)Application.Current.Resources["ControlFillColorSecondaryBrush"], BorderThickness = (i <
borders.Length - 2 ? new Thickness(0, 0, 0, 1) : new Thickness(0)) };
        GridFileInfo.Children.Add(borders[i]);
        Grid.SetRow(borders[i], i / 2);
        Grid.SetColumn(borders[i], i % 2);
      }

      TextBlock lbl1 = new TextBlock { Text = "Name", VerticalAlignment = VerticalAlignment.Center,
Foreground = (Brush)Application.Current.Resources["TextFillColorTertiaryBrush"], Margin = new
Thickness(16, 0, 0, 0) };
      TextBlock lbl2 = new TextBlock { Text = "File size", VerticalAlignment =
VerticalAlignment.Center, Foreground =
(Brush)Application.Current.Resources["TextFillColorTertiaryBrush"], Margin = new Thickness(16, 0, 0,
0) };
      TextBlock lbl3 = new TextBlock { Text = "Resolution", VerticalAlignment =
VerticalAlignment.Center, Foreground =
(Brush)Application.Current.Resources["TextFillColorTertiaryBrush"], Margin = new Thickness(16, 0, 0,
0) };
      TextBlock lbl4 = new TextBlock { Text = "Image format", VerticalAlignment =
VerticalAlignment.Center, Foreground =
(Brush)Application.Current.Resources["TextFillColorTertiaryBrush"], Margin = new Thickness(16, 0, 0,
0) };
      TextBlock lbl5 = new TextBlock { Text = "", VerticalAlignment = VerticalAlignment.Center,
Foreground = (Brush)Application.Current.Resources["TextFillColorTertiaryBrush"], Margin = new
Thickness(16, 0, 0, 0) };
      borders[0].Child = lbl1; borders[2].Child = lbl2; borders[4].Child = lbl3; borders[6].Child =
lbl4; borders[8].Child = lbl5;

      TextBlock name = new TextBlock { Text = Path.GetFileNameWithoutExtension(file),
VerticalAlignment = VerticalAlignment.Center, Margin = new Thickness(16, 0, 0, 0) };
      TextBlock size = new TextBlock { Text = Backend.sizeMakeReadable(Convert.ToInt64(exif["File
Size"])), VerticalAlignment = VerticalAlignment.Center, Margin = new Thickness(16, 0, 0, 0) };
      TextBlock res = new TextBlock { Text = $"{exif["Image Size"].Replace("x", " × ")}
({exif["Megapixels"]} MP)", VerticalAlignment = VerticalAlignment.Center, Margin = new Thickness(16,
0, 0, 0) };
      TextBlock format = new TextBlock { VerticalAlignment = VerticalAlignment.Center, Margin = new
Thickness(16, 0, 0, 0) };
      TextBlock extra = new TextBlock { VerticalAlignment = VerticalAlignment.Center, Margin = new
Thickness(16, 0, 0, 0) };
      borders[1].Child = name; borders[3].Child = size; borders[5].Child = res; borders[7].Child =
format; borders[9].Child = extra;

      if (exif["MIME Type"] == "image/png") {
        format.Text = "PNG";
        lbl5.Text = "Color";
        if (exif["Color Type"] == "RGB with Alpha") extra.Text = "RGBA 32-bit";
        else if (exif["Color Type"] == "RGB") extra.Text = "RGB 24-bit";
        else if (exif["Color Type"] == "Palette") extra.Text = $"Indexed {exif["Bit Depth"]}-bit";
      }
      if (exif["MIME Type"] == "image/jpeg") {
        format.Text = "JPEG";
```

```
        lbl5.Text = "Quality setting";
        string q = (await Backend.ExifTool($"-t -JpegQualityEstimate \"{file}\"")).Split('\t')[1];
        string sub = exif["Y Cb Cr Sub Sampling"].Split(' ')[0];
        extra.Text = $"{q} / 100";
        if (sub == "YCbCr4:4:4") extra.Text += ", full color resolution (YCbCr 4:4:4)";
        if (sub == "YCbCr4:4:0") extra.Text += ", half color resolution (YCbCr 4:4:0)";
        if (sub == "YCbCr4:2:2") extra.Text += ", half color resolution (YCbCr 4:2:2)";
        if (sub == "YCbCr4:2:0") extra.Text += ", quarter color resolution (YCbCr 4:2:0)";
        if (sub == "YCbCr4:0:0") extra.Text += ", no color (YCbCr 4:0:0)";
      }
      if (exif["MIME Type"] == "image/bmp") {
        format.Text = "BMP";
        lbl5.Text = "Compression";
        extra.Text = exif["Compression"];
      }
      if (exif["MIME Type"] == "image/webp") {
        format.Text = "WebP";
        lbl5.Text = "Compression type";
        if (exif["File Type"].Contains("lossless")) extra.Text = "Lossless";
        else extra.Text = $"Lossy (YCbCr 4:2:0)";
      }
    }

    private async Task ExecuteConversion(convertOptions options, List<formatOptions> activeFormats) {
      if (loadedImagePath == null || loadedImagePathPNG == null || activeFormats.Count == 0) return;

      // temp files to be deleted
      long[] allSizes = new long[activeFormats.Count];
      long[] allTimes = new long[activeFormats.Count];
      double[] allScores = new double[activeFormats.Count];
      Grid[] gridFormats = new Grid[activeFormats.Count];
      Border[] borderSizes = new Border[activeFormats.Count];
      Border[] borderTimes = new Border[activeFormats.Count];
      Border[] borderScores = new Border[activeFormats.Count];

      foreach (string f in convertedImagePaths.Values) (await
StorageFile.GetFileFromPathAsync(f)).DeleteAsync();
      convertedImagePaths.Clear();

            // init full view buttons
            StackPanelFormatButtons.Children.Clear();
      Button buttonSource = new Button { Content = "Original", Tag = "Source" };
      buttonSource.Click += SelectFormatImage;
      StackPanelFormatButtons.Children.Add(buttonSource);
      StackPanelFormatButtons.Children.Add(new AppBarSeparator());

      // init grid
      GridCompare.ColumnDefinitions.Clear();
      GridCompare.RowDefinitions.Clear();
      GridCompare.Children.Clear();
```

```
      GridCompare.ColumnDefinitions.Add(new ColumnDefinition { Width = new GridLength(200,
GridUnitType.Pixel) });
      for (int i = 0; i < activeFormats.Count; i++) {
        GridCompare.ColumnDefinitions.Add(new ColumnDefinition { Width = new GridLength(1,
GridUnitType.Star) });
      }
      for (int i = 0; i < gridFormats.Length; i++) {
        gridFormats[i] = new Grid();
        for (int j = 0; j < 6; j++) {
          gridFormats[i].RowDefinitions.Add(new RowDefinition { Height = new GridLength(1,
GridUnitType.Star) });
        }
        GridCompare.Children.Add(gridFormats[i]);
        Grid.SetColumn(gridFormats[i], i + 1);

        gridFormats[i].BorderBrush =
(Brush)Application.Current.Resources["ControlFillColorSecondaryBrush"];
        gridFormats[i].BorderThickness = new Thickness(1, 0, 1, 0);
            }

      // first column for labels
      Grid gridLabels = new Grid { ColumnDefinitions = { new ColumnDefinition { Width = new
GridLength(1, GridUnitType.Auto) }, new ColumnDefinition { Width = new GridLength(1,
GridUnitType.Star) } } };
      GridCompare.Children.Add(gridLabels);
      string[] rowLabels = { "Format", "File Size", "Similarity", "Encoding time", "Preview", "Save to
disk" };
      for (int i = 0; i < rowLabels.Length; i++) {
        TextBlock label = new TextBlock { Text = rowLabels[i], HorizontalAlignment =
HorizontalAlignment.Left, VerticalAlignment = VerticalAlignment.Center, Padding = new Thickness(16),
Style = (Style)Application.Current.Resources["BodyLargeTextBlockStyle"] };
        gridLabels.RowDefinitions.Add(new RowDefinition { Height = new GridLength(1,
GridUnitType.Star) });
        gridLabels.Children.Add(label);
        Grid.SetRow(label, i); Grid.SetColumn(label, 1);
        if (i < 4) {
          Button btn = new Button { Content = new FontIcon { Glyph = "\uE8AB", FontSize = 16 }, Width
= 32, Height = 32, Padding = new Thickness(0), Tag = rowLabels[i], IsEnabled = false };
          btn.Click += SortColumns;
          gridLabels.Children.Add(btn);
          Grid.SetRow(btn, i); Grid.SetColumn(btn, 0);
        }
      }

      // format names
      for (int i = 0; i < activeFormats.Count; i++) {
        TextBlock labelName = new TextBlock { Text = formatLabels[(int)activeFormats[i]].Replace("
Lossless", "\nLossless"), HorizontalAlignment = HorizontalAlignment.Center, VerticalAlignment =
VerticalAlignment.Center, HorizontalTextAlignment = TextAlignment.Center, Style =
(Style)Application.Current.Resources["BodyLargeTextBlockStyle"], Tag = (int)activeFormats[i] };
        gridFormats[i].Children.Add(labelName);
        Grid.SetRow(labelName, 0);
```

```
      }

      // PER FORMAT CONVERSION
      for (int i = 0; i < activeFormats.Count; i++) {
        string currentImagePath = "";
        long currentSize = 0;
        long currentTime = 0;
        double currentScore = 0;
        byte actualQuality = 50;

        if (options.mode == convertMode.MANUAL) {
          actualQuality = options.quality;
        }
        else if (options.mode == convertMode.SIZE) {
          byte low = 0; byte high = 101; byte mid;
          long leastDif = long.MaxValue;
          while (high - low > 1) {
            mid = (byte)((high + low) / 2);
            currentImagePath = activeFormats[i] switch {
              formatOptions.JPEG => await Backend.JPEG(loadedImagePathPNG, mid),
              formatOptions.WEBP => await Backend.WEBP(loadedImagePathPNG, mid,
effortMap[options.effort, (int)formatOptions.WEBP]),
              formatOptions.AVIF => await Backend.AVIF(loadedImagePathPNG, mid,
effortMap[options.effort, (int)formatOptions.AVIF]),
              formatOptions.JPEGXL => await Backend.JPEGXL(loadedImagePathPNG, mid,
effortMap[options.effort, (int)formatOptions.JPEGXL])
            };

            currentSize = new FileInfo(currentImagePath).Length;
            Debug.WriteLine($"{activeFormats[i]} | {low} - {mid} - {high} | {currentSize / 1024}");
            if (currentSize > options.targetSize) high = mid;
            else if (currentSize < options.targetSize) low = mid;
            if (Math.Abs(currentSize - options.targetSize) < leastDif) {
              leastDif = Math.Abs(currentSize - options.targetSize);
              actualQuality = mid;
            }
            File.Delete(currentImagePath);
          }
        }
        else if (options.mode == convertMode.QUALITY) {
          byte low = 0; byte high = 101; byte mid;
          double leastDif = 1000;
          while (high - low > 1) {
            mid = (byte)((high + low) / 2);
            currentImagePath = activeFormats[i] switch {
              formatOptions.JPEG => await Backend.JPEG(loadedImagePathPNG, mid),
              formatOptions.WEBP => await Backend.WEBP(loadedImagePathPNG, mid,
effortMap[options.effort, (int)formatOptions.WEBP]),
              formatOptions.AVIF => await Backend.AVIF(loadedImagePathPNG, mid,
effortMap[options.effort, (int)formatOptions.AVIF]),
              formatOptions.JPEGXL => await Backend.JPEGXL(loadedImagePathPNG, mid,
effortMap[options.effort, (int)formatOptions.JPEGXL])
```

```
            };

            currentScore = await Backend.SSIM(loadedImagePathPNG, currentImagePath);
            Debug.WriteLine($"{activeFormats[i]} | {low} - {mid} - {high} | {currentScore}");
            if (currentScore > options.targetQuality) high = mid;
            else if (currentScore < options.targetQuality) low = mid;
            if (Math.Abs(currentScore - options.targetQuality) < leastDif) {
              leastDif = Math.Abs(currentScore - options.targetQuality);
              actualQuality = mid;
            }
                        File.Delete(currentImagePath);
                    }
        }

        // convert final time and actually time it
        Stopwatch sw = new Stopwatch();
        sw.Start();
        currentImagePath = activeFormats[i] switch {
          formatOptions.JPEG => await Backend.JPEG(loadedImagePathPNG, actualQuality),
          formatOptions.WEBP => await Backend.WEBP(loadedImagePathPNG, actualQuality,
effortMap[options.effort, (int)formatOptions.WEBP]),
          formatOptions.AVIF => await Backend.AVIF(loadedImagePathPNG, actualQuality,
effortMap[options.effort, (int)formatOptions.AVIF]),
          formatOptions.JPEGXL => await Backend.JPEGXL(loadedImagePathPNG, actualQuality,
effortMap[options.effort, (int)formatOptions.JPEGXL]),
          formatOptions.PNG => await Backend.PNG(loadedImagePathPNG, effortMap[options.effort,
(int)formatOptions.PNG]),
          formatOptions.LWEBP => await Backend.WEBP(loadedImagePathPNG, 100, effortMap[options.effort,
(int)formatOptions.LWEBP], true),
          formatOptions.LAVIF => await Backend.AVIF(loadedImagePathPNG, 100, effortMap[options.effort,
(int)formatOptions.LAVIF], true),
          formatOptions.LJPEGXL => await Backend.JPEGXL(loadedImagePathPNG, 100,
effortMap[options.effort, (int)formatOptions.LJPEGXL], true)
        };
        sw.Stop();
        currentTime = sw.ElapsedMilliseconds - 20; // account for overhead i guess

        // add file size label
        currentSize = new FileInfo(currentImagePath).Length;
        Grid gridSize = new Grid { RowSpacing = 8, VerticalAlignment = VerticalAlignment.Center };
        for (int j = 0; j < 3; j++) gridSize.RowDefinitions.Add(new RowDefinition { Height = new
GridLength(1, GridUnitType.Auto) });
        TextBlock labelSize = new TextBlock { Text = $"{Backend.sizeMakeReadable(currentSize)}",
HorizontalAlignment = HorizontalAlignment.Center, VerticalAlignment = VerticalAlignment.Center, Style
= (Style)Application.Current.Resources["BodyLargeTextBlockStyle"], Tag = currentSize };
        TextBlock labelSizeP = new TextBlock { Text = $"{(currentSize * 100f / loadedImageSize) -
100:+0.0;-0.0;-0}%", HorizontalAlignment = HorizontalAlignment.Center, VerticalAlignment =
VerticalAlignment.Center, Foreground = (Brush)Application.Current.Resources[currentSize >
loadedImageSize ? "SystemFillColorCriticalBrush" : "SystemFillColorSuccessBrush"] };
        borderSizes[i] = new Border { Width = 80, Height = 4, CornerRadius = new CornerRadius(2),
Background = (Brush)Application.Current.Resources["SystemFillColorSolidNeutralBackgroundBrush"] };
        gridSize.Children.Add(labelSize); Grid.SetRow(labelSize, 0);
```

```
        gridSize.Children.Add(labelSizeP); Grid.SetRow(labelSizeP, 2);
        gridSize.Children.Add(borderSizes[i]); Grid.SetRow(borderSizes[i], 1);
        gridFormats[i].Children.Add(gridSize); Grid.SetRow(gridSize, 1);

        // add similarity label
        Grid gridScore = new Grid { RowSpacing = 8, VerticalAlignment = VerticalAlignment.Center };
        for (int j = 0; j < 2; j++) gridScore.RowDefinitions.Add(new RowDefinition { Height = new
GridLength(1, GridUnitType.Auto) });
        TextBlock labelScore = new TextBlock { HorizontalAlignment = HorizontalAlignment.Center,
VerticalAlignment = VerticalAlignment.Center, Style =
(Style)Application.Current.Resources["BodyLargeTextBlockStyle"] };
        borderScores[i] = new Border { Width = 80, Height = 4, CornerRadius = new CornerRadius(2),
Background = (Brush)Application.Current.Resources["SystemFillColorSolidNeutralBackgroundBrush"] };
        gridScore.Children.Add(labelScore); Grid.SetRow(labelScore, 0);
        gridScore.Children.Add(borderScores[i]); Grid.SetRow(borderScores[i], 1);
        gridFormats[i].Children.Add(gridScore); Grid.SetRow(gridScore, 2);
        if (options.mode == convertMode.QUALITY) labelScore.Text = $"{currentScore:F1}";

        // add elapsed time label
        Grid gridTime = new Grid { RowSpacing = 8, VerticalAlignment = VerticalAlignment.Center };
        for (int j = 0; j < 2; j++) gridTime.RowDefinitions.Add(new RowDefinition { Height = new
GridLength(1, GridUnitType.Auto) });
        TextBlock labelTime = new TextBlock { Text = $"{currentTime:N0}ms", HorizontalAlignment =
HorizontalAlignment.Center, VerticalAlignment = VerticalAlignment.Center, Tag = currentTime };
        borderTimes[i] = new Border { Width = 80, Height = 4, CornerRadius = new CornerRadius(2),
Background = (Brush)Application.Current.Resources["SystemFillColorSolidNeutralBackgroundBrush"] };
        gridTime.Children.Add(labelTime); Grid.SetRow(labelTime, 0);
        gridTime.Children.Add(borderTimes[i]); Grid.SetRow(borderTimes[i], 1);
        gridFormats[i].Children.Add(gridTime); Grid.SetRow(gridTime, 3);

        // load converted image
        string imageKey = formatLabels[(int)activeFormats[i]];
        previewImages[imageKey] = new BitmapImage(new Uri(currentImagePath));
        fullImages[imageKey] = new BitmapImage(new Uri(currentImagePath));

        // add thumbnail image
        Border imageBorder = new Border { CornerRadius = new CornerRadius(8), HorizontalAlignment =
HorizontalAlignment.Center, VerticalAlignment = VerticalAlignment.Center };
        Image imageThumb = new Image { Source = previewImages[imageKey], Tag = imageKey };
        imageThumb.Tapped += OpenFullPreview;
        imageThumb.PointerEntered += Image_PointerEntered;
        imageThumb.PointerExited += Image_PointerExited;
        imageBorder.Child = imageThumb;
        gridFormats[i].Children.Add(imageBorder); Grid.SetRow(imageBorder, 4);

        Button buttonSave = new Button { Content = $"Save
{formatLabels[(int)activeFormats[i]].Replace(" Lossless", "")}", Tag =
formatLabels[(int)activeFormats[i]], HorizontalAlignment = HorizontalAlignment.Center };
        buttonSave.Click += SaveFormatImage;
        gridFormats[i].Children.Add(buttonSave); Grid.SetRow(buttonSave, 5);
```

```
        Button buttonFormat = new Button { Content = formatLabels[(int)activeFormats[i]], Tag =
formatLabels[(int)activeFormats[i]] };
        buttonFormat.Click += SelectFormatImage;
        StackPanelFormatButtons.Children.Add(buttonFormat);

        convertedImagePaths.Add(formatLabels[(int)activeFormats[i]], currentImagePath);
        allSizes[i] = currentSize;
        allTimes[i] = currentTime;
        allScores[i] = currentScore;
      }

      // set indicator colors for sizes and times
      for (int i = 0; i < borderSizes.Length; i++) {
        string color;
        if (allSizes[i] > allSizes.Max() - (allSizes.Max() - allSizes.Min()) / 5) color =
"SystemFillColorCriticalBrush";
        else if (allSizes[i] < allSizes.Min() + (allSizes.Max() - allSizes.Min()) / 10) color =
"SystemFillColorSuccessBrush";
        else color = "SystemFillColorCautionBrush";
        borderSizes[i].Background = (Brush)Application.Current.Resources[color];

        if (allTimes[i] > allTimes.Max() - (allTimes.Max() - allTimes.Min()) / 5) color =
"SystemFillColorCriticalBrush";
        else if (allTimes[i] < allTimes.Min() + (allTimes.Max() - allTimes.Min()) / 10) color =
"SystemFillColorSuccessBrush";
        else color = "SystemFillColorCautionBrush";
        borderTimes[i].Background = (Brush)Application.Current.Resources[color];
      }

      // add similarity scores
      if (options.mode ≠ convertMode.QUALITY) {
        for (int i = 0; i < activeFormats.Count; i++) {
          allScores[i] = await Backend.SSIM(loadedImagePathPNG,
convertedImagePaths[formatLabels[(int)activeFormats[i]]]);
          ((gridFormats[i].Children[2] as Grid).Children[0] as TextBlock).Text = $"{allScores[i]:F1}";
                }
      }

      // set indicator colors for similarity
      for (int i = 0; i < borderScores.Length; i++) {
        string color;
        if (allScores[i] < allScores.Min() + (allScores.Max() - allScores.Min()) / 5) color =
"SystemFillColorCriticalBrush";
        else if (allScores[i] ≥ allScores.Max() - (allScores.Max() - allScores.Min()) / 10) color =
"SystemFillColorSuccessBrush";
        else color = "SystemFillColorCautionBrush";
        borderScores[i].Background = (Brush)Application.Current.Resources[color];
      }

      // decide which result to highlight
      double[] eval = new double[gridFormats.Length];
      for (int i = 0; i < gridFormats.Length; i++) {
```

```
        eval[i] = allScores[i] - 8 * Math.Log(allSizes[i]);
      }
            for (int i = 0; i < gridFormats.Length; i++) {
                if (eval[i] == eval.Max()) gridFormats[i].Background =
(Brush)Application.Current.Resources["CardBackgroundFillColorDefaultBrush"];
            }

      foreach (UIElement element in gridLabels.Children) {
        if (element is Button) {
          (element as Button).IsEnabled = true;
        }
            }
    }

    private async void ButtonLoad_Click(object sender, RoutedEventArgs e) {
      FileOpenPicker picker = new();
      var hwnd = WindowNative.GetWindowHandle(App.MainAppWindow);
      InitializeWithWindow.Initialize(picker, hwnd);
      picker.FileTypeFilter.Add(".png");
      picker.FileTypeFilter.Add(".jpg");
      picker.FileTypeFilter.Add(".bmp");
      picker.FileTypeFilter.Add(".webp");
      picker.CommitButtonText = "Open";

      StorageFile file = await picker.PickSingleFileAsync();

      if (file == null) return;
      loadedImagePath = file.Path;
      loadedImageSize = new FileInfo(loadedImagePath).Length;
      await DisplayExifInfo(loadedImagePath);
      loadedImagePathPNG = await Backend.PNG(loadedImagePath, 5);
      fullImages["Source"] = new BitmapImage(new Uri(loadedImagePathPNG));
      previewImages["Source"] = new BitmapImage(new Uri(loadedImagePathPNG));
      ImageSrcPreview.Width = double.NaN; ImageSrcPreview.Height = double.NaN;
      ImageSrcPreview.Source = previewImages["Source"];

      StackPanelFormatButtons.Children.Clear();
            Button buttonSource = new Button { Content = "Original", Tag = "Source" };
            buttonSource.Click += SelectFormatImage;
            StackPanelFormatButtons.Children.Add(buttonSource);

            ButtonConvert.IsEnabled = true;
    }

    private async void ButtonConvert_Click(object sender, RoutedEventArgs e) {
      ButtonConvert.IsEnabled = false;
      ButtonLoad.IsEnabled = false;

      List<formatOptions> activeFormats = [];
      for (int i = SelectorBarLoss.SelectedItem.Text == "Lossless" ? 4 : 0; i <
(SelectorBarLoss.SelectedItem.Text == "Lossy" ? 4 : 8); i++) {
        if (selectedFormats[i]) activeFormats.Add((formatOptions)i);
```

```
      }
      await ExecuteConversion(conOptions, activeFormats);
      ButtonConvert.IsEnabled = true;
      ButtonLoad.IsEnabled = true;
    }

    private async void SaveFormatImage(object sender, RoutedEventArgs e) {
      string fmt = (string)(sender as Button).Tag;
      FileSavePicker picker = new();
      var hwnd = WindowNative.GetWindowHandle(App.MainAppWindow);
      InitializeWithWindow.Initialize(picker, hwnd);
      picker.FileTypeChoices.Add($"{fmt} ({Path.GetExtension(convertedImagePaths[fmt])})", new
List<string>() { Path.GetExtension(convertedImagePaths[fmt]) });
      picker.SuggestedFileName = $"{Path.GetFileNameWithoutExtension(loadedImagePath)} ({fmt})";
      StorageFile file = await picker.PickSaveFileAsync();

      if (file == null) return;
      StorageFile src = await StorageFile.GetFileFromPathAsync(convertedImagePaths[fmt]);
      src.CopyAndReplaceAsync(file);
    }

    private void SelectFormatImage(object sender, RoutedEventArgs e) {
      if (!fullImages.ContainsKey((sender as Button).Tag as string)) return;
      ImagePopup.Source = fullImages[(sender as Button).Tag as string];
      foreach (UIElement b in StackPanelFormatButtons.Children) {
        if (b is Button) {
          (b as Button).Style = (Style)Application.Current.Resources["DefaultButtonStyle"];
        }
      }
      (sender as Button).Style = (Style)Application.Current.Resources["AccentButtonStyle"];
    }

    private void SortColumns(object sender, RoutedEventArgs e) {
      byte[] order = new byte[GridCompare.Children.Count - 1];
      for (int i = 0; i < order.Length; i++) order[i] = (byte)i;

      if ((sender as Button).Tag == "Format") {
        for (int i = order.Length - 1; i > 0; i--) {
          for (int j = 0; j < i; j++) {
            if (Convert.ToInt16(((GridCompare.Children[order[j]] as Grid).Children[0] as
TextBlock).Tag) > Convert.ToInt16(((GridCompare.Children[order[j + 1]] as Grid).Children[0] as
TextBlock).Tag))
              (order[j], order[j + 1]) = (order[j + 1], order[j]);
          }
        }
      }
      else if ((sender as Button).Tag == "File Size") {
                for (int i = order.Length - 1; i > 0; i--) {
                    for (int j = 0; j < i; j++) {
            if (Convert.ToInt64((((GridCompare.Children[order[j]] as Grid).Children[1] as
Grid).Children[0] as TextBlock).Tag) > Convert.ToInt64((((GridCompare.Children[order[j + 1]] as
Grid).Children[1] as Grid).Children[0] as TextBlock).Tag)) {
```

```
                (order[j], order[j + 1]) = (order[j + 1], order[j]);
                        }
                    }
                }
            }
      else if ((sender as Button).Tag == "Similarity") {
                for (int i = order.Length - 1; i > 0; i--) {
                    for (int j = 0; j < i; j++) {
            if (Convert.ToDouble((((GridCompare.Children[order[j]] as Grid).Children[2] as
Grid).Children[0] as TextBlock).Text) < Convert.ToDouble((((GridCompare.Children[order[j + 1]] as
Grid).Children[2] as Grid).Children[0] as TextBlock).Text)) {
                (order[j], order[j + 1]) = (order[j + 1], order[j]);
                        }
                    }
                }
            }
      else if ((sender as Button).Tag == "Encoding time") {
                for (int i = order.Length - 1; i > 0; i--) {
                    for (int j = 0; j < i; j++) {
            if (Convert.ToInt32((((GridCompare.Children[order[j]] as Grid).Children[3] as
Grid).Children[0] as TextBlock).Tag) > Convert.ToInt32((((GridCompare.Children[order[j + 1]] as
Grid).Children[3] as Grid).Children[0] as TextBlock).Tag)) {
                (order[j], order[j + 1]) = (order[j + 1], order[j]);
                        }
                    }
                }
            }

      for (int i = 0; i < order.Length; i++)
        Grid.SetColumn((FrameworkElement)GridCompare.Children[order[i]], i + 1);
    }

    private void SliderQuality_ValueChanged(object sender, RangeBaseValueChangedEventArgs e) {
      if (NumberBoxQuality == null) return;
      conOptions.quality = Convert.ToByte(SliderQuality.Value);
      NumberBoxQuality.Value = conOptions.quality;
    }

    private void NumberBoxQuality_ValueChanged(NumberBox sender, NumberBoxValueChangedEventArgs args)
{
      if (SliderQuality == null) return;
      conOptions.quality = Convert.ToByte(NumberBoxQuality.Value);
      SliderQuality.Value = conOptions.quality;
    }

    private void ComboBoxEffort_SelectionChanged(object sender, SelectionChangedEventArgs e) {
      conOptions.effort = Convert.ToByte(ComboBoxEffort.SelectedIndex);
    }

    private void NumberBoxTargetSize_ValueChanged(NumberBox sender, NumberBoxValueChangedEventArgs
args) {
      if (ComboBoxTargetSizeUnits == null) return;
```

```
      conOptions.targetSize = Convert.ToInt32(NumberBoxTargetSize.Value) *
(ComboBoxTargetSizeUnits.SelectedIndex == 0 ? 1024 : 1024 * 1024);
    }

    private void ComboBoxTargetSizeUnits_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
      if (NumberBoxTargetSize == null) return;
      conOptions.targetSize = Convert.ToInt32(NumberBoxTargetSize.Value) *
(ComboBoxTargetSizeUnits.SelectedIndex == 0 ? 1024 : 1024 * 1024);
        }

    private void ComboBoxMode_SelectionChanged(object sender, SelectionChangedEventArgs e) {
      if (GridSettingEffort == null || GridSettingQuality == null || GridSettingTargetSize == null ||
GridSettingTargetQuality == null) return;
      GridSettingQuality.Visibility = Visibility.Collapsed;
      GridSettingTargetSize.Visibility = Visibility.Collapsed;
      GridSettingTargetQuality.Visibility = Visibility.Collapsed;

      switch(ComboBoxMode.SelectedIndex) {
        case 0: {
          conOptions.mode = convertMode.MANUAL;
          GridSettingQuality.Visibility = Visibility.Visible; break;
        }
        case 1: {
          conOptions.mode = convertMode.QUALITY;
          GridSettingTargetQuality.Visibility = Visibility.Visible; break;
        }
        case 2: {
          conOptions.mode = convertMode.SIZE;
          GridSettingTargetSize.Visibility = Visibility.Visible; break;
        }
      }
    }

    private void SliderTargetQuality_ValueChanged(object sender, RangeBaseValueChangedEventArgs e) {
      if (NumberBoxTargetQuality == null) return;
      conOptions.targetQuality = Convert.ToByte(SliderTargetQuality.Value);
      NumberBoxTargetQuality.Value = conOptions.targetQuality;
    }

    private void NumberBoxTargetQuality_ValueChanged(NumberBox sender, NumberBoxValueChangedEventArgs
args) {
      if (SliderTargetQuality == null) return;
      conOptions.targetQuality = Convert.ToByte(NumberBoxTargetQuality.Value);
      SliderTargetQuality.Value = conOptions.targetQuality;
    }

    private void ButtonFormat_Click(object sender, RoutedEventArgs args) {
      ToggleButton btn = sender as ToggleButton;
      selectedFormats[(int)(btn.Tag)] = (bool)btn.IsChecked;
    }
```

```
    private void SelectorBarLoss_SelectionChanged(SelectorBar sender,
SelectorBarSelectionChangedEventArgs args) {
      foreach (SelectorBarItem item in SelectorBarLoss.Items) {
        item.FontWeight = item == SelectorBarLoss.SelectedItem ? FontWeights.Bold :
FontWeights.Normal;
      }

      // add format selection buttons
      GridFormatSelection.ColumnDefinitions.Clear();
      GridFormatSelection.RowDefinitions.Clear();
      GridFormatSelection.Children.Clear();
      for (int i = 0; i < 2; i++) GridFormatSelection.ColumnDefinitions.Add(new ColumnDefinition
{ Width = new GridLength(1, GridUnitType.Star) });
      for (int i = 0; i < (SelectorBarLoss.SelectedItem.Text == "All" ? 4 : 2); i++)
GridFormatSelection.RowDefinitions.Add(new RowDefinition { Height = new GridLength(1,
GridUnitType.Star) });
      for (int i = 0; i < (SelectorBarLoss.SelectedItem.Text == "All" ? 8 : 4); i++) {
        ToggleButton btn = new ToggleButton { Content = formatLabels[i +
(SelectorBarLoss.SelectedItem.Text == "Lossless" ? 4 : 0)], Tag = i +
(SelectorBarLoss.SelectedItem.Text == "Lossless" ? 4 : 0), HorizontalAlignment =
HorizontalAlignment.Stretch, IsChecked = selectedFormats[i + (SelectorBarLoss.SelectedItem.Text ==
"Lossless" ? 4 : 0)] };
        btn.Click += ButtonFormat_Click;
        GridFormatSelection.Children.Add(btn);
        Grid.SetRow(btn, i / GridFormatSelection.ColumnDefinitions.Count); Grid.SetColumn(btn, i %
GridFormatSelection.ColumnDefinitions.Count);
      }

      if (SelectorBarLoss.SelectedItem.Text == "Lossy") {
        GridSettingMode.Visibility = Visibility.Visible;
      }
      else {
        GridSettingMode.Visibility = Visibility.Collapsed;
        ComboBoxMode.SelectedIndex = 0;
      }

      if (SelectorBarLoss.SelectedItem.Text == "Lossless") GridSettingQuality.Visibility =
Visibility.Collapsed;
      else if (ComboBoxMode.SelectedIndex == 0) GridSettingQuality.Visibility = Visibility.Visible;
    }

    private void Image_PointerEntered(object sender, PointerRoutedEventArgs e) {
      (sender as UIElement).Opacity = 0.8;
    }

    private void Image_PointerExited(object sender, PointerRoutedEventArgs e) {
      (sender as UIElement).Opacity = 1.0;
    }
  }
}
```