МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«ОПТИМІЗАЦІЯ GIF-АНІМАЦІЙ: РОЗРОБКА ІНСТРУМЕНТУ ТА АНАЛІЗ АЛГОРИТМІВ КВАНТУВАННЯ КОЛЬОРУ»**

Виконав: студент 4 курсу, групи 6.1222
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
В.С. Кляшторний
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, доцент, к.ф.-м.н Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних наук, к.пед.н., доцент

_____ Пшенична О.С.
(підпис)

" 08 " 12 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кляшторному Владиславу Сергійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Оптимізація GIF-анімацій: розробка інструменту та аналіз алгоритмів квантування кольору

керівник роботи Решевська Катерина Сергіївна, к. т. н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 04.06.2026

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Дослідження теоретичних основ оптимізації GIF-анімацій.
2. Розробка програмного інструменту для створення та оптимізації GIF-файлів.
3. Аналіз результатів роботи розробленої програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання 05.12.2025

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	04.04.2026	
2.	Дослідження теоретичних основ	14.04.2026	
	оптимізації GIF-анімацій.		
3.	Огляд алгоритмів квантування кольору.	25.04.2026	
4.	Проєктування структури та функціоналу	06.05.2026	
	програмного інструменту.		
5.	Розробка програми для створення та	15.05.2026	
	оптимізації GIF-файлів.		
6.	Тестування програми та аналіз результатів.	18.05.2026	
7.	Оформлення та нормоконтроль	28.05.2026	
	кваліфікаційної роботи бакалавра.		
8.	Захист кваліфікаційної роботи.	19.06.2026	

Студент _____ В.С. Кляшторний
(підпис) (ініціали та прізвище)

Керівник роботи _____ К.С. Решевська
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Оптимізація GIF-анімацій: розробка інструменту та аналіз алгоритмів квантування кольору»: 62 с., 10 рис., 3 табл., 16 джерел, 3 додатків.

АЛГОРИТМИ КВАНТУВАННЯ КОЛЬОРУ, КОНВЕРТАЦІЯ ВІДЕО, ОПТИМІЗАЦІЯ ФАЙЛІВ, ПАЛІТРА КОЛЬОРІВ, ПРОГРАМНИЙ ІНСТРУМЕНТ, СТИСНЕННЯ ЗОБРАЖЕНЬ, GIF-АНІМАЦІЯ.

Об’єкт дослідження – процес створення та оптимізації GIF-анімацій на основі відеофайлів.

Предмет дослідження – алгоритми квантування кольору, методи оптимізації GIF-файлів та програмні засоби керування параметрами обробки анімацій.

Мета роботи: розробка програмного інструменту для створення та оптимізації GIF-анімацій, а також аналіз ефективності алгоритмів квантування кольору.

Методи дослідження – аналіз, порівняння, синтез, моделювання, експериментальне тестування.

Результати дослідження – розроблено програмний інструмент для конвертації відео у GIF та оптимізації GIF-анімацій із можливістю налаштування основних параметрів обробки.

У кваліфікаційній роботі розглянуто особливості формату GIF, підходи до його оптимізації та алгоритми квантування кольору. Розроблено програмний інструмент для конвертації відео у GIF з можливістю налаштування кількості кольорів, частоти кадрів, роздільної здатності та рівня стиснення. Результати дослідження дають змогу оцінити ефективність алгоритмів квантування кольору для практичної оптимізації GIF-анімацій.

SUMMARY

Bachelor's Qualifying Thesis « Optimization of GIF Animations: Development of a Tool and Analysis of Color Quantization Algorithms»: 62 pages, 10 figures, 3 tables, 16 references, 3 supplements.

COLOR QUANTIZATION ALGORITHMS, VIDEO CONVERSION, FILE OPTIMIZATION, COLOR PALETTE, SOFTWARE TOOL, IMAGE COMPRESSION, GIF ANIMATION.

Object of the study – the process of creating and optimizing GIF animations based on video files.

Subject of the study – color quantization algorithms, methods of GIF optimization, and software tools for controlling animation processing parameters.

Aim of the study: to develop a software tool for creating and optimizing GIF animations and to analyze the effectiveness of color quantization algorithms.

Methods of research – analysis, comparison, synthesis, modeling, experimental testing.

Results of the study – a software tool for video-to-GIF conversion and GIF optimization with configurable processing parameters has been developed.

The bachelor's thesis examines the features of the GIF format, approaches to its optimization, and color quantization algorithms. A software tool for video-to-GIF conversion was developed with support for adjusting the number of colors, frame rate, resolution, and compression level. The results of the study make it possible to evaluate the effectiveness of color quantization algorithms for the practical optimization of GIF animations.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 7
1 Теоретичні основи оптимізації GIF-анімацій..... 8
1.1 Особливості формату GIF та принципи його роботи..... 8
1.2 Основні підходи до оптимізації GIF-анімацій 9
1.3 Квантування кольору як ключовий етап зменшення розміру GIF..... 11
1.4 Огляд поширених алгоритмів квантування кольору..... 12
1.4.1 Алгоритм Median Cut..... 13
1.4.2 Алгоритм Octree 14
1.4.3 Алгоритм NeuQuant 15
1.4.4 Інші підходи та їх практичні обмеження..... 16
1.5 Критерії оцінювання ефективності оптимізації GIF-анімацій 17
1.6 Висновки до розділу 1 18
2 Розробка інструменту для оптимізації GIF-анімацій 20
2.1 Постановка задачі розробки програмного інструменту..... 20
2.2 Визначення функціональних і нефункціональних вимог 21
2.3 Вибір засобів і технологій реалізації 24
2.4 Архітектура розробленого інструменту 25
2.5 Реалізація модуля обробки кадрів GIF-анімації 27
2.6 Реалізація модуля квантування кольору..... 29
2.7 Реалізація модуля оцінювання результатів оптимізації..... 32
2.8 Опис інтерфейсу користувача та сценарію використання програми 34
2.9 Особливості програмної реалізації та обмеження розробленого рішення 37
2.10 Висновки до розділу 2 39

3 Аналіз ефективності алгоритмів квантування кольору 41
3.1 Організація експериментального дослідження 41
3.2 Вибір тестових відеофрагментів і параметрів порівняння 42
3.3 Порівняльний аналіз розміру GIF-файлів після оптимізації 43
3.4 Аналіз візуальної якості результатів квантування 45
3.5 Аналіз швидкодії розробленого інструменту 47
3.6 Вибір доцільного алгоритму та рекомендації щодо оптимізації GIF-анімацій 49
3.7 Висновки до розділу 3 50
Висновки 52
Перелік посилань 53
Додаток А Інтерфейс розробленого програмного інструменту 55
Додаток Б Фрагмент коду програмної реалізації алгоритмів квантування кольору (PaletteQuantizer.cs) 56
Додаток В Фрагмент коду програмної реалізації модуля обробки відеофайлів та створення GIF-анімацій (FfmpegService.cs) 59

ВСТУП

Сьогодні цифровий контент відіграє ключову роль у більшості сфер діяльності, а короткі анімовані зображення у форматі GIF досі залишаються одним із найпопулярніших способів передачі візуальної інформації в інтернеті. Їх активно використовують у соціальних мережах, на сайтах, в рекламі, месенджерах та онлайн-сервісах. Такий формат зручний для швидкого поширення, проте має суттєві технічні обмеження, насамперед щодо якості зображення та розміру файлу.

Незважаючи на велику кількість сучасних програм для роботи з графікою та відео, повноцінна й зручна робота саме з GIF-файлами часто залишається недосконалою. Навіть професійні інструменти не завжди дозволяють гнучко керувати ключовими параметрами анімації, такими як кількістю кольорів, частотою кадрів, роздільною здатністю, ступенем стиснення та остаточним розміром файлу. Через це користувачам доводиться постійно шукати компроміс між якістю та вагою або застосовувати кілька різних програм одночасно.

Особливо важливу роль у процесі оптимізації GIF відіграє квантування кольору. Саме воно безпосередньо впливає як на візуальне сприйняття зображення, так і на кінцевий розмір файлу. Від вибору алгоритму квантування значною мірою залежить результат, тому їх аналіз має не лише теоретичну, а й практичну цінність.

Актуальність теми зумовлена потребою у створенні зручного програмного інструменту для створення та оптимізації GIF-анімацій, а також необхідністю глибшого дослідження алгоритмів квантування кольору. Це дозволить не тільки спростити роботу з GIF, але й визначити найефективніші підходи до їх оптимізації залежно від конкретних завдань.

1 ТЕОРЕТИЧНІ ОСНОВИ ОПТИМІЗАЦІЇ GIF-АНІМАЦІЙ

1.1 Особливості формату GIF та принципи його роботи

Формат GIF є одним із найвідоміших і довговічніших графічних форматів, який і досі широко використовується для простих зображень та коротких анімацій. Створений наприкінці 1980-х, він залишається популярним завдяки простоті, відмінній сумісності з більшістю платформ і можливості автоматичного відтворення без додаткового програмного забезпечення. GIF активно застосовують для мемів, коротких анімованих роликів, рекламних банерів, демонстрацій інтерфейсів та іншого динамічного контенту, де важливі компактність і швидке завантаження.

Основна особливість формату полягає в тому, що це растровий формат із палітровим кольором. Зображення зберігається не в повному RGB, а за допомогою таблиці кольорів (палітри), де кожному відтінку відповідає індекс. У стандартному GIF один кадр може містити максимум 256 кольорів. Це значно зменшує розмір файлу, але суттєво обмежує якість передачі складних зображень, зокрема фотографій, градієнтів чи відеокадрів із великою кількістю напівтонів. Через це при конвертації відео в GIF часто з’являється зернистість, спрощення деталей і втрата плавності кольорових переходів.

Ще одна ключова особливість формату полягає в підтримці анімації. GIF може містити послідовність кадрів, які відтворюються з певною затримкою, створюючи ефект руху. Анімація може програватися один раз або зациклюватися. Кожен кадр може мати власну палітру, хоча частіше використовують одну глобальну на весь файл. При цьому формат не підтримує звук, тому призначений виключно для візуального контенту.

Для стиснення даних у GIF застосовується алгоритм LZW, який добре працює з зображеннями, що містять багато однакових ділянок і обмежену кількість кольорів. Саме тому формат ідеально підходить для іконок, схем,

простої графіки та анімацій з невеликими змінами між кадрами. Однак при складній деталізації, плавних градієнтах чи високій частоті кадрів розмір файлу швидко зростає, а якість після зменшення палітри помітно падає.

Окремо варто згадати прозорість. GIF підтримує лише один повністю прозорий колір без напівпрозорості. На відміну від PNG чи WebP, тут неможливо зробити м’які тіні чи акуратне накладання. Для багатьох простих анімацій цього вистачає, але в складніших випадках це суттєве обмеження.

До сильних сторін GIF належать універсальність, широка підтримка браузерами, простота створення та поширення. Серед недоліків слід виділити обмеження на 256 кольорів, відсутність звуку, відносно великий розмір при складній анімації та сильна залежність кінцевого результату від правильної оптимізації.

Таким чином, GIF поєднує простоту та практичність, але його технічні обмеження безпосередньо впливають на якість і розмір анімації. Саме тому питання оптимізації, а саме зменшення кількості кольорів, вибір алгоритму квантування, налаштування частоти кадрів та роздільної здатності, набувають особливого значення.

1.2 Основні підходи до оптимізації GIF-анімацій

Оптимізація GIF-анімацій стає ключовим етапом підготовки контенту до використання в інтернеті. Зазвичай користувач хоче отримати файл, який не тільки добре виглядає, але й має прийнятний розмір, швидко завантажується та зберігає достатню якість. Тому головне завдання полягає в тому, щоб знайти баланс між якістю зображення, плавністю руху та обсягом файлу.

Найпоширеніший спосіб оптимізації є зменшення кількості кольорів. Оскільки GIF обмежений 256 кольорами на кадр, правильний підбір палітри дозволяє суттєво скоротити розмір без сильної втрати якості. Однак надто агресивне скорочення призводить до різких переходів, зернистості та втрати

дрібних деталей. Тому цей метод вважається одним із найефективніших, але й найбільш ризикованих.

Важливим інструментом є також зміна частоти кадрів. Більша кількість FPS робить анімацію плавнішою, але сильно збільшує розмір файлу. У багатьох випадках надмірно висока частота не дає помітного візуального ефекту, особливо в коротких роликах.

Часто застосовують і зменшення роздільної здатності. Чим менше пікселів у кадрі, тим менший файл. Цей підхід добре працює для GIF, які показують у невеликих вікнах на сайтах, у месенджерах чи соціальних мережах. Занадто сильне масштабування, звичайно, розмиває деталі, тому тут теж потрібна обережність.

Додатковий ефект дає скорочення тривалості анімації та видалення зайвих кадрів. Не всі фрагменти відео однаково важливі. Часто достатньо залишити лише ключові моменти, щоб зберегти сенс і динаміку. Те саме стосується обрізання непотрібних ділянок кадру, видалення фону чи другорядних елементів.

Добре працює оптимізація міжкадрових змін. Якщо велика частина зображення залишається статичною, зберігають лише змінені області. Такий підхід особливо ефективний для анімацій з нерухомим фоном або рухом окремих об’єктів.

Не менш важливі налаштування палітри, алгоритму стиснення та використання дизерингу. Різні методи формування палітри навіть за однакової кількості кольорів дають різний результат. Дизеринг допомагає імітувати проміжні відтінки, але іноді додає «шум» і збільшує розмір.

Таким чином, ефективна оптимізація GIF рідко ґрунтується на одному методі. Найкращий результат дає комплексне поєднання зменшення кольорів, частоти кадрів, роздільної здатності, тривалості та розумної роботи з міжкадровими змінами. Саме такий підхід дозволяє суттєво зменшити розмір файлу з мінімальною втратою якості.

1.3 Квантування кольору як ключовий етап зменшення розміру GIF

Квантування кольору є одним із найважливіших етапів створення та оптимізації GIF-анімацій. Саме від нього значною мірою залежить, як виглядатиме фінальна анімація, наскільки точно передадуться відтінки та яким буде розмір файлу. Для GIF цей процес є критичним, адже формат дозволяє використовувати лише до 256 кольорів в одному кадрі. Тому будь-яке повноколірне зображення чи відеокадр перед збереженням обов’язково має пройти скорочення палітри.

Суть квантування полягає в зменшенні кількості унікальних кольорів до заданого рівня. Замість тисяч або мільйонів відтінків, які містить оригінальне зображення, формується значно менша палітра, а кожен піксель замінюється на найближчий колір із цієї палітри. Це головний спосіб суттєво зменшити обсяг даних.

Чим менша палітра, тим менший розмір файлу. Однак надмірне скорочення кольорів призводить до втрати природності: з’являються різкі переходи, зернистість, зникають дрібні деталі, а градієнти стають ламаними або плямистими. Тому квантування завжди є пошуком компромісу між розміром і візуальною якістю.

Особливо помітно це проявляється при конвертації відео в GIF. Відеокадри багаті на напівтони, тіні та складні кольорові переходи. Після квантування частина цієї інформації втрачається, і результат сильно залежить від того, наскільки вдало алгоритм обрав ключові кольори. Правильно сформована палітра дозволяє зберегти загальну виразність навіть при значному скороченні, а невдалий вибір псує якість навіть за відносно великого розміру файлу.

Процес квантування зазвичай включає три основні етапи: аналіз початкової кольорової інформації, формування нової зменшеної палітри та заміна кольорів пікселів. Саме на етапі створення палітри найбільше проявляються відмінності між різними алгоритмами.

Для покращення результату часто застосовують дизеринг, тобто метод імітації відсутніх відтінків через чергування сусідніх кольорів. Він робить переходи м’якшими, але може створювати додатковий «шум» і трохи збільшувати розмір файлу, тому його потрібно використовувати обережно.

Отже, квантування кольору є не просто технічною операцією, а центральним елементом оптимізації GIF. Від вибору алгоритму та параметрів палітри безпосередньо залежить баланс між якістю та розміром анімації. Саме тому детальне дослідження алгоритмів квантування становить ключову частину цієї роботи.

1.4 Огляд поширених алгоритмів квантування кольору

Для зменшення кількості кольорів у зображеннях існує кілька алгоритмів квантування, кожен із яких працює за власним принципом і має свої сильні та слабкі сторони. Вибір алгоритму суттєво впливає не тільки на візуальну якість GIF, а й на швидкість обробки та ефективність стиснення. Тому важливо враховувати не лише фінальну кількість кольорів, а й те, як саме формується палітра.

Найпоширенішими алгоритмами в роботі з GIF вважаються Median Cut, Octree та NeuQuant. Вони добре зарекомендували себе завдяки відносно помірним вимогам до ресурсів і прийнятній якості результату.

Median Cut є один із найпростіших і найпоширеніших алгоритмів. Він послідовно ділить кольоровий простір на менші області, групуючи схожі кольори, а потім обирає представницький колір для кожної з них. Алгоритм досить швидкий і дає збалансований результат у більшості випадків. Однак він може пропускати рідкісні, але важливі відтінки в складних зображеннях.

Octree використовує деревоподібну структуру для групування кольорів. Завдяки цьому він ефективно працює з великими обсягами даних і добре

підходить для обробки багатьох кадрів. Алгоритм показує хорошу швидкодію, хоча якість результату може залежати від конкретної реалізації.

NeuQuant працює на принципах самоорганізованої нейронної мережі. Замість простого поділу простору він «навчається» на зображенні та намагається знайти найбільш характерні кольори. Завдяки цьому часто дає найкращу візуальну якість, особливо на складних зображеннях із градієнтами та великою кількістю напівтонів. Головний недолік полягає у значно вищій обчислювальна складність і довший час обробки.

Окрім цих трьох, існують й інші методи, зокрема різні варіанти кластеризації. Вони можуть давати кращий результат для певних типів зображень, але частіше за все поступаються за швидкістю, тому рідше використовуються в практичних GIF-інструментах.

Жоден алгоритм не є універсально найкращим. Один краще зберігає плавні переходи, інший швидше працює з анімацією, третій ефективніше зменшує розмір файлу. Тому вибір алгоритму завжди має залежати від конкретних вимог: пріоритету швидкості, якості чи мінімального розміру.

Таким чином, порівняння цих алгоритмів дозволяє зрозуміти їхні сильні сторони та обмеження і визначити, які з них найбільш доцільно використовувати в інструменті оптимізації GIF-анімацій.

1.4.1 Алгоритм Median Cut

Алгоритм Median Cut є одним із найвідоміших і найчастіше використовуваних методів квантування кольору. Його популярність зумовлена відносною простотою реалізації, зрозумілим принципом роботи та доброю якістю результату в багатьох практичних задачах. Саме тому його часто застосовують як базовий метод при створенні палітри для GIF-анімацій.

Суть алгоритму полягає в послідовному поділі кольорового простору на області зі схожими кольорами. Спочатку всі кольори зображення розглядаються як єдина множина. Потім обирається колірний канал з найбільшим діапазоном

значень (R, G чи B), і множина ділиться навпіл за медіаною. Цей процес повторюється доти, доки не утвориться потрібна кількість груп. Для кожної групи визначається представницький колір, який і стає частиною фінальної палітри.

Головною перевагою Median Cut є швидкість роботи та відносно рівномірний розподіл кольорів. Алгоритм добре справляється з більшістю звичайних зображень, вимагає небагато ресурсів і дозволяє швидко обробляти велику кількість кадрів. Завдяки цьому він часто використовується, коли потрібна оперативна оптимізація.

Водночас алгоритм має помітні недоліки. Він не враховує особливості людського зору і погано зберігає рідкісні, але візуально важливі відтінки. У складних зображеннях із градієнтами, напівтонами чи дрібними деталями результат може виглядати штучно, з’являються різкі переходи та втрата природності. Також якість знижується, коли одну палітру доводиться формувати для багатьох різнорідних кадрів анімації.

У контексті оптимізації GIF-анімацій Median Cut можна розглядати як практичний компроміс між швидкістю та якістю. Він не є найкращим у всіх ситуаціях, але добре підходить як базовий алгоритм для порівняння з більш просунутими методами.

1.4.2 Алгоритм Octree

Алгоритм Octree є одним із поширених методів квантування кольору, який активно використовується для створення обмеженої палітри. Його головна особливість полягає в застосуванні деревоподібної структури даних, що дозволяє впорядковувати кольори за їхнім розташуванням у кольоровому просторі. Завдяки цьому алгоритм забезпечує досить високу швидкість роботи і добре підходить для задач оптимізації GIF-анімацій.

Принцип роботи Octree полягає в послідовному розміщенні всіх кольорів зображення в ієрархічному дереві. Кожен вузол дерева відповідає певній області

кольорового куба і розбивається на вісім гілок залежно від значень компонентів R, G та B. У процесі додавання пікселів формуються групи близьких кольорів. Якщо кількість кольорів перевищує потрібну, дерево спрощується шляхом об’єднання вузлів, що в результаті дає компактну палітру.

Серед головних переваг Octree слід виділити високу продуктивність і ефективна робота з великими обсягами даних. Алгоритм особливо зручний при обробці цілих анімацій, оскільки добре справляється з великою кількістю кадрів. Він природно групує схожі відтінки, тому добре працює з нескладною графікою, інтерфейсними елементами та анімаціями з відносно однорідними сценами.

Водночас Octree має і недоліки. Якість результату сильно залежить від способу побудови та скорочення дерева. У складних зображеннях із численними дрібними переходами та градієнтами алгоритм часто поступається за точністю передачі кольорів більш адаптивним методам. Він також не завжди найкраще зберігає візуально важливі, але рідкісні відтінки.

У контексті оптимізації GIF-анімацій Octree можна розглядати як ефективний компромісний варіант, коли пріоритетом є швидкість обробки та робота з великою кількістю кадрів при помірних вимогах до якості.

1.4.3 Алгоритм NeuQuant

Алгоритм NeuQuant вважається одним із найпрогресивніших методів квантування кольору, особливо коли головним пріоритетом є висока візуальна якість. На відміну від простіших алгоритмів, він працює на принципах самоорганізованої нейронної мережі, яка поступово адаптується до особливостей конкретного зображення.

Принцип роботи NeuQuant полягає в тому, що спочатку створюється набір «нейронів», кожен з яких представляє певний колір. У процесі навчання ці нейрони коригуються відповідно до реального розподілу кольорів у зображенні. В результаті формується палітра, яка добре передає найбільш характерні та візуально важливі відтінки оригіналу.

Головною перевагою алгоритму є значно вища якість результату порівняно з Median Cut та Octree. NeuQuant краще зберігає плавні кольорові переходи, градієнти та напівтони, завдяки чому GIF-анімації, особливо конвертовані з відео, виглядають природніше навіть при обмеженій кількості кольорів.

Однак за цю якість доводиться платити швидкістю. Алгоритм є складнішим у обчислені і працює помітно повільніше, особливо при обробці великої кількості кадрів. Тому він не завжди зручний для швидкої пакетної оптимізації.

NeuQuant найкраще проявляє себе на складних зображеннях із багатою кольоровою гамою, м’якими переходами та великою кількістю деталей. У таких випадках він часто дає відчутно кращий візуальний результат.

Отже, алгоритм NeuQuant доцільно розглядати як перспективний варіант для задач, де якість зображення важливіша за швидкість обробки. Його порівняння з іншими методами дозволить визначити, в яких ситуаціях використання складнішого алгоритму є виправданим для практичного інструменту оптимізації GIF.

1.4.4 Інші підходи та їх практичні обмеження

Окрім трьох основних алгоритмів (Median Cut, Octree та NeuQuant), у квантуванні кольору використовують і інші методи: різні алгоритми кластеризації, статистичні підходи, комбіновані схеми та модифікації вже відомих алгоритмів. Деякі з них дають хороші результати в певних ситуаціях, проте на практиці їх застосування для оптимізації GIF-анімацій часто виявляється недоцільним.

Алгоритми кластеризації дозволяють гнучко групувати кольори відповідно до їх реального розподілу в зображенні, що іноді забезпечує високу точність. Однак вони вимогливі до обчислювальних ресурсів і погано підходять для швидкої обробки великої кількості кадрів.

Існують також методи, які намагаються враховувати особливості людського зору та зберігати найпомітніші для людини відтінки. Теоретично це повинно покращувати візуальне сприйняття, але на практиці виграш часто не виправдовує суттєве ускладнення алгоритму.

Комбіновані підходи поєднують кілька методів одночасно, наприклад, один для аналізу та іншого для формування фінальної палітри. Такі рішення можуть бути ефективними, але значно ускладнюють реалізацію, налаштування та подальше тестування.

Головними практичними обмеженнями альтернативних методів є висока обчислювальна складність, низька швидкодія, складність реалізації та нестабільність результату при роботі з різнорідними анімаціями. Багато з них також погано представлені в реальних GIF-інструментах через труднощі інтеграції.

Тому в межах цієї роботи основну увагу зосереджено саме на алгоритмах Median Cut, Octree та NeuQuant. Вони являють собою оптимальний баланс між зрозумілістю, швидкістю роботи та якістю результату, що робить їх найбільш придатними для порівняльного аналізу та подальшої реалізації в практичному інструменті оптимізації GIF-анімацій.

1.5 Критерії оцінювання ефективності оптимізації GIF-анімацій

Оцінювання ефективності оптимізації GIF-анімацій є важливим етап дослідження. Одного показника (наприклад, лише розміру файлу) недостатньо. Справді ефективна оптимізація повинна забезпечувати хороший баланс між якістю анімації та її практичними характеристиками.

Основним критерієм є розмір кінцевого файлу. Зменшення обсягу безпосередньо впливає на швидкість завантаження та зручність використання в інтернеті. Однак надто сильне стиснення майже завжди призводить до помітної втрати якості.

Другим ключовим критерієм є візуальна якість зображення. Після оптимізації анімація має зберігати чіткість основних об’єктів, природність кольорів і прийнятну деталізацію. Якщо з’являються сильні спотворення, зернистість або різкі переходи, результат не можна вважати вдалим.

Окремо варто оцінювати збереження кольорів. Оскільки GIF обмежений 256 кольорами, важливо, щоб після квантування залишилися найбільш значущі відтінки, а не виникали різкі переходи, плямистість чи втрата дрібних деталей.

Важливим показником є також плавність анімації. Зниження частоти кадрів або видалення проміжних кадрів може зробити рух нерівномірним, що погіршує загальне сприйняття навіть за хорошої якості окремих кадрів.

Для практичного інструменту суттєве значення має швидкість обробки. Алгоритм може давати відмінну якість, але якщо обробка триває надто довго, особливо при роботі з великими файлами чи пакетній оптимізації, його цінність зменшується.

Також стабільність результату відіграє значну роль, адже алгоритм повинен показувати передбачувану якість на різних типах анімацій: простих, динамічних, з градієнтами чи насичених кольорами.

Таким чином, ефективність оптимізації GIF-анімацій потрібно оцінювати комплексно за сукупністю критеріїв: розмір файлу, візуальна якість, збереження кольорів, плавність руху, швидкість обробки та стабільність.

1.6 Висновки до розділу 1

У першому розділі розглянуто теоретичні основи роботи з GIF-анімаціями та їх оптимізації. З’ясовано, що попри значний вік формату, GIF досі залишається затребуваним завдяки простоті використання, універсальній сумісності та зручності поширення в інтернеті. Водночас формат має суттєві обмеження: максимум 256 кольорів на кадр, відсутність звуку та швидке зростання розміру файлу при складній анімації.

Аналіз основних підходів до оптимізації показав, що найбільший ефект дає комплексне застосування кількох методів: зменшення кількості кольорів, коригування частоти кадрів, роздільної здатності, тривалості анімації та оптимізація міжкадрових змін. Ефективна оптимізація майже ніколи не зводиться до одного інструменту, а вимагає врахування особливостей конкретного матеріалу та поставлених завдань.

Особливу увагу приділено квантуванню кольору, оскільки саме цей етап найбільше впливає на кінцевий розмір файлу та візуальну якість анімації. Було розглянуто найпоширеніші алгоритми квантування: Median Cut, Octree та NeuQuant. Кожен із них має свої сильні сторони: одні вирізняються швидкістю, тоді як інші забезпечують кращим збереженням кольорів і природністю зображення.

Також визначено основні критерії оцінювання ефективності оптимізації: розмір кінцевого файлу, візуальну якість, ступінь збереження кольорів, плавність анімації, швидкість обробки та стабільність результатів. Саме за цими критеріями надалі проводитиметься порівняльний аналіз алгоритмів і оцінювання розробленого програмного інструменту.

Таким чином, перший розділ заклав необхідну теоретичну базу для подальшого дослідження. Отримані результати підтверджують актуальність розробки спеціалізованого інструменту для створення та оптимізації GIF-анімацій і створюють підґрунтя для переходу до практичної частини роботи.

2 РОЗРОБКА ІНСТРУМЕНТУ ДЛЯ ОПТИМІЗАЦІЇ GIF-АНІМАЦІЙ

2.1 Постановка задачі розробки програмного інструменту

Після теоретичного аналізу особливостей формату GIF та основних методів його оптимізації виникла потреба у створенні власного програмного інструменту. Головна мета полягає не просто в конвертації відео в GIF, а надати користувачеві зручний і гнучкий контроль над параметрами, які безпосередньо впливають на якість, розмір і швидкість створення анімації.

На практиці створення та оптимізація GIF-анімацій часто вимагає використання кількох різних програм або онлайн-сервісів. Один інструмент виконує конвертацію, інший відповідає за зміну розміру, а третій за зменшення кількості кольорів тощо. Такий розрізнений підхід забирає багато часу, ускладнює процес і не дає можливості швидко порівняти різні варіанти налаштувань. Саме тому було вирішено розробити єдиний застосунок, який поєднує всі ключові етапи роботи з GIF в одному інтерфейсі.

Інструмент має забезпечувати перетворення відеофайлів популярних форматів (MP4, AVI, MOV, MKV, WEBM та інших) у GIF-анімації. Для цього в програмі використовується потужна бібліотека FFmpeg, яка виконує основну медіаобробку. Сама ж програма виступає зручною графічною оболонкою, що дозволяє користувачеві працювати без командного рядка.

Важливою частиною задачі є реалізація детальних налаштувань оптимізації. Користувач повинен мати можливість самостійно задавати частоту кадрів, розміри (ширину та висоту), кількість кольорів, рівень стиснення, режим циклічного відтворення, а також цільовий розмір файлу. Такий підхід дає змогу гнучко балансувати між якістю зображення, розміром файлу та швидкістю обробки залежно від поточних потреб.

Окремий акцент зроблено на підтримці різних алгоритмів квантування кольору. У програмі реалізовано вибір між Median Cut, Octree, NeuQuant, а також

стандартним методом FFmpeg. Це дозволяє не лише використовувати інструмент у повсякденній роботі, але й проводити порівняльний аналіз ефективності алгоритмів у подальшій експериментальній частині дослідження.

Для зручності використання передбачено сучасні функції: додавання файлів через Drag & Drop, пакетну обробку кількох відео одночасно, попередній перегляд готової анімації, журнал обробки та швидкий доступ до папки з результатами. Інтерфейс виконано в лаконічному темному стилі, щоб нічого не відволікало від основного процесу.

Таким чином, задача розробки полягає у створенні зручного настільного застосунку для Windows, який поєднує конвертацію відео в GIF, розширені можливості оптимізації, вибір алгоритмів квантування та сучасний користувацький інтерфейс. Такий інструмент має практичну та наукову цінність, зокрема як платформа для подальшого дослідження ефективності методів оптимізації GIF-анімацій.

2.2 Визначення функціональних і нефункціональних вимог

Перед початком реалізації програмного інструменту було визначено основні вимоги до його роботи. Це потрібно для того, щоб майбутній застосунок не був набором випадкових функцій, а вирішував конкретну практичну задачу: швидке створення та оптимізацію GIF-анімацій із можливістю керування основними параметрами результату.

Функціональні вимоги визначають, які саме дії має виконувати програма. У рамках цієї роботи основний акцент зроблено на роботі з відеофайлами, гнучкому налаштуванні параметрів оптимізації, виборі алгоритму квантування кольору та зручній взаємодії з користувачем. Програма повинна підтримувати додавання одного або декількох файлів, їх обробку в черзі, створення GIF-анімацій та надання користувачеві готового результату. Основні функціональні вимоги до розробленого програмного інструменту наведено в таблиці 2.1.

Таблиця 2.1 – Функціональні вимоги до програмного інструменту

Вимога	Опис
Додавання файлів	Користувач може додавати відеофайли або GIF-файли через кнопку вибору файлів або за допомогою Drag & Drop.
Підтримка поширених форматів	Програма має працювати з форматами MP4, AVI, MOV, MKV, WEBM, WMV та іншими форматами, які підтримує FFmpeg.
Конвертація відео у GIF	Застосунок повинен створювати GIF-анімацію на основі вибраного відеофайлу.
Пакетна обробка	Користувач може додати декілька файлів і обробити їх послідовно в одній черзі.
Налаштування FPS	Програма має дозволяти змінювати частоту кадрів кінцевої GIF-анімації.
Налаштування розміру кадру	Користувач може задавати ширину та висоту GIF-анімації.
Налаштування кількості кольорів	Програма має дозволяти вибирати кількість кольорів у палітрі в межах, допустимих для GIF.
Вибір алгоритму квантування	Користувач може обрати режим формування палітри: FFmpeg Palette, Median Cut, Octree або NeuQuant.
Налаштування стиснення	Застосунок має підтримувати рівень стиснення, який впливає на параметри повторної оптимізації.
Орієнтація на цільовий розмір	Користувач може вказати бажаний розмір GIF-файлу, до якого програма намагатиметься наблизити результат.
Попередній перегляд	Після обробки користувач має змогу переглянути готову GIF-анімацію безпосередньо в інтерфейсі програми.
Журнал обробки	Програма повинна відображати основні повідомлення про хід обробки, помилки та результати.
Відкриття папки результатів	Користувач може швидко перейти до папки, у якій збережено готові GIF-файли.

Окрім функціональних, були визначені й нефункціональні вимоги. Вони описують не конкретні дії, а загальні характеристики програми: її зручність, швидкодію, стабільність і зрозумілість для користувача. Для даного інструменту це особливо важливо, оскільки він задуманий не як складне професійне

середовище, а як компактна й інтуїтивно зрозуміла програма для швидкого вирішення практичних завдань. Перелік основних нефункціональних вимог до програмного інструменту подано в таблиці 2.2.

Таблиця 2.2 – Нефункціональні вимоги до програмного інструменту

Вимога	Опис
Простота інтерфейсу	Інтерфейс має бути лаконічним, без зайвих елементів, щоб користувач швидко розумів порядок роботи.
Зручність використання	Основні дії повинні виконуватися без складних налаштувань і без необхідності працювати з командним рядком.
Стабільність роботи	Програма має коректно обробляти помилки, зокрема відсутність FFmpeg або некоректний вхідний файл.
Швидкодія	Обробка файлів повинна виконуватися за прийнятний час, особливо при роботі з короткими відеофрагментами.
Підтримка пакетної роботи	Застосунок має залишатися зручним навіть при додаванні декількох файлів у чергу.
Зрозумілість результату	Користувач повинен бачити статус обробки, розмір вхідного файлу та розмір готового GIF.
Візуальна стриманість	Темна чорно-біла тема інтерфейсу має не відволікати від роботи та підкреслювати практичний характер програми.
Можливість подальшого розширення	Структура програми повинна дозволяти надалі додавати нові алгоритми або параметри оптимізації.

Отже, сформовані функціональні та нефункціональні вимоги задають чіткі орієнтири для розробки. Вони забезпечують баланс між необхідною функціональністю (конверсія, оптимізація, вибір алгоритмів) та зручністю використання. Саме ці вимоги стали основою для подальшого проєктування

архітектури програми, вибору технологій та безпосередньої реалізації застосунку.

2.3 Вибір засобів і технологій реалізації

Для створення програмного інструменту було обрано технології, які дозволяють отримати зручний настільний застосунок для Windows і забезпечити надійну роботу з відео різних форматів. Основним стеком розробки стали мова програмування C#, платформа .NET 8, технологія Windows Forms та потужний зовнішній інструмент FFmpeg.

Мову C# обрано через її зручність і швидкість розробки прикладних Windows-програм. Вона має сучасний синтаксис, відмінну підтримку в середовищі розробки та дозволяє ефективно працювати з файловою системою, подіями користувача та зовнішніми процесами. У цій роботі C# використовується переважно для керування логікою програми: додаванням файлів, збереженням налаштувань, запуском обробки, відображенням статусу та результатів.

Для створення інтерфейсу користувача було обрано Windows Forms. Ця технологія, хоч і не найсучасніша, ідеально підходить для компактних утиліт, де основний акцент робиться на функціональності та зручність, а не складна графіка. За її допомогою легко реалізувати списки файлів, налаштування параметрів, Drag & Drop, попередній перегляд і журнал обробки.

Як базову платформу обрано .NET 8. Вона забезпечує стабільну роботу застосунку, сучасні можливості мови C# та зручну структуру проєкту. Крім того, .NET дозволяє добре організувати код, розділивши його на окремі модулі, що полегшує подальше розширення програми.

Для обробки відео ключовим інструментом став FFmpeg. Це потужний кросплатформний інструмент командного рядка, який підтримує практично всі популярні формати відео (MP4, AVI, MOV, MKV, WEBM тощо) та вміє створювати GIF-файли. Замість того, щоб реалізовувати декодування та

кодування відео з нуля, програма формує необхідні команди і передає їх FFmpeg. Такий підхід значно прискорив розробку і підвищив надійність обробки медіа.

Окрім цього, у програмі реалізовано окремий модуль формування палітри кольорів. Він забезпечує підтримку різних алгоритмів квантування: Median Cut, Octree та NeuQuant, а також стандартного методу FFmpeg.

Дизайн інтерфейсу виконано в лаконічному темному стилі. Оскільки програма призначена для роботи з візуальним контентом, було важливо, щоб інтерфейс не відволікав зайвими елементами та яскравими кольорами. Простота і функціональність стали головними пріоритетами.

Таким чином, обраний технологічний стек повністю відповідає поставленим завданням. C# і Windows Forms забезпечують зручний інтерфейс, .NET 8 гарантує стабільність, FFmpeg відповідає за потужну медіаобробку, а власний модуль квантування кольору пов’язує практичну реалізацію з теоретичною частиною дослідження.

2.4 Архітектура розробленого інструменту

Архітектура програмного інструменту побудована за принципом чіткого розділення відповідальності між різними модулями. Такий підхід зробив програму зрозумілішою, зручнішою для підтримки та відкритою для подальшого розширення.

Інтерфейс користувача відповідає за всю взаємодію з людиною. Через нього користувач додає файли, налаштовує параметри, запускає обробку, переглядає результати та слідкує за журналом. Інтерфейс реалізований у головному класі Form1, де зібрано елементи вікна, обробку подій, Drag & Drop та оновлення стану черги.

Модель даних представлена двома основними класами: ConversionJob та ConversionSettings. Клас ConversionJob описує окреме завдання обробки (шлях до файлу, статус, розміри до і після, повідомлення про помилки). Клас

ConversionSettings зберігає всі параметри, які задає користувач: частоту кадрів, розміри, кількість кольорів, алгоритм квантування, dithering тощо.

Сервіс обробки медіафайлів (FfmpegService) винесено в окремий модуль. Він відповідає за формування команд для FFmpeg, запуск процесу, контроль виконання та обробку результатів. Завдяки цьому інтерфейс користувача не «зазнає» деталей роботи з FFmpeg і залишається чистим.

Модуль квантування кольору (PaletteQuantizer) реалізує алгоритми Median Cut, Octree та NeuQuant. Цей модуль є однією з найважливіших частин програми, оскільки дозволяє порівнювати різні підходи до формування палітри. Якщо користувач обирає стандартний режим, палітра створюється засобами FFmpeg.

FFmpeg виступає як основний медіарушій. Він виконує декодування відео, масштабування, зміну частоти кадрів, застосування палітри та фінальне збереження GIF-файлу.

Загальна логіка роботи програми така: користувач додає файли в чергу, налаштовує параметри та запускає обробку. Програма формує налаштування, передає їх у FfmpegService. Залежно від обраного алгоритму квантування, палітра створюється або власним модулем, або FFmpeg. Після цього FFmpeg створює GIF, а програма оновлює статус, розмір файлу та попередній перегляд. Загальну архітектуру розробленого програмного інструменту подано на рисунку 2.1.

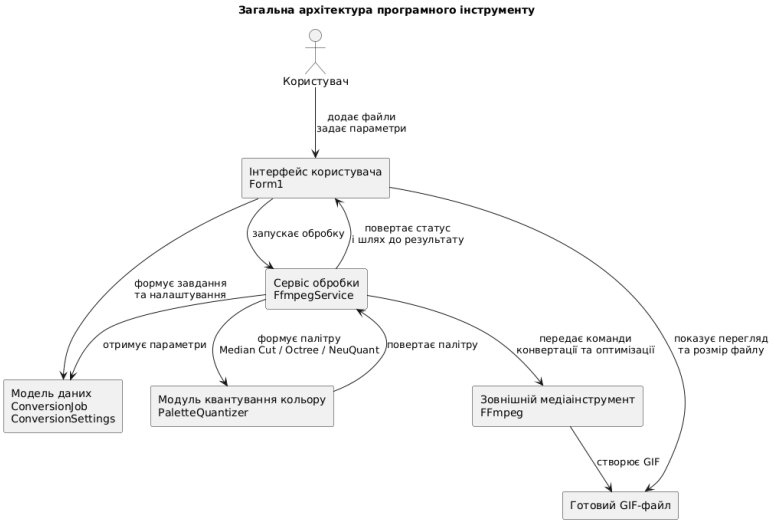


Рисунок 2.1 – Загальна архітектура програмного інструменту

Така архітектура вийшла достатньо простою, але водночас гнучкою. Кожний модуль має чітко визначену відповідальність, тому в майбутньому можна легко додавати нові алгоритми квантування, параметри оптимізації чи навіть нові формати без суттєвої переробки всього застосунку.

2.5 Реалізація модуля обробки кадрів GIF-анімації

Однією з ключових частин програмного інструменту є модуль обробки кадрів. Саме він відповідає за підготовку вихідного відео до перетворення в GIF-анімацію: зчитування відеофайлу, зміну частоти кадрів, масштабування зображення та підготовку матеріалу для подальшого квантування кольору. Вся ця логіка реалізована в сервісі FfmpegService.

Для роботи з відео використовується FFmpeg. Замість того, щоб самостійно реалізовувати декодування відеопотоку, програма формує набір параметрів і фільтрів, які передаються FFmpeg. Такий підхід значно спростив розробку і дозволив зосередитися на логіці оптимізації та зручності інтерфейсу.

Основними параметрами, що впливають на обробку кадрів, є частота кадрів (FPS), ширина та висота майбутньої анімації. Зниження частоти кадрів зменшує розмір файлу, але може зробити рух менш плавним. Зміна розмірів безпосередньо впливає на кількість пікселів у кожному кадрі, а отже й на загальний обсяг даних. Саме тому ці параметри винесені в інтерфейс і доступні для налаштування користувачем.

У сервісі FfmpegService для підготовки кадрів використовується метод, який формує ланцюжок фільтрів FFmpeg:

```
private static string BuildFrameFilters(ConversionSettings settings)
{
    return
$"fps={settings.FramesPerSecond},scale={settings.MaxWidth}:{settings.MaxHeight}:flags=lanczos";
```

```
    }
```

Тут параметр fps відповідає за частоту кадрів, а scale відповідає за масштабування зображення. Прапорець flags=lanczos забезпечує якісніший ресамплинг, щоб при зменшенні роздільної здатності втрачалося якомога менше деталей.

Після підготовки фільтрів сервіс формує повну команду для FFmpeg. Залежно від обраного алгоритму квантування, палітра може створюватися або засобами FFmpeg, або власним модулем. Однак початкова обробка кадрів (зміна FPS і розміру) залишається єдиною для всіх режимів.

Додатково в програмі реалізовано механізм повторних спроб обробки, якщо користувач задав бажаний максимальний розмір файлу. У такому випадку програма може автоматично трохи зменшувати кількість кольорів, FPS або роздільну здатність, щоб наблизитися до цільового розміру:

```
    public ConversionSettings ForAttempt(int attempt)
    {
        var pressure = Math.Clamp((CompressionLevel + attempt * 18) / 100.0, 0.0, 1.0);

        return this with
        {
            Colors = Clamp((int)Math.Round(Colors * (1.0 - pressure * 0.45)), 16, 256),
            FramesPerSecond = Clamp((int)Math.Round(FramesPerSecond * (1.0 - pressure * 0.35)), 5, 60),
            MaxWidth = Clamp((int)Math.Round(MaxWidth * (1.0 - pressure * 0.25)), 120, 3840),
            MaxHeight = Clamp((int)Math.Round(MaxHeight * (1.0 - pressure * 0.25)), 120, 2160)
        };
    }
```

Цей механізм не заміняє повноцінний інтелектуальний аналіз, але суттєво підвищує зручність використання. Користувачеві не доводиться вручну багаторазово підбирати параметри, програма сама виконує кілька ітерацій. Приклад налаштування параметрів кадру в інтерфейсі програми показано на рисунку 2.2.



Рисунок 2.2 – Приклад налаштування параметрів кадру в інтерфейсі програми

Таким чином, модуль обробки кадрів виконує важливий підготовчий етап: приводить відео до потрібних параметрів, готує матеріал для квантування кольору та забезпечує додаткову автоматичну оптимізацію за розміром. Саме він є сполучною ланкою між налаштуваннями користувача та фінальним GIF-файлом.

2.6 Реалізація модуля квантування кольору

Окреме й важливе місце в програмі займає модуль квантування кольору. Саме цей модуль безпосередньо пов’язаний з основною темою дослідження, адже квантування найбільше впливає на візуальну якість і кінцевий розмір GIF-анімації. Якщо зміна FPS чи роздільної здатності зменшує обсяг даних за рахунок кількості кадрів або пікселів, то квантування працює з кольоровою інформацією і вирішує, які саме відтінки залишаться в остаточній палітрі.

Модуль реалізовано в окремому класі PaletteQuantizer. Таке виділення в самостійний компонент дозволяє чітко відокремити логіку формування палітри від інтерфейсу та сервісу FFmpeg. Завдяки цьому алгоритми можна легко

змінювати, доповнювати або порівнювати між собою без переробки інших частин програми.

Перед створенням палітри програма отримує зразки кольорів з відео. Для цього FFmpeg передає кадри у форматі rgb24, після чого програма вибирає частину пікселів як представницьку вибірку. Такий підхід дозволяє не обробляти кожен піксель кожного кадру, але водночас отримувати достатньо точне уявлення про кольорову гаму матеріалу.

Також у програмі передбачено чотири режими формування палітри, а саме базовий режим FFmpeg Palette і власні реалізації алгоритмів Median Cut, Octree та NeuQuant.

Вибір алгоритму здійснюється користувачем безпосередньо в інтерфейсі. Це дає можливість створювати GIF-анімації з однаковими параметрами (FPS, розмір, кількість кольорів), але з різними методами квантування. Надалі саме ця функція використовується для експериментального порівняння в третьому розділі. Фрагмент вибору алгоритму виглядає так:

```
public static IReadOnlyList<Color> BuildPalette(
    IReadOnlyList<RgbColor> samples,
    int colorCount,
    QuantizationAlgorithm algorithm)
{
    if (samples.Count == 0)
    {
        return BuildFallbackPalette(colorCount);
    }

    return algorithm switch
    {
        QuantizationAlgorithm.MedianCut                                    =>
BuildMedianCutPalette(samples, colorCount),
        QuantizationAlgorithm.Octree => BuildOctreePalette(samples,
colorCount),
```

```
            QuantizationAlgorithm.NeuQuant                              =>
BuildNeuQuantPalette(samples, colorCount),
            _ => BuildMedianCutPalette(samples, colorCount)
        };
    }
```

Алгоритм Median Cut реалізовано класичним способом, тобто через послідовне поділення множини кольорів на групи за каналом з найбільшим діапазоном значень. Для кожної фінальної групи обчислюється середній колір.

Octree працює на основі групування кольорів у комірки кольорового простору. Він відносно швидкий і добре показує себе при обробці великої кількості кадрів.

NeuQuant реалізований як спрощена модель самоорганізовуваної нейронної мережі. Набір кольорів «нейронів» поступово адаптується до кольорової структури відео, що часто дозволяє отримати більш природний результат на складних зображеннях із градієнтами.

Після формування палітри вона зберігається у вигляді окремого зображення, яке потім передається FFmpeg для створення GIF-файлу:

```
    public static void SavePalette(IReadOnlyList<Color> colors, string
path)
    {
        using var bitmap = new Bitmap(16, 16);
        var fallbackColor = colors.Count > 0 ? colors[^1] : Color.Black;

        for (var i = 0; i < 256; i++)
        {
            var color = i < colors.Count ? colors[i] : fallbackColor;
            bitmap.SetPixel(i % 16, i / 16, color);
        }

        bitmap.Save(path, ImageFormat.Png);
    }
```

Палітра зберігається як зображення 16×16 пікселів (максимум 256 кольорів). Якщо кольорів менше, вільні комірки заповнюються останнім кольором для коректної структури. Вибір алгоритму квантування кольору в інтерфейсі програми показано на рисунку 2.3.

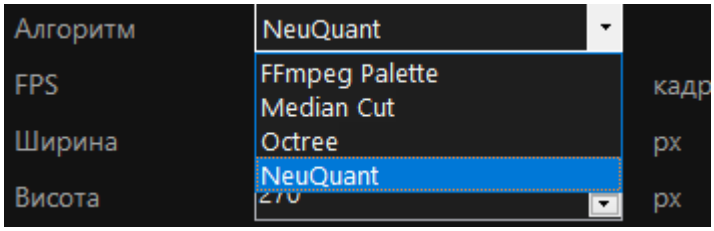


Рисунок 2.3 – Вибір алгоритму квантування кольору в інтерфейсі програми

Отже, модуль квантування кольору є однією з найважливіших частин розробленого інструменту. Він забезпечує практичну реалізацію алгоритмів, розглянутих у теоретичному розділі, і створює надійну основу для подальшого порівняльного аналізу їх ефективності. Завдяки цьому програма може слугувати не тільки практичним інструментом, а й дослідницькою платформою.

2.7 Реалізація модуля оцінювання результатів оптимізації

Після створення GIF-анімації важливо не тільки отримати файл, а й одразу зрозуміти, наскільки вдалою була оптимізація. Для користувача ключовими показниками є розмір початкового файлу, розмір готового GIF, статус обробки та можливість швидко перевірити результат. Саме тому в програмі було реалізовано простий, але ефективний модуль оцінювання результатів.

Оцінювання результату базується на моделі ConversionJob. Для кожного файлу в черзі зберігається вся необхідна інформація: шлях до вихідного файлу, його назва, початковий розмір, поточний статус обробки, шлях до створеного GIF та його розмір. Завдяки цьому програма може чітко відображати стан кожного завдання в таблиці:

```
public sealed class ConversionJob
{
    public string InputPath { get; }

    public string FileName => Path.GetFileName(InputPath);

    public string Status { get; set; } = "Очікує";

    public long OriginalBytes { get; }

    public long OutputBytes { get; set; }

    public string? OutputPath { get; set; }

    public string? ErrorMessage { get; set; }
}
```

Головним кількісним показником є розмір готового GIF-файлу. Після завершення обробки програма автоматично визначає його обсяг і порівнює з початковим розміром та цільовим значенням, якщо воно було задане. У разі перевищення цільового розміру програма може виконати кілька додаткових спроб оптимізації, поступово зменшуючи FPS, кількість кольорів або роздільну здатність. Це дозволяє автоматично наблизити результат до бажаного обсягу без ручного підбору параметрів.

У списку файлів користувач бачить основні дані: назву файлу, статус, розмір вихідного відео та розмір готового GIF. Розміри відображаються в зрозумілому форматі (КБ або МБ), що значно полегшує сприйняття інформації.

Важливою частиною оцінювання є попередній перегляд готової анімації. Після завершення обробки GIF автоматично відображається в інтерфейсі програми. Це дає змогу користувачеві відразу перевірити візуальну якість, наявність спотворень, природність кольорів та плавність руху.

Додатково реалізований журнал обробки, у якому фіксуються ключові події: додавання файлів, вибрані параметри, повідомлення від FFmpeg, спроби повторної оптимізації, успішне завершення або помилки. Журнал корисний як для звичайного користувача, так і для подальшого аналізу результатів.

Варто зазначити, що в межах другого розділу модуль оцінювання реалізований на практичному, прикладному рівні. Програма надає користувачеві розміри файлів, статус, попередній перегляд та журнал. Детальне порівняння алгоритмів за якістю зображення, ступенем стиснення та швидкодію буде проведено в третьому розділі за допомогою спеціальних експериментів і діаграм. Приклад відображення результатів обробки у програмі наведено на рисунку 2.4.



Рисунок 2.4 – Відображення результатів обробки у програмі

Таким чином, модуль оцінювання результатів виконує важливу прикладну функцію, адже він робить процес роботи з програмою прозорим і зрозумілим. Відображення ключових показників, попереднього перегляду та журналу дозволяє користувачеві швидко оцінити ефективність оптимізації та створює надійну основу для подальшого наукового аналізу в експериментальній частині роботи.

2.8 Опис інтерфейсу користувача та сценарію використання програми

Інтерфейс розробленого програмного інструменту створено з акцентом на простоту та швидкість виконання основної задачі. Оскільки програма

призначена для конвертації та оптимізації GIF-анімацій, у ній відсутні зайві декоративні елементи та складна навігація. Всі ключові дії зосереджені в одному головному вікні: додавання файлів, налаштування параметрів, запуск обробки, перегляд результатів та аналіз журналу.

Вікно програми умовно поділено на дві частини. Ліва частина призначена для роботи з файлами. Тут розташована область для Drag & Drop, список доданих файлів та кнопки керування чергою. Користувач може додати відео або GIF як через стандартне діалогове вікно, так і простим перетягуванням файлів у програму. Такий підхід особливо зручний при пакетній обробці кількох файлів. Загальний вигляд головного вікна програмного інструменту наведено на рисунку 2.5.

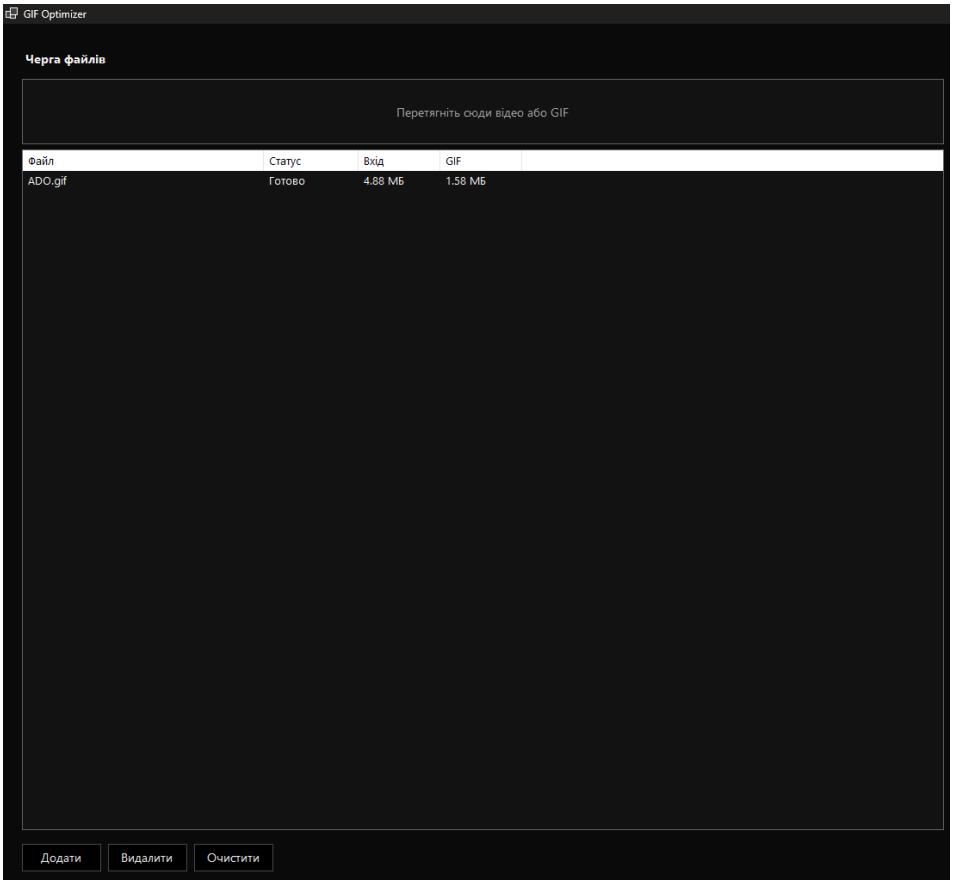


Рисунок 2.5 – Головне вікно програмного інструменту

У списку файлів відображається назва, поточний статус, розмір вихідного файлу та розмір створеного GIF. Статус допомагає зрозуміти, на якому етапі

знаходиться кожне завдання (очікує, в процесі, завершено, скасовано або помилка). Це дуже корисно під час роботи з кількома файлами одночасно.

Права частина містить усі параметри оптимізації. Користувач може вказати шлях до FFmpeg, обрати папку для збереження результатів, вибрати алгоритм квантування кольору, задати частоту кадрів, ширину, висоту, кількість кольорів, рівень стиснення, дизерінг та цільовий розмір файлу. Таке розміщення дозволяє швидко змінювати налаштування і відразу запускати обробку. Налаштування параметрів оптимізації GIF-анімації показано на рисунку 2.6.

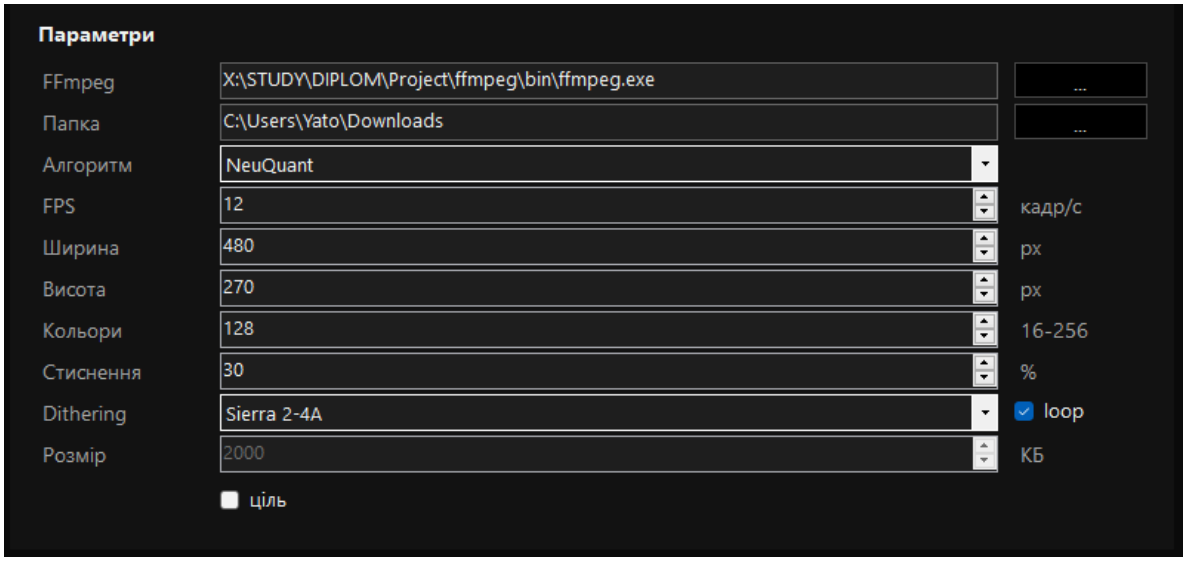


Рисунок 2.6 – Налаштування параметрів оптимізації GIF-анімації

Особливе місце займає блок вибору алгоритму квантування кольору. Доступні варіанти: FFmpeg Palette, Median Cut, Octree та NeuQuant. Це робить програму корисною не лише для повсякденного використання, а й для порівняльних досліджень.

Нижче параметрів розташована область попереднього перегляду. Після завершення обробки готовий GIF відображається безпосередньо в програмі, що дозволяє швидко оцінити візуальну якість, природність кольорів і плавність анімації. Приклад попереднього перегляду готового GIF-файлу наведено на рисунку 2.7.



Рисунок 2.7 – Попередній перегляд готового GIF-файлу

У нижній частині правої панелі розміщено журнал обробки. Він фіксує основні події: додавання файлів, вибрані параметри, повідомлення FFmpeg, спроби повторної оптимізації, успішне завершення або помилки. Журнал допомагає зрозуміти, що саме відбувалося під час обробки, і за потреби скоригувати налаштування.

Типовий сценарій використання програми виглядає так: користувач додає один або кілька файлів у чергу, обирає папку для збереження, задає необхідні параметри (FPS, розмір, кількість кольорів, алгоритм квантування) і запускає обробку. Після завершення він переглядає результат у вбудованому плеєрі, оцінює розмір файлу та, за необхідності, змінює параметри для повторної спроби.

Такий інтерфейс вийшов інтуїтивно зрозумілим і функціональним. Користувач не працює напряму з командним рядком FFmpeg, але при цьому має доступ до всіх важливих параметрів, які впливають на якість і розмір анімації. Це повністю відповідає головній меті розробки, а саме зробити процес створення та оптимізації GIF простішим, швидшим і більш контрольованим.

2.9 Особливості програмної реалізації та обмеження розробленого рішення

Під час реалізації програмного інструменту основна увага приділялася практичності, зрозумілості та зручності використання. Було обрано підхід, за якого вся складна робота з відео покладена на FFmpeg, а власна програма

відповідає за інтерфейс, керування процесом, налаштування параметрів та формування палітри кольорів.

Однією з ключових особливостей реалізації є використання FFmpeg як зовнішнього процесу. Програма лише формує команди, передає їх FFmpeg і обробляє результат. Це дозволило підтримувати велику кількість відеоформатів (MP4, AVI, MOV, MKV, WEBM та інші) без необхідності писати власний код для декодування. Завдяки цьому інструмент став надійнішим і простішим у розробці.

Водночас такий підхід створює певну залежність: для роботи програми на комп’ютері користувача повинен бути присутній файл ffmpeg.exe. У розробленому рішенні передбачено автоматичний пошук FFmpeg у папці програми, у вкладених директоріях tools або ffmpeg/bin, а також у системному PATH. Якщо автоматичний пошук не вдався, користувач може вручну вказати шлях у налаштуваннях.

Іншою важливою особливістю є підтримка пакетної обробки файлів. Кожне завдання зберігається як окремий об’єкт зі своїм статусом, початковим і кінцевим розміром. Обробка виконується послідовно, а не паралельно. Таке рішення обрано свідомо для забезпечення стабільності та зменшення навантаження на систему, адже одночасний запуск кількох процесів FFmpeg може суттєво збільшити споживання процесора та пам’яті.

Реалізація алгоритмів квантування кольору в окремому модулі PaletteQuantizer (Median Cut, Octree, NeuQuant) є важливою відмінністю програми. Це дозволяє порівнювати різні методи в межах одного інструменту. Однак власні реалізації алгоритмів орієнтовані насамперед на навчальні та дослідницькі цілі, тому в деяких складних випадках вони можуть поступатися спеціалізованим професійним бібліотекам за швидкістю або точністю.

Для прискорення роботи при побудові палітри використовується не весь масив пікселів, а вибірка кольорів з відео. Це робить обробку практичною за часом, але може впливати на точність, особливо якщо у відео присутні рідкісні, але візуально важливі відтінки.

Механізм досягнення цільового розміру файлу працює через серію повторних спроб. Якщо GIF перевищує заданий обсяг, програма поступово зменшує FPS, кількість кольорів або роздільну здатність. Цей підхід простий і ефективний у більшості випадків, але не гарантує точного потрапляння в заданий розмір, оскільки кінцевий обсяг залежить не тільки від параметрів, а й від самого змісту відео (рух, деталізація, градієнти).

Попередній перегляд у програмі призначений для швидкої візуальної оцінки, а не для точного професійного аналізу. Детальне порівняння якості, швидкодії та ефективності алгоритмів буде проведено в третьому розділі.

Отже, розроблене рішення поєднує простий і зрозумілий інтерфейс, підтримку популярних відеоформатів, гнучкі параметри оптимізації та власний модуль квантування кольору. Головними перевагами є практичність, прозорість процесу та можливість використання для дослідницьких цілей. Основними обмеженнями залишаються залежність від зовнішнього FFmpeg, послідовна обробка файлів, використання вибірки кольорів та орієнтовний характер автоматичної підгонки під цільовий розмір.

2.10 Висновки до розділу 2

У другому розділі було розглянуто процес проєктування та практичної реалізації програмного інструменту для створення та оптимізації GIF-анімацій. На основі теоретичного фундаменту, закладеного в першому розділі, визначено ключові вимоги до застосунку, його функціональність, архітектуру та технологічний стек.

У результаті розроблено настільний застосунок для Windows, який дозволяє користувачеві зручно додавати відеофайли різних форматів (MP4, AVI, MOV, MKV, WEBM та інші), виконувати їх пакетну обробку, гнучко налаштовувати параметри оптимізації та одразу бачити результат. Програма

використовує потужний інструмент FFmpeg як основний рушій обробки медіа, виступаючи при цьому сучасною та зручною графічною оболонкою.

Особливу увагу приділено параметрам, які найбільше впливають на кінцевий результат: частоті кадрів, роздільній здатності, кількості кольорів, рівню стиснення, використанню методу дизерингу, режиму циклічного відтворення та цільовому розміру файлу. Завдяки цьому користувач отримує реальний контроль над балансом між якістю зображення, плавністю анімації та розміром готового GIF.

Одним із найважливіших елементів розробки став модуль квантування кольору. У ньому реалізовано як базовий режим FFmpeg Palette, так і власні реалізації алгоритмів Median Cut, Octree та NeuQuant. Це не тільки розширює практичні можливості інструменту, але й створює зручну платформу для подальшого наукового порівняння цих алгоритмів.

Інтерфейс програми виконано в лаконічному темному стилі. Він інтуїтивно зрозумілий: черга файлів з підтримкою Drag & Drop, детальна панель параметрів, область попереднього перегляду готової анімації та журнал обробки. Усе це знаходиться в одному вікні, що робить роботу комфортною і не відволікає користувача непотрібними елементами.

Таким чином, у другому розділі вдалося реалізувати повноцінний робочий інструмент, який поєднує зручність використання з необхідною функціональністю для наукового дослідження. Розроблений застосунок повністю готовий до проведення експериментів, які будуть представлені в наступному розділі, де проведено порівняльний аналіз ефективності різних алгоритмів квантування кольору за критеріями розміру файлу, швидкості обробки та візуальної якості результату.

3 АНАЛІЗ ЕФЕКТИВНОСТІ АЛГОРИТМІВ КВАНТУВАННЯ КОЛЬОРУ

3.1 Організація експериментального дослідження

Третій розділ присвячено практичній перевірці розробленого програмного інструменту та порівняльному аналізу ефективності алгоритмів квантування кольору. Якщо перший розділ заклав теоретичні основи, а другий описав процес проєктування та реалізації програми, то тут основна увага зосереджена на реальних результатах її роботи.

Метою експериментального дослідження є з’ясування того, як різні параметри оптимізації та алгоритми квантування кольору впливають на ключові характеристики GIF-анімації: розмір файлу, швидкість обробки та візуальну якість. Для цього було проведено кілька серій тестів із різними налаштуваннями програми.

У дослідженні перевірялися найбільш значущі параметри: кількість кольорів у палітрі, частота кадрів, роздільна здатність, рівень стиснення, а також вибір алгоритму квантування. Особливий акцент зроблено на порівнянні чотирьох підходів: Median Cut, Octree, NeuQuant та базового режиму FFmpeg Palette, оскільки саме вони є центральними для цієї роботи.

Для об’єктивності аналізу використовувалося кілька тестових відеофрагментів, які суттєво відрізняються за характером контенту. Зокрема, були обрані: проста сцена з обмеженою кількістю кольорів і мінімальним рухом, динамічне відео з активним рухом, фрагмент із плавними градієнтами та великою кількістю деталей, а також змішаний тип відео. Такий підбір тестових матеріалів дозволяє оцінити сильні та слабкі сторони алгоритмів у різних реальних умовах, а не лише на одному типовому прикладі.

Під час кожного експерименту фіксувався однаковий набір показників: тип вхідного файлу, вибраний алгоритм квантування, значення FPS, ширина та висота, кількість кольорів, розмір отриманого GIF-файлу, час обробки та

суб’єктивна візуальна оцінка якості. Саме ці дані стали основою для подальшого порівняльного аналізу та побудови діаграм. Оцінювання результатів проводилося за трьома основними напрямами: розмір кінцевого файлу, швидкість обробки і візуальна якість.

Важливою умовою дослідження була максимальна ідентичність умов експерименту. При порівнянні алгоритмів квантування всі інші параметри (FPS, роздільна здатність, кількість кольорів тощо) залишалися незмінними. Це дає можливість оцінити вплив саме алгоритму, а не сторонніх факторів.

Таким чином, експериментальне дослідження побудоване на системному порівнянні результатів роботи програми за різних умов. Отримані дані дозволять не тільки визначити найбільш доцільні налаштування для практичного використання, але й встановити, який алгоритм квантування кольору забезпечує найкращий загальний баланс між розміром файлу, швидкістю обробки та візуальною якістю GIF-анімації.

3.2 Вибір тестових відеофрагментів і параметрів порівняння

Для проведення експериментального дослідження було підготовлено три тестові відеофрагменти у форматі MP4. Використання кількох файлів різного типу є важливим, оскільки ефективність оптимізації GIF значною мірою залежить не тільки від налаштувань програми, а й від особливостей самого відео: кількості руху, рівня деталізації, кольорової насиченості та динаміки зміни кадрів.

Перший файл first_standard.mp4 використовується як базовий приклад для загальної перевірки роботи програми. Другий second_dynamic.mp4 містить активніший рух і призначений для тестування алгоритмів на динамічному контенті. Третій third_colors.mp4 характеризується високою кольоровою насиченістю та плавними переходами, що дозволяє особливо добре оцінити

якість роботи алгоритмів квантування. Основні характеристики обраних тестових відеофрагментів наведено в таблиці 3.1.

Таблиця 3.1 – Характеристики тестових відеофрагментів

Назва файлу	Тип тесту	Роздільна здатність	FPS	Тривалість	Початковий розмір
first_standart.mp4	Стандартний фрагмент	1280×720	30	6,00 с	0,81 МБ
second_dynamic.mp4	Динамічний фрагмент	1920×1080	25	6,00 с	3,51 МБ
third_colors.mp4	Кольорово насичений фрагмент	1280×720	25	6,00 с	1,37 МБ

Для забезпечення об’єктивності порівняння було встановлено однакові базові параметри обробки для всіх тестів. Це дозволяє чітко відстежувати вплив саме алгоритму квантування або окремого параметра, а не випадкових змін інших налаштувань. Базові параметри експериментів: частота кадрів 12 кадрів/с, роздільна здатність 480×270 пікселів, кількість кольорів 128, циклічне відтворення увімкнено, дизеринг Sierra 2-4A.

Надалі ці тестові відеофрагменти використовуватимуться для порівняння роботи алгоритмів FFmpeg Palette, Median Cut, Octree та NeuQuant, а також для оцінки впливу ключових параметрів (кількості кольорів, FPS та роздільної здатності) на розмір і якість GIF-анімацій.

3.3 Порівняльний аналіз розміру GIF-файлів після оптимізації

За результатами першої серії тестування чітко видно, що розмір кінцевого GIF-файлу залежить не тільки від таких параметрів, як частота кадрів, роздільна здатність та кількість кольорів, а й від вибраного алгоритму формування палітри.

Для об’єктивного порівняння в цій серії тестів для всіх алгоритмів використовувалися однакові налаштування: розмір кадру 444×250 пікселів, частота 11 кадрів за секунду та палітра на 111 кольорів.

Найменший розмір файлу в усіх трьох тестових відеофрагментах показав алгоритм Octree. Для стандартного фрагмента розмір GIF становив 384,0 КБ, що виявилося меншим, ніж у FFmpeg Palette та інших алгоритмів. У динамічному відео Octree також лідирував з результатом 2,53 МБ, тоді як інші алгоритми дали файли розміром понад 3,6 МБ. Подібна тенденція спостерігалася і на кольорово насиченому фрагменті, де Octree сформував GIF розміром 2,26 МБ. Порівняння розміру GIF-файлів, отриманих за різними алгоритмами квантування, наведено на рисунку 3.1.

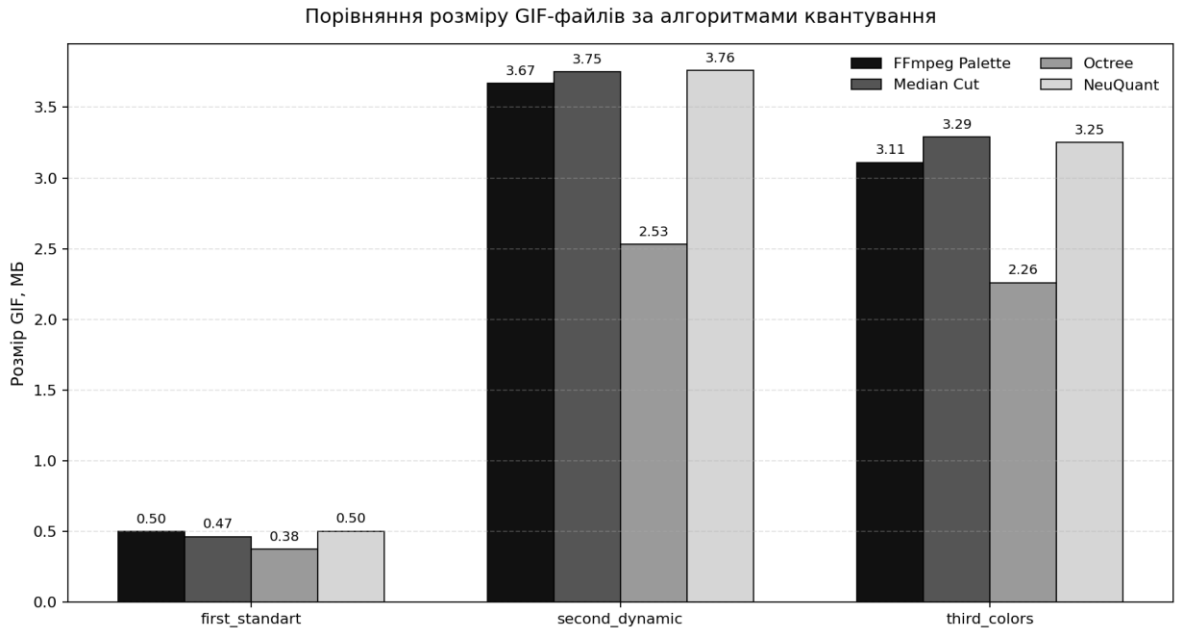


Рисунок 3.1 – Порівняння розміру GIF-файлів за алгоритмами

Алгоритми Median Cut і NeuQuant у цій серії тестів дали більші файли, особливо на динамічному та кольоровому відео. Це не означає, що вони є гіршими загалом, оскільки розмір файлу не є єдиним критерієм оцінювання. Наприклад, NeuQuant може краще зберігати складні кольорові переходи, але через це результат іноді займає більше пам’яті. Саме тому для повної оцінки потрібно також враховувати візуальну якість.

Окремо варто зазначити, що GIF-файл у деяких випадках може бути більшим за початковий MP4-файл. Це пояснюється тим, що MP4 використовує значно ефективніші методи відеостиснення, зокрема міжкадрове стиснення, тоді як GIF є простішим і менш ефективним форматом для збереження динамічного відео. Тому в межах цього дослідження важливо порівнювати не GIF із MP4 напряму, а різні варіанти GIF між собою.

3.4 Аналіз візуальної якості результатів квантування

Після порівняння розмірів GIF-файлів було проведено детальну візуальну оцінку отриманих результатів. Це важливий етап, оскільки значне зменшення розміру файлу не завжди означає прийнятну якість. Іноді алгоритм може сильно спотворювати кольори, контури або плавність переходів, що робить анімацію непридатною для використання. Візуальне порівняння результатів роботи різних алгоритмів квантування подано на рисунку 3.2.

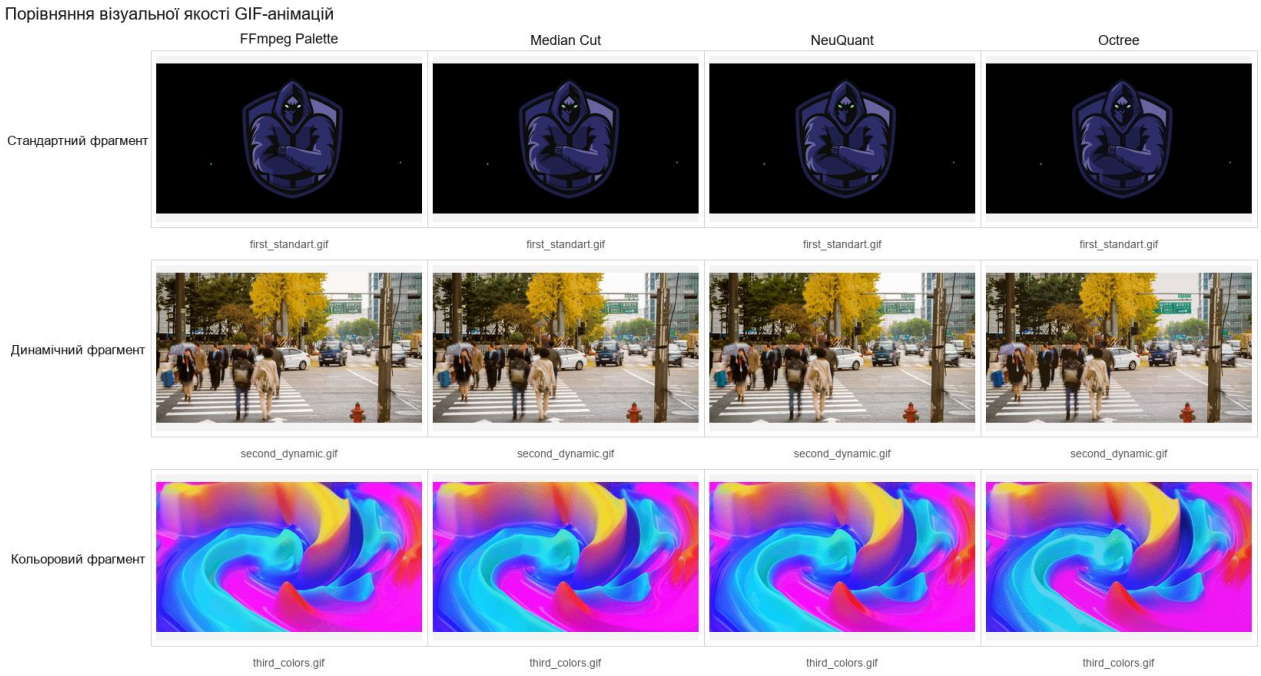


Рисунок 3.2 – Порівняння візуальної якості GIF-анімацій за різними алгоритмами квантування

За результатами перегляду можна зробити такі висновки. Режим FFmpeg Palette забезпечив досить стабільну якість на всіх трьох тестових фрагментах. На стандартному відео з логотипом контури та кольори збереглися добре. У динамічному фрагменті з’явилася помітна зернистість, але загальна структура кадру залишилася зрозумілою. На кольорово насиченому відео характерний дизеринг присутній, проте переходи між відтінками виглядають відносно природно.

Median Cut продемонстрував якість, близьку до базового режиму. На простому фрагменті результат виглядає нормально, основні елементи добре збережені. Однак на динамічному та особливо на кольоровому відео помітна зернистість, насамперед у ділянках із плавними градієнтами та великою кількістю деталей.

NeuQuant візуально дав один із найкращих результатів, особливо на стандартному та динамічному фрагментах. Кольори виглядають природніше, а деталі зберігаються краще, ніж у інших алгоритмів. Головним недоліком залишається більший розмір файлів, тому цей алгоритм доцільно використовувати, коли пріоритетом є якість зображення.

Octree, який показав найменший розмір файлів, водночас має найбільш помітні візуальні недоліки. На стандартному фрагменті спостерігаються спотворення контурів і поява геометричних артефактів. У динамічному відео сильна зернистість, а на кольорово насиченому фрагменті грубі переходи в градієнтах та втрата плавності.

Таким чином, візуальна оцінка підтвердила, що вибір алгоритму квантування не може ґрунтуватися лише на розмірі файлу. Якщо головна мета є максимальне стиснення, то Octree буде ефективним рішенням. Однак якщо важлива якість зображення та природність кольорів, кращими варіантами виглядають NeuQuant, Median Cut або базовий FFmpeg Palette.

Для практичного використання оптимальним підходом є вибір алгоритму залежно від типу контенту. Для простої графіки, іконок, текстових анімацій або

мемів з обмеженою кількістю кольорів можна сміливо застосовувати більш агресивні методи стиснення, такі як Octree, щоб отримати мінімальний розмір файлу. Водночас для динамічних відео, складних градієнтів, реалістичних зображень або контенту з великою кількістю деталей варто віддавати перевагу алгоритмам, які краще зберігають візуальну якість, тому доречніше NeuQuant або FFmpeg Palette.

3.5 Аналіз швидкодії розробленого інструменту

Окрім розміру файлу та візуальної якості, важливим критерієм оцінки є швидкодія обробки. Для практичного інструменту цей показник має велике значення, адже користувач очікує, що створення та оптимізація GIF не займатиме надто багато часу, особливо при роботі з кількома файлами.

Для перевірки швидкодії було проведено обробку всіх трьох тестових відеофрагментів для кожного алгоритму квантування. Умови тестування були однаковими: ті самі файли, однакові параметри (роздільна здатність, FPS, кількість кольорів). Час вимірювався для повної обробки набору з трьох файлів. Щоб зменшити вплив випадкових коливань навантаження системи, результати розглядаються як середній час виконання серії.

Отримані дані дозволяють не лише порівняти алгоритми між собою, але й оцінити практичну придатність програми для реального використання, де швидкість обробки часто є одним із визначальних факторів для користувача. Результати порівняння часу обробки тестового набору наведено на рисунку 3.3.

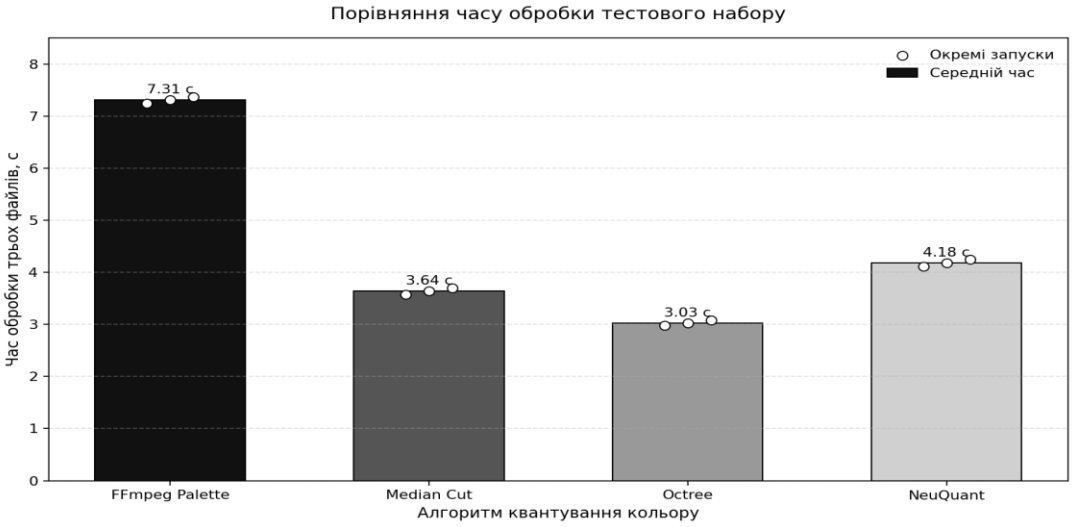


Рисунок 3.3 – Порівняння часу обробки тестового набору за алгоритмами квантування кольору

За результатами тестів найшвидшим виявився алгоритм Octree, для якого обробка трьох файлів зайняла в середньому 3,03 секунди. Це добре узгоджується з попередніми результатами, де Octree також забезпечував найменший розмір GIF-файлів. Така швидкість пояснюється відносно простим принципом групування кольорів.

Median Cut також показав добру швидкодію, із середнім часом обробки 3,64 секунди. При цьому якість результату в багатьох випадках була стабільнішою, ніж у Octree, що робить його досить збалансованим варіантом для практичного застосування.

NeuQuant обробив файли за 4,18 секунди. Він працює помітно повільніше, що очікувано, оскільки базується на ітераційному навчанні нейронної мережі. Однак цей алгоритм часто забезпечує кращу передачу кольорів у складних сценах, тому додатковий час може бути виправданим, коли пріоритетом є якість.

Найбільший час обробки зафіксовано у базовому режимі FFmpeg Palette, який становив 7,31 секунди. Це пов’язано з тим, що FFmpeg виконує генерацію палітри та її застосування у кілька окремих етапів.

Отже, за критерієм швидкодії найкращі результати продемонстрували Octree та Median Cut. Проте, як і в попередніх підрозділах, вибір алгоритму не

повинен ґрунтуватися лише на одному показнику. Octree, хоча й найшвидший і дає найменший файл, може суттєво поступатися якістю зображення. Тому в реальному використанні рекомендується враховувати всі три ключові критерії одночасно: розмір файлу, візуальну якість та швидкість обробки.

3.6 Вибір доцільного алгоритму та рекомендації щодо оптимізації GIF-анімацій

На основі проведених експериментів можна зробити висновок, що жоден із розглянутих алгоритмів квантування кольору не є універсально найкращим для всіх типів контенту. Кожен алгоритм має свої сильні сторони та недоліки, тому вибір найкращого варіанта повинен залежати від пріоритетів у конкретному випадку: мінімальний розмір файлу, швидкість обробки чи висока візуальна якість.

Octree продемонстрував найкращі результати за розміром файлу та швидкодію. У всіх тестових фрагментах він сформував найменші GIF і працював найшвидше. Цей алгоритм доцільно використовувати, коли головною метою є максимальне стиснення і швидка обробка, наприклад, для технічних прев'ю, чорнових анімацій, іконок або випадків, де якість зображення не є критичною.

Водночас візуальна оцінка показала, що Octree часто створює помітні артефакти: спотворення контурів, сильну зернистість і грубі кольорові переходи, особливо на динамічних та кольорово насичених фрагментах. Тому його не варто застосовувати там, де важлива акуратна передача деталей і природність зображення.

Median Cut можна вважати найбільш збалансованим варіантом. Він забезпечує стабільні результати як за розміром, так і за якістю, поступаючись Octree у стисненні, але перевершуючи його за візуальною прийнятністю. Цей алгоритм добре підходить для більшості повсякденних завдань, таких як

створення мемів, інтерфейсних анімацій, коротких демонстрацій продукту, навчальних матеріалів та соціальних мереж.

NeuQuant виявився найкращим за візуальною якістю. Він краще зберігає кольорові переходи, градієнти та загальне сприйняття кадру, хоча створює більші файли і працює повільніше. Цей алгоритм рекомендовано використовувати для презентацій, демонстраційних матеріалів, портфоліо та іншого контенту, де якість зображення має пріоритет над розміром.

Базовий режим FFmpeg Palette показав себе як надійне і стабільне рішення. Він не лідирує за жодним окремим показником, але забезпечує прогнозований і досить збалансований результат для різних типів відео. Його зручно використовувати як стандартний варіант за замовчуванням.

3.7 Висновки до розділу 3

У третьому розділі було проведено експериментальне дослідження розробленого програмного інструменту та порівняльний аналіз ефективності алгоритмів квантування кольору. Для тестування використовувалися три відеофрагменти різного типу: стандартний, динамічний та кольорово насичений. Такий підбір матеріалів дозволив оцінити роботу програми в різних реальних умовах.

У ході експериментів порівнювалися чотири підходи: FFmpeg Palette, Median Cut, Octree та NeuQuant. Тестування проводилося за однакових умов, що дало змогу чітко визначити вплив саме алгоритму квантування на розмір файлу, швидкість обробки та візуальну якість отриманих GIF-анімацій.

За результатами дослідження Octree забезпечив найменший розмір файлів і найвищу швидкодію. Однак візуальна оцінка показала, що цей алгоритм часто створює помітні артефакти, особливо на динамічних сценах і відео з великою кількістю кольорових переходів.

Median Cut і FFmpeg Palette продемонстрували найбільш збалансовані результати. Вони поступаються Octree за розміром і швидкістю, але значно краще зберігають візуальну якість і природність зображення. NeuQuant виявився найкращим за якістю передачі кольорів і градієнтів, хоча й потребує більше часу на обробку та створює файли більшого розміру.

Проведене дослідження підтвердило, що ефективність оптимізації GIF-анімацій неможливо оцінювати за одним критерієм. Орієнтація лише на мінімальний розмір часто призводить до суттєвого погіршення візуальної якості. Тому при виборі параметрів і алгоритму важливо враховувати тип контенту, ступінь динаміки, наявність градієнтів, а також кінцеве призначення анімації.

Таким чином, розроблений програмний інструмент успішно вирішив поставлені завдання. Він дозволяє не тільки швидко створювати GIF-анімації з відеофайлів, але й проводить порівняльний аналіз різних алгоритмів квантування кольору в умовах реального використання. Завдяки цьому користувач може самостійно експериментувати з параметрами та обирати оптимальний підхід для кожного конкретного випадку.

Отримані результати дають чіткі практичні рекомендації щодо вибору алгоритму квантування та налаштувань залежно від типу контенту, пріоритетів (розмір, швидкість чи якість) та кінцевого призначення анімації. Це робить інструмент корисним як для повсякденної роботи, так і для глибшого розуміння процесів оптимізації GIF.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи розглянуто теоретичні основи формату GIF, основні методи його оптимізації, а також розроблено власний програмний інструмент для створення й оптимізації GIF-анімацій із відеофайлів.

В теоретичній частині проаналізовано переваги та обмеження формату GIF. Незважаючи на популярність і широку сумісність, формат має суттєві недоліки: обмеження в 256 кольорів, відсутність звуку та швидке зростання розміру файлу при складній анімації, що підтверджує актуальність теми.

Розроблений настільний застосунок значно спрощує процес створення GIF. Він підтримує додавання файлів через Drag & Drop, пакетну обробку, гнучке налаштування параметрів (FPS, роздільна здатність, кількість кольорів, dithering тощо) та попередній перегляд результату. Для медіаобробки використовується FFmpeg, а для дослідницької частини створено власний модуль квантування кольору з реалізацією алгоритмів Median Cut, Octree та NeuQuant.

Експериментальне дослідження показало, що жоден алгоритм не є універсальним. Octree забезпечує найменший розмір файлів і найвищу швидкість, але часто створює помітні візуальні артефакти. Median Cut і FFmpeg Palette демонструють найкращий баланс, а NeuQuant найкраще підходить для випадків, коли пріоритетом є висока візуальна якість.

Загалом робота завершилася створенням функціонального інструменту, який спрощує повсякденну роботу з GIF-анімаціями. Проведений аналіз підтвердив, що ефективна оптимізація вимагає комплексного підходу та гнучкого вибору алгоритму й параметрів залежно від типу контенту, його динаміки та кінцевого призначення анімації.

ПЕРЕЛІК ПОСИЛАНЬ

1. GIF Graphics Interchange Format. Library of Congress. URL : https://www.loc.gov/preservation/digital/formats/fdd/fdd000133.shtml (дата звернення : 04.05.2026).

2. FFmpeg Documentation. FFmpeg URL : https://ffmpeg.org/ffmpeg.html (дата звернення : 16.05.2026).

3. FFmpeg Filters Documentation. FFmpeg. URL : https://ffmpeg.org/ffmpeg-filters.html (дата звернення : 16.05.2026).

4. FFprobe Documentation. FFmpeg. URL : https://ffmpeg.org/ffprobe.html (дата звернення : 16.05.2026).

5. C# language documentation. Microsoft Learn. URL : https://learn.microsoft.com/dotnet/csharp/ (дата звернення : 16.05.2026).

6. .NET documentation. Microsoft Learn. URL : https://learn.microsoft.com/dotnet/fundamentals/ (дата звернення : 16.05.2026).

7. Windows Forms Overview. Microsoft Learn. URL : https://learn.microsoft.com/dotnet/desktop/winforms/overview/ (дата звернення : 16.05.2026).

8. ProcessStartInfo Class. Microsoft Learn. URL : https://learn.microsoft.com/dotnet/api/system.diagnostics.processstartinfo (дата звернення : 16.05.2026).

9. GifGun 2. aescripts + aeplugins. URL : https://aescripts.com/gifgun/ (дата звернення : 13.05.2026).

10. Median cut color quantization algorithm: retrospective. PubMed. URL : https://pubmed.ncbi.nlm.nih.gov/41841952/ (дата звернення : 04.05.2026).

11. Gervautz M., Purgathofer W. A Simple Method for Color Quantization: Octree Quantization. TU Wien. URL : https://www.cg.tuwien.ac.at/research/publications/1988/purgathofer-1988-simple/ (дата звернення : 10.05.2026).

12. Dekker A. Optimal Colour Quantization using Kohonen Neural Networks. National University of Singapore. URL : https://dl.comp.nus.edu.sg/xmlui/handle/1900.100/1319 (дата звернення : 10.05.2026).

13. Color Quantization and Dithering. ImageMagick Examples. URL : https://usage.imagemagick.org/quantize/ (дата звернення : 10.05.2026).

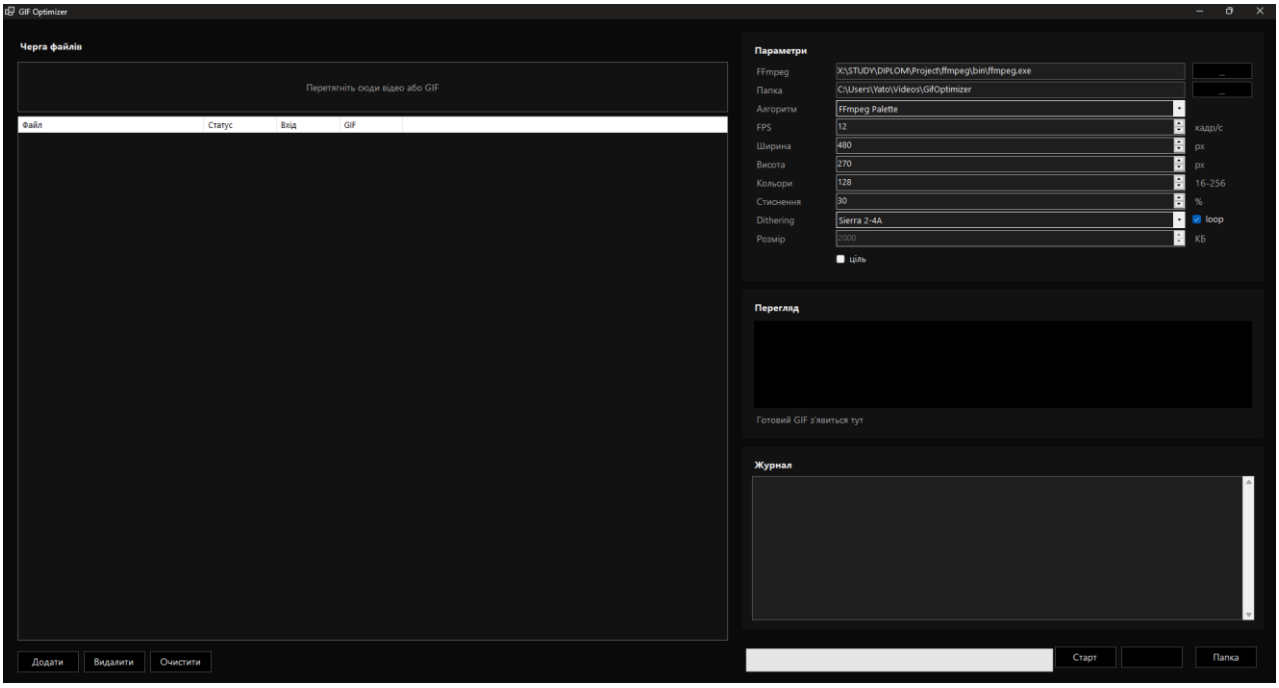
14. Image file formats. Pillow Documentation. URL : https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html (дата звернення : 16.05.2026).

15. Matplotlib.pyplot.bar. Matplotlib Documentation. URL : https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html (дата звернення : 18.05.2026).

16. Component Diagram syntax and features. PlantUML. URL : https://plantuml.com/en/component-diagram (дата звернення : 18.05.2026).

ДОДАТОК А

Інтерфейс розробленого програмного інструменту



ДОДАТОК Б

Фрагмент коду програмної реалізації алгоритмів квантування кольору (PaletteQuantizer.cs)

```
    {
        public ColorBox(List<RgbColor> colors)
        {
            Colors = colors;
            UpdateBounds();
        }
        public List<RgbColor> Colors { get; }
        public int Score => Colors.Count * Math.Max(RangeR,
Math.Max(RangeG, RangeB));
        private int RangeR { get; set; }
        private int RangeG { get; set; }
        private int RangeB { get; set; }
        public (ColorBox Left, ColorBox Right) Split()
        {
            Comparison<RgbColor> comparison = GetSplitChannel() switch
            {
                0 => (left, right) => left.R.CompareTo(right.R),
                1 => (left, right) => left.G.CompareTo(right.G),
                _ => (left, right) => left.B.CompareTo(right.B)
            };
            Colors.Sort(comparison);
            var middle = Math.Max(1, Colors.Count / 2);
            return (
                new ColorBox(Colors.Take(middle).ToList()),
                new ColorBox(Colors.Skip(middle).ToList()));
        }
        public Color GetAverageColor()
        {
            long r = 0;
            long g = 0;
            long b = 0;
            foreach (var color in Colors)
            {
                r += color.R;
                g += color.G;
                b += color.B;
            }
            var count = Math.Max(1, Colors.Count);
            return Color.FromArgb((int)(r / count), (int)(g / count),
(int)(b / count));
        }
```

```
        private int GetSplitChannel()
        {
            if (RangeR >= RangeG && RangeR >= RangeB)
            {
                return 0;
            }
            return RangeG >= RangeB ? 1 : 2;
        }
        private void UpdateBounds()
        {
            var minR = 255;
            var minG = 255;
            var minB = 255;
            var maxR = 0;
            var maxG = 0;
            var maxB = 0;
            foreach (var color in Colors)
            {
                minR = Math.Min(minR, color.R);
                minG = Math.Min(minG, color.G);
                minB = Math.Min(minB, color.B);
                maxR = Math.Max(maxR, color.R);
                maxG = Math.Max(maxG, color.G);
                maxB = Math.Max(maxB, color.B);
            }
            RangeR = maxR - minR;
            RangeG = maxG - minG;
            RangeB = maxB - minB;
        }
    }
    private sealed class Bucket
    {
        private long _r;
        private long _g;
        private long _b;
        public int Count { get; private set; }
        public void Add(RgbColor color)
        {
            _r += color.R;
            _g += color.G;
            _b += color.B;
            Count++;
        }
        public Color GetAverageColor()
        {
            var count = Math.Max(1, Count);
            return Color.FromArgb((int)(_r / count), (int)(_g / count),
(int)(_b / count));
        }
    }
    private sealed class NeuralColor
```

```
    {
        private double _r;
        private double _g;
        private double _b;
        public NeuralColor(byte r, byte g, byte b)
        {
            _r = r;
            _g = g;
            _b = b;
        }
        public double DistanceTo(RgbColor color)
        {
            var dr = _r - color.R;
            var dg = _g - color.G;
            var db = _b - color.B;
            return dr * dr + dg * dg + db * db;
        }
        public void MoveTowards(RgbColor color, double amount)
        {
            _r += (color.R - _r) * amount;
            _g += (color.G - _g) * amount;
            _b += (color.B - _b) * amount;
        }
        public Color ToColor()
        {
            return Color.FromArgb(Clamp(_r), Clamp(_g), Clamp(_b));
        }
        private static int Clamp(double value)
        {
            return Math.Min(255, Math.Max(0, (int)Math.Round(value)));
        }
    }
}
```

ДОДАТОК В

Фрагмент коду програмної реалізації модуля обробки відеофайлів та створення GIF-анімацій (FfmpegService.cs)

```
using System.Diagnostics;
using Project.Models;

namespace Project.Services;

public sealed class FfmpegService
{
    private const int MaxTargetAttempts = 5;
    private const int MaxSampleFrames = 90;
    private const int MaxSamplesPerFrame = 1400;

    public static string? FindDefaultFfmpeg()
    {
        var candidates = new[]
        {
            Path.Combine(AppContext.BaseDirectory, "ffmpeg.exe"),
            Path.Combine(AppContext.BaseDirectory, "tools",
"ffmpeg.exe"),
            Path.Combine(AppContext.BaseDirectory, "ffmpeg", "bin",
"ffmpeg.exe"),
            Path.Combine(Environment.CurrentDirectory, "ffmpeg.exe"),
            Path.Combine(Environment.CurrentDirectory, "tools",
"ffmpeg.exe"),
            Path.Combine(Environment.CurrentDirectory, "ffmpeg", "bin",
"ffmpeg.exe")
        };

        foreach (var candidate in candidates)
        {
            if (File.Exists(candidate))
            {
                return candidate;
            }
        }

        var path = Environment.GetEnvironmentVariable("PATH");
        if (string.IsNullOrWhiteSpace(path))
        {
            return null;
        }
```

```
        foreach (var directory in path.Split(Path.PathSeparator,
StringSplitOptions.RemoveEmptyEntries))
        {
            var candidate = Path.Combine(directory.Trim(), "ffmpeg.exe");
            if (File.Exists(candidate))
            {
                return candidate;
            }
        }

        return null;
    }

    public async Task ConvertAsync(
        ConversionJob job,
        ConversionSettings baseSettings,
        IProgress<string> progress,
        CancellationToken cancellationToken)
    {
        if (!File.Exists(baseSettings.FfmpegPath))
        {
            throw new FileNotFoundException("FFmpeg не знайдено. Вкажіть
шлях до ffmpeg.exe.", baseSettings.FfmpegPath);
        }

        Directory.CreateDirectory(baseSettings.OutputDirectory);

        var finalOutput =
CreateUniqueOutputPath(baseSettings.OutputDirectory, job.InputPath);
        var tempDirectory = Path.Combine(Path.GetTempPath(),
"GifOptimizer", Guid.NewGuid().ToString("N"));
        Directory.CreateDirectory(tempDirectory);

        string? selectedAttempt = null;

        try
        {
            for (var attempt = 0; attempt < MaxTargetAttempts; attempt++)
            {
                cancellationToken.ThrowIfCancellationRequested();

                var settings = baseSettings.ForAttempt(attempt);
                var attemptOutput = Path.Combine(tempDirectory,
$"attempt_{attempt + 1}.gif");
                var palettePath = Path.Combine(tempDirectory,
$"palette_{attempt + 1}.png");

                progress.Report($"[{job.FileName}] спроба {attempt + 1}:
{settings.MaxWidth}x{settings.MaxHeight}, {settings.FramesPerSecond} FPS,
{settings.Colors} кольорів,
{GetAlgorithmTitle(settings.QuantizationAlgorithm)}");
```

```
                await RenderGifAsync(job.InputPath, attemptOutput,
palettePath, settings, progress, cancellationToken);

                var outputSize = new FileInfo(attemptOutput).Length;
                selectedAttempt = attemptOutput;

                if (!baseSettings.UseTargetSize || outputSize <=
baseSettings.TargetSizeKb * 1024L)
                {
                    break;
                }

                progress.Report($"[{job.FileName}] розмір
{FormatBytes(outputSize)} перевищує ціль {baseSettings.TargetSizeKb}
КБ");
            }

            if (selectedAttempt is null)
            {
                throw new InvalidOperationException("Не вдалося створити
GIF-файл.");
            }

            File.Copy(selectedAttempt, finalOutput);
```