Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка програмної системи для апскейлінгу відеопотоку з використанням методів глибокого навчання та часової узгодженості кадрів

Виконав: студент IV курсу, групи СП-41
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Духній В.І.
(підпис) (прізвище та ініціали)

Керівник _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Духній Владислав Іванович

1. Тема роботи Розробка програмної системи для апскейлінгу відеопотоку з використанням методів глибокого навчання та часової узгодженості кадрів

Керівник роботи Стоянов Юрій Миколайович к.т.н., доцент кафедри ПІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом по університету від «6» квітня 2026 року № 4/9-169

2. Термін подання студентом роботи 22.06.2026

3. Вихідні дані до роботи *Технічне завдання*

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз предметної області та вимог до програмної системи.
2 Проєктування та розробка програмної системи.
3 Тестування, впровадження та підтримка програмної системи.
4 Безпека життєдіяльності, основи охорони праці.
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Слайди презентації

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання 6.04.2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*6.04 – 12.04*	
2.	*Робота над першим розділом «*Аналіз предметної області та вимог до програмної системи*»*	*13.04 – 26.04*	
3.	*Робота над другим розділом «*Проєктування та розробка програмної системи*»*	*27.04 – 03.05*	
4.	*Робота над третім розділом «*Тестування, впровадження та підтримка програмної системи*»*	*04.05 – 17.05*	
	Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»	*18.05 – 24.05*	
5.	*Оформлення пояснювальної записки і графічного матеріалу*	*25.05 – 07.06*	
6.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*08.06 – 14.06*	
7.	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*	
8.	*Захист кваліфікаційної роботи бакалавра*		
9.			
10.			
11.			
12.			

Студент _____ Духній В.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Духній В. І. Розробка програмної системи для апскейлінгу відеопотоку з використанням методів глибокого навчання та часової узгодженості кадрів : робота на здобуття кваліфікаційного ступеня бакалавра : спец. 121 – інженерія програмного забезпечення / наук. кер. Ю. М. Стоянов. Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2026. 67 с. // С. – 67, табл. – 8, рис. – 12, бібліогр. – 25, слайд. – __, додат. – 2.

Ключові слова: інженерія програмного забезпечення, глибоке навчання, супер роздільність, апскейлінг відео, генеративно-змагальна мережа, Real-ESRGAN, часова узгодженість кадрів, оптичний потік, відновлення облич, обробка зображень.

Метою роботи є проєктування та розробка програмної системи, яка підвищує роздільну здатність зображень і відеопотоку методами глибокого навчання та зменшує міжкадрове мерехтіння за рахунок забезпечення часової узгодженості кадрів.

У першому розділі проаналізовано предметну область підвищення роздільної здатності, розглянуто еволюцію методів глибокого навчання для суперроздільності та проблему часової узгодженості у відео, виконано огляд програмних аналогів, визначено акторів і варіанти використання, сформульовано вимоги до системи та критерії успішності.

У другому розділі обґрунтовано вибір процесу й засобів розробки, спроєктовано архітектуру системи та місце в ній компонентів глибокого навчання, описано структури даних, об'єктно-орієнтовану модель і веб-інтерфейс взаємодії.

У третьому розділі описано реалізацію та інтеграцію переднавчених моделей, реалізацію модуля часової узгодженості на основі оптичного потоку, наведено результати функціонального й навантажувального тестування, оцінювання якості за метриками та розгортання системи.

У четвертому розділі розглянуто питання безпеки життєдіяльності та основ охорони праці, пов'язані з робочим місцем розробника програмного забезпечення.

Об'єктом дослідження є процес підвищення роздільної здатності зображень та відеопотоку засобами глибокого навчання.

Предметом дослідження є методи, моделі та програмні засоби апскейлінгу відео на основі генеративно-змагальних мереж із забезпеченням часової узгодженості кадрів. Методи дослідження: аналіз програмних аналогів, проєктування архітектури програмного забезпечення, методи глибокого навчання для суперроздільності, оцінювання оптичного потоку для часової стабілізації, експериментальне оцінювання якості за метриками PSNR, SSIM і LPIPS, модульне та інтеграційне тестування.

ABSTRACT

Vladyslav Dukhnii. Development of a software system for video stream upscaling using deep learning methods and temporal frame consistency : bachelor thesis : specialty 121 "Software Engineering" / supervisor Yu. M. Stoianov. Ternopil : Ternopil Ivan Puluj National Technical University, 2026. 67 p. // Pages – 67, tables – 8, figures – 12, references – 25, presentation slides – __, appendices – 2.

Keywords: software engineering, deep learning, super-resolution, video upscaling, generative adversarial network, Real-ESRGAN, temporal frame consistency, optical flow, face restoration, image processing.

The purpose of the work is to design and develop a software system that increases the resolution of images and video streams with deep learning methods and reduces inter-frame flicker by enforcing temporal frame consistency.

The first chapter analyses the problem domain of resolution enhancement, reviews the evolution of deep learning methods for super-resolution and the temporal consistency problem in video, surveys software analogues, defines actors and use cases, and states the system requirements and success criteria.

The second chapter justifies the development process and toolset, designs the system architecture and the place of the deep learning components within it, and describes the data structures, the object-oriented model and the web interaction interface.

The third chapter describes the implementation and integration of the pretrained models, the optical-flow-based temporal consistency module, the results of functional and load testing, the quality evaluation against metrics, and the system deployment.

The fourth chapter addresses life safety and occupational health issues related to a software developer's workplace.

The object of the study is the process of increasing the resolution of images and video streams by deep learning methods.

The subject of the study is the methods, models and software tools for video upscaling based on generative adversarial networks with temporal frame consistency.

Research methods: analysis of software analogues, software architecture design, deep learning methods for super-resolution, optical flow estimation for temporal stabilisation, experimental quality evaluation using the PSNR, SSIM and LPIPS metrics, unit and integration testing.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ГН – глибоке навчання.

ЗГ – згорткова нейронна мережа.

ШІ – штучний інтелект.

API – програмний інтерфейс застосунку (Application Programming Interface).

CLAHE – адаптивне вирівнювання гістограми з обмеженням контрасту (Contrast Limited Adaptive Histogram Equalization).

FP16 – формат напівточних чисел з рухомою комою.

FPS – кількість кадрів за секунду (frames per second).

GAN – генеративно-змагальна мережа (Generative Adversarial Network).

GPU – графічний процесор (Graphics Processing Unit).

GUI – графічний інтерфейс користувача (Graphical User Interface).

LPIPS – навчена метрика перцептивної подібності (Learned Perceptual Image Patch Similarity).

PSNR – пікове відношення сигналу до шуму (Peak Signal-to-Noise Ratio).

RRDBNet – мережа з залишкових щільних блоків (Residual-in-Residual Dense Block Network).

SR – суперроздільність (Super-Resolution).

SSIM – індекс структурної подібності (Structural Similarity Index Measure).

VRAM – відеопам'ять графічного процесора.

VSR – суперроздільність відео (Video Super-Resolution).

ЗМІСТ

ВСТУП

Обсяги відеоконтенту, що створюється, передається й архівується, стрімко зростають, а вимоги до його візуальної якості постійно підвищуються. Водночас значна частина матеріалів — оцифровані архіви, записи з камер невисокої роздільної здатності, стиснені відеопотоки — має недостатню роздільну здатність або містить спотворення. Задача підвищення роздільної здатності (апскейлінгу) є актуальною для відновлення таких матеріалів, адаптації контенту під сучасні екрани високої роздільної здатності та покращення сприйняття відео.

Класичні методи масштабування на основі інтерполяції не здатні відтворити втрачені деталі, тому в останнє десятиліття провідними стали методи глибокого навчання, зокрема генеративно-змагальні мережі, які навчаються правдоподібно відновлювати високочастотні структури. Проте під час застосування таких методів до відео постає окрема проблема: незалежна покадрова обробка призводить до того, що відновлені деталі відрізняються від кадру до кадру, через що у відтворюваному відео з'являється мерехтіння. Це явище — порушення часової узгодженості кадрів — суттєво знижує якість результату й вимагає окремих засобів усунення.

Аналіз наявних рішень показує, що потужні комерційні системи апскейлінгу відео є закритими й платними, а доступні відкриті інструменти здебільшого обробляють відео суто покадрово, не забезпечуючи часової узгодженості, і мають складні для непідготовленого користувача інтерфейси. Це обумовлює актуальність розробки доступної програмної системи, яка поєднує сучасні методи глибокого навчання для апскейлінгу із забезпеченням часової узгодженості кадрів і простим інтерфейсом користувача.

Метою кваліфікаційної роботи є проєктування та розробка програмної системи для підвищення роздільної здатності зображень і відеопотоку з використанням методів глибокого навчання та забезпеченням часової узгодженості кадрів.

Для досягнення поставленої мети необхідно виконати такі задачі:

- Проаналізувати предметну область підвищення роздільної здатності, методи глибокого навчання для суперроздільності та проблему часової узгодженості кадрів, а також виконати огляд наявних програмних рішень.
- Сформулювати вимоги до системи й критерії її успішності, визначити акторів і варіанти використання.
- Обґрунтувати вибір моделей, алгоритмів, мови та засобів розробки;
- спроєктувати архітектуру системи й місце в ній компонентів глибокого навчання.
- Реалізувати конвеєр обробки зображень і відео, інтегрувати переднавчені моделі та розробити модуль часової узгодженості кадрів.
- Виконати тестування системи й оцінити якість відновлення та ефективність модуля часової узгодженості за об'єктивними метриками.
- Підготувати засоби розгортання системи та інструкцію користувача.

Об'єктом дослідження є процес підвищення роздільної здатності зображень та відеопотоку засобами глибокого навчання.

Предметом дослідження є методи, моделі та програмні засоби апскейлінгу відео на основі генеративно-змагальних мереж із забезпеченням часової узгодженості кадрів.

Методи дослідження: аналіз та порівняння програмних аналогів; методи проєктування архітектури програмного забезпечення та об'єктно-орієнтованого моделювання; методи глибокого навчання для суперроздільності зображень; оцінювання щільного оптичного потоку для часової стабілізації; експериментальне оцінювання якості за метриками PSNR, SSIM і LPIPS та похибкою перенесення між кадрами; модульне й функціональне тестування.

Практичне значення отриманих результатів полягає в тому, що розроблена система є відкритим і доступним інструментом апскейлінгу зображень і відео, який автоматично адаптується до наявного апаратного забезпечення (GPU NVIDIA, Apple Silicon або CPU), надає простий вебінтерфейс і, на відміну від поширених відкритих аналогів, забезпечує часову узгодженість кадрів відео.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

У цьому розділі окреслено предметну область підвищення роздільної здатності зображень і відео, розглянуто перехід від класичних методів інтерполяції до методів глибокого навчання та проблему часової узгодженості кадрів. Виконано огляд наявних програмних рішень, визначено акторів і варіанти використання, проаналізовано дані, з якими працює система, та сформульовано функціональні і нефункціональні вимоги разом із критеріями успішності.

1.1 Аналіз предметної області

Підвищення роздільної здатності (апскейлінг, англ. super-resolution) — це задача відновлення зображення високої роздільної здатності з одного або кількох зображень низької роздільної здатності. Задача є некоректно поставленою (ill-posed): одному входу низької роздільної здатності відповідає безліч правдоподібних виходів, оскільки під час зменшення зображення частина високочастотної інформації втрачається безповоротно. Тому якісний апскейлінг полягає не у простому розтягуванні пікселів, а у правдоподібному відновленні втрачених деталей [8].

Класичні методи масштабування — інтерполяція за найближчим сусідом, білінійна, бікубічна та фільтр Ланцоша — оцінюють значення нового пікселя як зважену комбінацію сусідніх. Вони обчислювально дешеві й передбачувані, але не здатні відтворити деталі, яких немає у вхідних даних, через що результат виглядає розмитим, а на контурах з'являються артефакти. Це обмеження стало причиною переходу до методів глибокого навчання, які навчаються відновлювати високочастотні структури на великих наборах зображень.

Першим впливовим підходом на основі згорткових нейронних мереж стала модель SRCNN, що безпосередньо навчала відображення зображення низької роздільної здатності у зображення високої роздільної здатності [8]. Подальший

розвиток пов'язаний із застосуванням генеративно-змагальних мереж: модель SRGAN запровадила перцептивну функцію втрат і змагальне навчання, завдяки чому результат став візуально різкішим і реалістичнішим, навіть якщо формальні піксельні метрики при цьому дещо знижувалися [8]. Архітектура ESRGAN удосконалила цей напрям, замінивши базовий блок на залишковий щільний блок (RRDB) без нормалізації за батчем, що підвищило стабільність навчання та деталізацію [1].

Окремою практичною проблемою є те, що моделі, навчені на штучно зменшених зображеннях, погано працюють із реальними матеріалами, які містять шум стиснення, розмиття та інші складні спотворення. Її розв'язує модель Real-ESRGAN, яка використовує модель деградації високого порядку (послідовне накладання розмиття, шуму, передискретизації та артефактів стиснення) для синтезу навчальних пар, наближених до реальних умов [2], [9]. Саме Real-ESRGAN із архітектурою генератора RRDBNet покладено в основу розроблюваної системи. Для специфічної задачі відновлення облич, які особливо чутливі до спотворень, застосовується модель GFPGAN, що використовує генеративний пріор обличчя для реалістичного відновлення рис [3]. Поряд із підходами на основі згорткових мереж розвиваються і трансформерні архітектури, зокрема SwinIR, які демонструють високу якість відновлення зображень [4]; вони згадані тут для повноти огляду, але виходять за межі обраного для системи інструментарію.

Окремий клас задач становить підвищення роздільної здатності відео. Найпростіший підхід полягає в покадровій обробці, коли кожен кадр незалежно проходить через модель суперроздільності зображень. Він простий у реалізації та дозволяє повторно використати наявні моделі для зображень, проте має суттєву ваду: оскільки модель відновлює високочастотні деталі для кожного кадру окремо, відновлені структури від кадру до кадру дещо різняться, що у відтворюваному відео сприймається як мерехтіння та «кипіння» текстур. Це явище називають порушенням часової узгодженості кадрів.

Для подолання цієї проблеми застосовують спеціалізовані моделі

суперроздільності відео, які враховують інформацію із сусідніх кадрів: EDVR використовує деформовані згортки для вирівнювання кадрів [10], а сімейство BasicVSR і BasicVSR++ — двонапрямлене поширення ознак та вирівнювання за оптичним потоком [5], [6]. Альтернативний напрям — забезпечення часової узгодженості як окремого етапу постобробки, що не потребує перенавчання основної моделі. Такі методи беруть уже оброблені кадри, оцінюють рух між сусідніми кадрами за допомогою оптичного потоку, переносять (warping) попередній результат у систему координат поточного кадру та змішують їх у ділянках, де оцінка руху є надійною [7], [10]; споріднений підхід використовує самонавчання для забезпечення часової когерентності у генеративних моделях відео [7]. Саме такий принцип — часова стабілізація на основі оптичного потоку як постобробка над покадровим апскейлінгом — реалізовано в розроблюваній системі, оскільки він поєднує помірну обчислювальну вартість із можливістю використати потужні переднавчені моделі для зображень.

Для оцінювання якості відновлення застосовують як класичні метрики, що порівнюють відновлене зображення з еталоном — пікове відношення сигналу до шуму (PSNR) та індекс структурної подібності (SSIM) [11], — так і навчену перцептивну метрику LPIPS, яка краще корелює зі сприйняттям людини [13]. Для відео додатково оцінюють часову узгодженість через похибку перенесення (warping error) між сусідніми кадрами. Оцінку руху між кадрами зручно виконувати алгоритмом щільного оптичного потоку Фарнебека [12].

1.2 Огляд існуючих рішень

На ринку та у відкритих репозиторіях існує низка рішень для підвищення роздільної здатності зображень і відео, які різняться ступенем відкритості, наявністю підтримки відео, способом апскейлінгу та апаратними вимогами.

Topaz Video AI (нова назва — Topaz Video) — професійний комерційний застосунок для покращення відео, що використовує набір спеціалізованих моделей глибокого навчання для підвищення роздільної здатності до 4K/8K,

шумозаглушення, стабілізації, інтерполяції кадрів та відновлення облич [15]. Рішення забезпечує високу якість, але є закритим, потребує платної передплати та потужного апаратного забезпечення, що обмежує його доступність для навчальних і дослідницьких цілей.

Video2x — відкритий інструмент, який, по суті, є оркестратором: він розбиває відео на кадри, передає кожен кадр обраному рушію апскейлінгу (зокрема Real-ESRGAN, Real-CUGAN, Anime4K), після чого збирає кадри назад у відео засобами FFmpeg і зберігає звукову доріжку [16]. Video2x підтримує тайлування для обробки кадрів великої роздільної здатності, проте його інтерфейс орієнтований радше на досвідчених користувачів, не має попереднього перегляду, а сама обробка є покадровою й не передбачає спеціального механізму часової узгодженості.

Серед інших відкритих рішень поширені waifu2x та Upscayl, орієнтовані переважно на зображення (зокрема анімаційні), а також безпосереднє використання Real-ESRGAN із командного рядка. Окремо стоять технології апскейлінгу реального часу для ігор (на кшталт DLSS і FSR), які працюють в інтерактивному режимі з доступом до додаткових даних рендерингу (векторів руху, буфера глибини) і тому не застосовні до довільного відеопотоку.

Порівняння розглянутих рішень за ключовими характеристиками наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння аналогів системи апскейлінгу відео

	Topaz Video AI	**Video2x**	**waifu2x / Upscayl**	**Розроблювана система**
1	2	3	4	5
Відкритий код	ні	так	так	так
Підтримка відео	так	так	ні / обмежено	так (MP4)
Метод апскейлінгу	власні ГН-моделі	Real-ESRGAN та ін.	ГН-моделі	Real-ESRGAN (RRDBNet)
Відновлення облич	так	ні	ні	так (GFPGAN)

Продовження таблиці 1.1

1	2	3	4	5
Часова узгодженість	покадрово	покадрово	—	оптичний потік (постобробка)
Інтерфейс	десктоп GUI	GUI / CLI	GUI / CLI	веб-інтерфейс (браузер)
Адаптація апаратури	GPU	GPU	GPU	CUDA / Apple MPS / CPU
Вартість	платно (передплата)	безкоштовно	безкоштовно	безкоштовно

Проведений огляд показує, що потужні комерційні рішення є закритими й платними, а доступні відкриті інструменти або зосереджені на зображеннях, або обробляють відео суто покадрово без забезпечення часової узгодженості, а їхні інтерфейси складні для непідготовленого користувача. Відтак існує ніша для відкритого рішення, яке поєднує апскейлінг на основі Real-ESRGAN, відновлення облич GFPGAN та конвеєр постобробки з окремим модулем часової узгодженості кадрів, надає простий веб-інтерфейс і автоматично адаптується до доступного апаратного забезпечення (CUDA, Apple MPS або CPU). Саме таку систему розроблено в межах кваліфікаційної роботи.

1.3 Аналіз та підготовка даних

Особливістю системи є те, що вона не передбачає навчання власної моделі з нуля, а інтегрує переднавчені моделі Real-ESRGAN та GFPGAN у режимі виведення (inference). Тому аналіз даних доцільно розглядати у трьох аспектах: дані, на яких навчені використовувані моделі; вхідні дані, які обробляє система; та дані для оцінювання якості.

Використовувані ваги Real-ESRGAN навчені із застосуванням моделі деградації високого порядку, яка синтезує пари «низька–висока роздільність» шляхом послідовного накладання випадкового розмиття, шуму, передискретизації та артефактів стиснення JPEG [2]. Такий спосіб формування навчальних даних

дозволяє моделі узагальнюватися на реальні матеріали, що містять складні спотворення, без потреби у спеціально зібраному датасеті. Цілісність завантажуваних ваг у системі контролюється перевіркою розміру файлу та контрольної суми SHA256, що гарантує відповідність ваг очікуваним моделям.

Вхідними даними системи є растрові зображення у форматах PNG, JPEG, WEBP і BMP та відеофайли у форматі MP4. Кожен кадр відео під час обробки перетворюється на зображення й проходить через той самий конвеєр, що й окреме зображення; для відео додатково застосовується модуль часової узгодженості. Розмір вхідних даних обмежено налаштовуваним порогом за кількістю мегапікселів, а зображення та кадри понад поріг внутрішнього розміру обробляються частинами (тайлами) для контролю використання відеопам'яті.

Для кількісного оцінювання якості формується перевірочний набір: з еталонних зображень високої роздільної здатності бікубічним зменшенням створюються входи низької роздільної здатності, які потім відновлюються системою та порівнюються з еталоном за метриками PSNR, SSIM і LPIPS. Для відео як показник часової узгодженості використовується похибка перенесення між сусідніми кадрами, обчислена за оптичним потоком. Такий підхід дозволяє отримати об'єктивні числові показники без потреби в анотованих даних.

1.4 Визначення акторів та варіантів використання

У системі визначено одного актора — користувача, у ролі якого може виступати контент-мейкер, дослідник або фахівець, що працює зі збереженням і відновленням візуальних матеріалів. Користувач взаємодіє із системою через веб-інтерфейс і не потребує спеціальних знань про внутрішню будову моделей.

Основні варіанти використання системи такі:

- Пакетне завантаження та апскейлінг набору зображень.
- Апскейлінг відеофайлу у форматі MP4 зі збереженням звукової доріжки.
- Вибір режиму масштабування (2x, 4x, 8x, а також спеціальні режими

для анімаційного контенту) та пресету продуктивності/якості.

- Увімкнення відновлення облич для матеріалів із людьми.
- Налаштування конвеєра постобробки (корекція кольору, усунення серпанку, прибирання артефактів стиснення, підкреслення контурів тощо).
- Керування часовою узгодженістю кадрів для відео (увімкнення та регулювання сили стабілізації).
- Перегляд і завантаження результатів обробки.
- Моніторинг ходу обробки та стану системи (доступність графічного процесора, обсяг пам'яті, швидкодія).

Узагальнену діаграму варіантів використання наведено на рисунку 1.1.

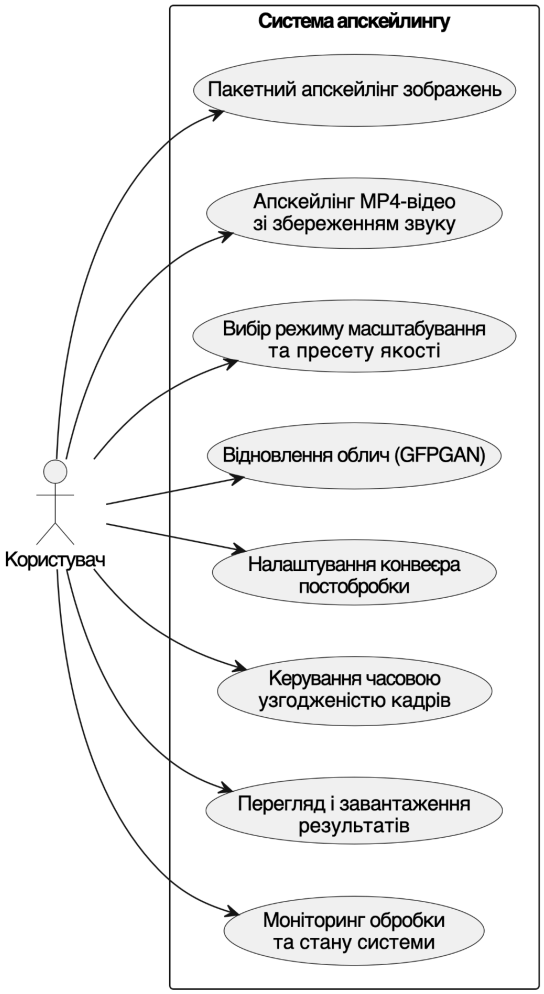

Рисунок 1.1 – Діаграма варіантів використання системи

1.5 Постановка завдання та вимоги до системи

Метою роботи є проєктування й розробка програмної системи апскейлінгу зображень і відеопотоку методами глибокого навчання із забезпеченням часової узгодженості кадрів. Для досягнення мети поставлено такі задачі: проаналізувати предметну область і наявні рішення; обґрунтувати вибір моделей, алгоритмів та засобів розробки; спроєктувати архітектуру системи; реалізувати конвеєр обробки зображень і відео та модуль часової узгодженості; виконати тестування й оцінювання якості; підготувати засоби розгортання.

До системи висунуто такі функціональні вимоги:

- Система повинна підвищувати роздільну здатність завантажених зображень із використанням переднавчених моделей глибокого навчання.
- Система повинна підтримувати кілька режимів масштабування, зокрема для звичайного та анімаційного контенту, а також багатопрохідний режим для великих коефіцієнтів збільшення.
- Система повинна обробляти відеофайли у форматі MP4 покадрово та формувати вихідне відео у форматі H.264/MP4 зі збереженням звуку.
- Система повинна забезпечувати часову узгодженість кадрів відео для зменшення міжкадрового мерехтіння.
- Система повинна підтримувати відновлення облич та налаштовуваний конвеєр постобробки.
- Система повинна відображати хід обробки, попередній перегляд результатів і надавати можливість їх завантаження.

До системи висунуто такі нефункціональні вимоги:

- Продуктивність: система повинна автоматично визначати доступне апаратне забезпечення й увімкнути пришвидшення (зокрема обчислення з напівточністю на сумісних графічних процесорах).
- Портативність: система повинна працювати на конфігураціях із графічним процесором NVIDIA (CUDA), на Apple Silicon (MPS) та у режимі лише центрального процесора.

- Керованість пам'яттю: обробка великих зображень і кадрів повинна виконуватися частинами (тайлами) для уникнення вичерпання відеопам'яті.
- Зручність: взаємодія повинна відбуватися через простий веб-інтерфейс без потреби в інсталяції додаткового програмного забезпечення на боці користувача.
- Надійність: збій окремого фільтра або відсутність звукової доріжки у відео не повинні переривати обробку.

Критеріями успішності системи визначено: перевищення показників якості PSNR, SSIM і LPIPS порівняно з бікубічним масштабуванням на перевірочному наборі; зниження похибки часової узгодженості (warping error) при ввімкненому модулі стабілізації порівняно з покадровою обробкою; прийнятну пропускну здатність обробки за наявності графічного процесора; коректну роботу на всіх трьох цільових типах апаратного забезпечення.

1.6 Висновки до першого розділу

У першому розділі проаналізовано предметну область підвищення роздільної здатності зображень і відео, показано обмеження класичних методів інтерполяції та обґрунтовано перехід до методів глибокого навчання, зокрема генеративно-змагальних мереж ESRGAN і Real-ESRGAN та моделі відновлення облич GFPGAN. Окремо розглянуто проблему часової узгодженості кадрів при покадровому апскейлінгу відео та підходи до її розв'язання, серед яких обрано часову стабілізацію на основі оптичного потоку як етап постобробки.

Виконано огляд наявних рішень, який показав, що потужні комерційні застосунки є закритими й платними, а доступні відкриті інструменти переважно обробляють відео суто покадрово без забезпечення часової узгодженості. Це обґрунтовує доцільність розробки відкритої системи, що поєднує апскейлінг на основі Real-ESRGAN, відновлення облич, конвеєр постобробки та модуль часової узгодженості з простим веб-інтерфейсом і автоматичною адаптацією до апаратного забезпечення. Визначено акторів і варіанти використання,

проаналізовано дані, з якими працює система, та сформульовано функціональні й нефункціональні вимоги разом із критеріями успішності, що є підґрунтям для проєктування системи в наступному розділі.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

У цьому розділі обґрунтовано вибір процесу розробки, спроєктовано архітектуру системи та визначено місце в ній компонентів глибокого навчання. Описано структури даних, об'єктно-орієнтовану модель, обґрунтовано вибір мови та засобів розробки, наведено реалізацію основних класів і методів, а також розробку інтерфейсу користувача.

2.1 Вибір процесу розробки

З огляду на дослідницький характер задачі обрано ітеративно-інкрементну модель розробки. На відміну від каскадної моделі, вона не вимагає повного й незмінного переліку вимог на початку, що відповідає специфіці роботи: низку рішень — зокрема спосіб забезпечення часової узгодженості кадрів, евристики адаптації до апаратного забезпечення та параметри конвеєра постобробки — уточнено експериментально під час розробки. Розробку організовано за принципом мінімально життєздатного продукту (MVP): спершу реалізовано базову функціональність апскейлінгу зображень, далі — обробку відео, після чого додано модуль часової узгодженості та засоби оцінювання якості.

Для забезпечення керованості й відтворюваності застосовано низку інженерних практик, рекомендованих стандартом SWEBOK [21]: ведення коду в системі контролю версій Git, автоматизоване модульне тестування з використанням pytest та безперервну інтеграцію (CI) на платформі GitHub Actions, що запускає тести й перевірку конфігурації при кожному внесенні змін, контейнеризацію за допомогою Docker для відтворюваного розгортання, а також виділення ключових етапів (попередній захист, захист) в окремі релізи. Якість коду підтримується статичним аналізатором ruff та форматувальником black.

Такий процес дозволив поетапно нарощувати функціональність, зберігаючи працездатність системи на кожній ітерації та контролюючи якість через автоматичні перевірки.

2.2 Проєктування архітектури системи

Систему спроєктовано за багаторівневим (шаруватим) принципом із чітким розділенням відповідальностей, що спрощує тестування й супровід. Загальну архітектуру наведено на рисунку 2.1.

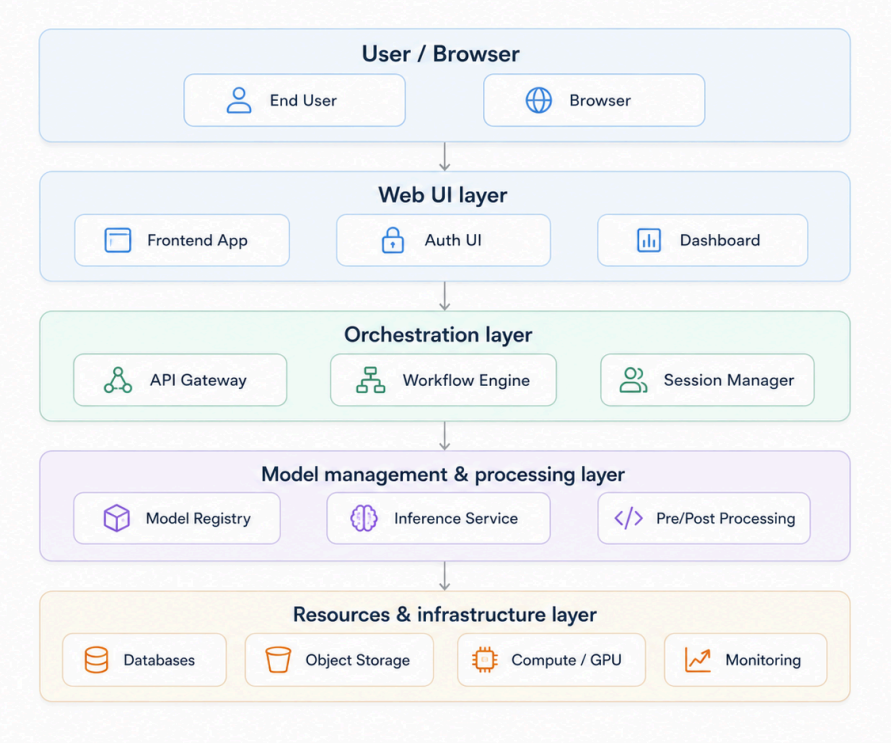

Рисунок 2.1 – Загальна архітектура програмної системи

Виокремлено такі рівні:

- Рівень інтерфейсу — веб-застосунок на Streamlit (streamlit_app.py), який відповідає за завантаження файлів, налаштування параметрів, відображення прогресу та видачу результатів.
- Рівень оркестрації — клас CoreProcessor, що розбирає обраний режим, керує батчуванням і послідовністю кроків обробки для зображень і відео.
- Рівень обробки — конвеєр фільтрів FilterPipeline, оптимізатор пам'яті MemoryOptimizedProcessor (тайлування) та модуль часової узгодженості TemporalStabilizer.
- Рівень моделей — менеджер моделей ModelManager, детектор

апаратних можливостей GPUCapabilities та обгортки моделей RealESRGANModel і GFPGANModel.

- Рівень відео-вводу/виводу — VideoReader і VideoWriter.
- Рівень ваг моделей — утиліта download_weights.py з перевіркою цілісності за SHA256.

Ключовим архітектурним рішенням є інкапсуляція компонентів глибокого навчання за фасадом ModelManager. Завантаження ваг моделей є дорогою операцією, тому ModelManager реалізовано як одинак (singleton) із кешем моделей та витісненням за принципом «найдавніше використаний» (LRU): це дозволяє повторно використовувати вже завантажені моделі між запитами й уникати повторного зчитування ваг. Завдяки такому розділенню верхні рівні не залежать від конкретних деталей моделей, а заміна або розширення набору моделей не зачіпає логіку оркестрації та інтерфейсу.

Потік даних відрізняється для зображень і відео. Для зображень CoreProcessor обробляє вхідні файли пакетами; великі зображення скеровуються на шлях із тайлуванням. Для відео кадри лениво зчитуються з VideoReader, проходять той самий конвеєр обробки, що й окремі зображення, додатково стабілізуються модулем часової узгодженості й послідовно записуються у VideoWriter, який наприкінці домішує звукову доріжку.

Додатковою вимогою до архітектури є ізоляція веб-інтерфейсу від обчислювального ядра. Streamlit у цій системі використовується лише як шар взаємодії з користувачем: він збирає параметри, передає їх у ProcessingConfig і відображає результат. Бізнес-логіка не розміщується в callback-ах інтерфейсу, тому CoreProcessor можна тестувати без браузера, без Streamlit runtime і без фактичного завантаження файлів через форму. Такий поділ зменшує зв'язність модулів і спрощує подальшу заміну інтерфейсу на інший API, наприклад REST-сервіс або пакетну консольну утиліту.

Під час проєктування враховано, що обробка медіафайлів є ресурсомісткою операцією й може завершитися помилкою для окремого файлу без критичного збою всього застосунку. Тому кожний запуск виконується у тимчасовому каталозі,

а помилки накопичуються у списку CoreProcessor.errors і виводяться користувачеві після завершення. Для відео результат спочатку записується у приватний тимчасовий файл, а у фінальне розташування переміщується лише після успішного закриття VideoWriter. Це захищає користувача від отримання пошкодженого MP4 у разі скасування, помилки FFmpeg або нестачі ресурсів.

Окремий контур керування ресурсами реалізовано через обмеження max_megapixels, вибір розміру тайлів і пакетну обробку кадрів. На системах із CUDA розмір батчу та тайла може бути більшим, а на CPU або Apple MPS застосовуються консервативніші значення. Такий підхід потрібний тому, що однаковий режим масштабування по-різному навантажує пам'ять залежно від розміру вхідного зображення, коефіцієнта збільшення та кількості додаткових фільтрів. Архітектура не покладається на статичне припущення про доступну відеопам'ять, а підлаштовує параметри перед запуском інференсу.

Потік даних навмисно побудовано без довготривалого сховища. Усі вхідні файли існують лише в межах поточного сеансу обробки, а результат одразу повертається через інтерфейс завантаження. Для дипломного проєкту це важливо з двох причин: по-перше, система не обробляє облікові записи й не потребує механізмів авторизації; по-друге, відсутність бази даних зменшує ризик накопичення великих медіафайлів на сервері. Якщо в майбутньому буде потрібна історія запусків, її доцільно додавати окремим сервісним шаром, не змінюючи CoreProcessor.

2.3 Проєктування структур даних

Система не використовує базу даних: вона не зберігає облікові записи, історію чи інші дані між сеансами. Обробка є відстановою (stateless) — завантажені файли розміщуються у тимчасовому каталозі сеансу, обробляються та віддаються користувачеві, після чого тимчасові дані видаляються. Таке рішення обґрунтоване відсутністю вимоги до персистентності й спрощує розгортання та супровід. Тому замість схеми бази даних доцільно описати структури даних, що

використовуються у пам'яті.

Центральною структурою є конфігурація одного запуску обробки ProcessingConfig — клас даних (dataclass), що агрегує всі параметри: перелік вхідних файлів, обраний режим масштабування, прапорці фільтрів і відновлення облич, рівні насиченості й контрасту, формат і якість виводу, обмеження за мегапікселями, тип медіа та параметри часової узгодженості. Скорочено структуру наведено в лістингу 2.1.

Лістинг 2.1 – Фрагмент структури конфігурації обробки

```
@dataclass
class ProcessingConfig:
    input_paths: List[str] = field(default_factory=list)
    selected_mode: str = "No Upscale"
    enhance_faces: bool = True
    media_type: str = "image"
    max_megapixels: int = 64
    temporal_consistency: bool = True
    temporal_strength: float = 0.5
```

Режими масштабування описано незмінним відображенням SCALE_MODE_CONFIGS, що зіставляє назву режиму зі структурою ScaleModeConfig. Остання визначає коефіцієнт першого проходу, ознаку анімаційного режиму та, для режимів x8, параметри другого проходу (подвійний прохід 2x => 4x). Такий підхід дозволяє декларативно задати всі режими без розгалуженої умовної логіки.

Завантажені моделі зберігаються у кеші ModelManager — упорядкованому словнику (OrderedDict), ключем у якому є рядок, складений з імені класу моделі та її параметрів (коефіцієнт, анімаційний режим, напівточність). Це забезпечує коректне розрізнення варіантів моделей і витіснення за LRU. Метадані відео представлено незмінним класом даних VideoMetadata (ширина, висота, частота кадрів, кількість кадрів, тривалість), а параметри запису — VideoWriterConfig. Налаштування апаратного забезпечення передаються у вигляді словника (напівточність, розмір тайла, відступ тайла, TF32, channels-last, розмір батчу).

Між етапами обробки дані передаються у трьох поданнях: зображення

PIL.Image на рівні конвеєра, масиви numpy для операцій OpenCV та тензори torch під час інференсу моделей. Перетворення між поданнями локалізовані у відповідних класах.

2.4 Побудова UML-діаграми класів

Об'єктно-орієнтовану модель системи наведено на рисунках 2.2–2.3. Вони відображають основні класи та зв'язки між ними.

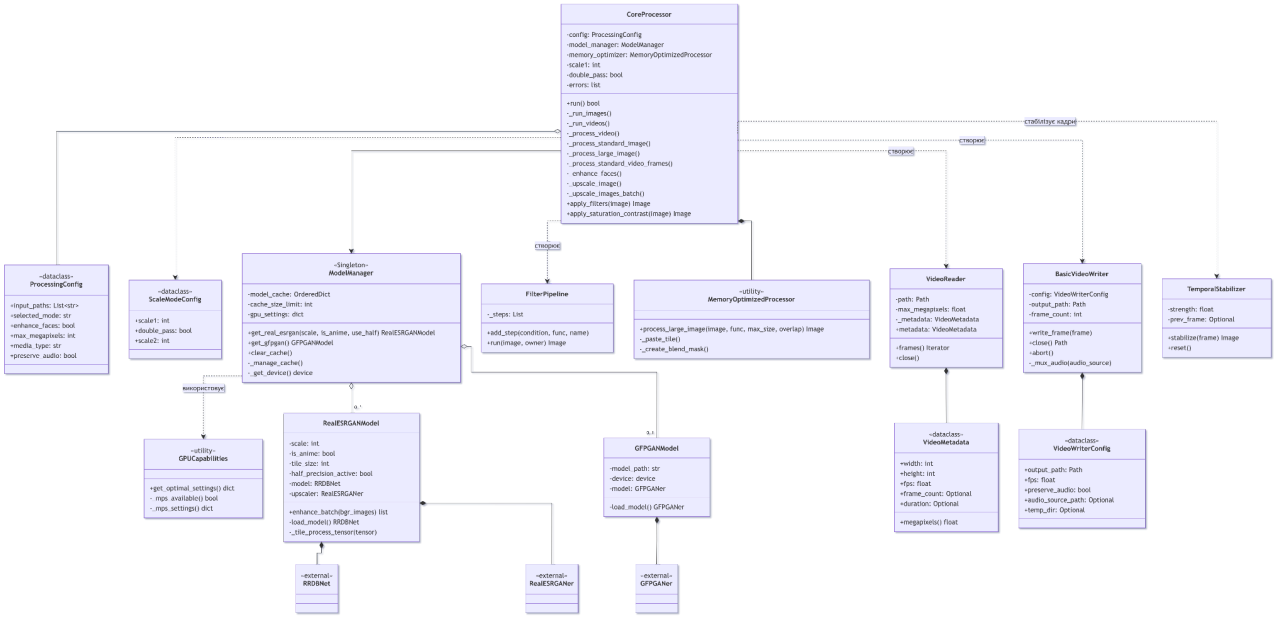

Рисунок 2.2 – UML-діаграма класів системи

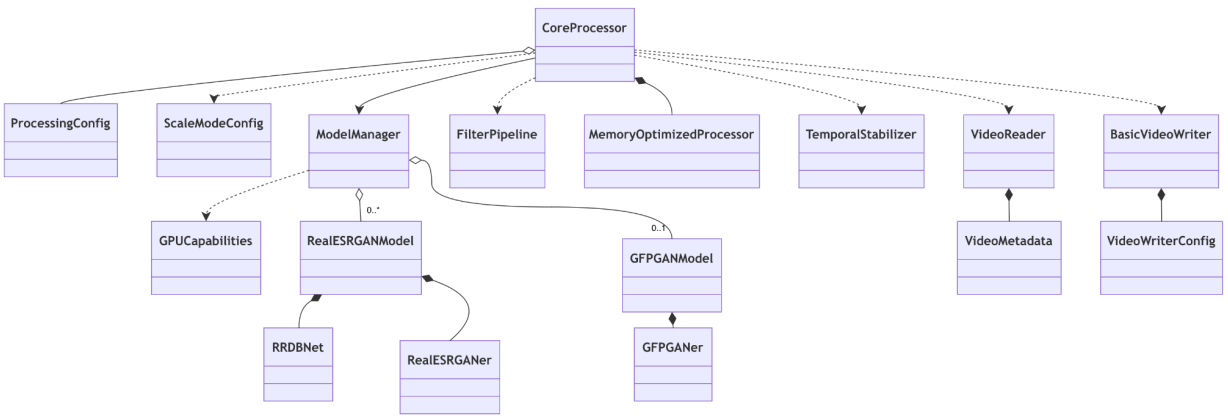

Рисунок 2.3 – UML-діаграма основних класів системи

Клас CoreProcessor є центром оркестрації: він агрегує об'єкт конфігурації ProcessingConfig, використовує ModelManager для отримання та кешування моделей, FilterPipeline — для побудови конвеєра фільтрів постобробки, MemoryOptimizedProcessor — для тайлового оброблення завеликих зображень, а також VideoReader і VideoWriter — для покадрового зчитування й запису відео. Клас ModelManager, реалізований за патерном «Одинак» (Singleton), створює та кешує об'єкти RealESRGANModel і GFPGANModel, спираючись на оптимальні параметри, визначені класом GPUCapabilities. Обгортки моделей інкапсулюють відповідні нейронні мережі: RealESRGANModel — генератор RRDBNet (через обгортку RealESRGANer), а GFPGANModel — модель відновлення облич GFPGANer. Кожен кадр відео обробляється тим самим конвеєром, що й окреме зображення. Такий розподіл відповідальностей відповідає принципу єдиного обов'язку (SRP) та спрощує модульне тестування кожного класу окремо.

2.5 Вибір мови та середовища розробки

Мовою розробки обрано Python 3.11 — стандарт для задач машинного навчання й комп'ютерного зору завдяки зрілій екосистемі бібліотек. Основою для роботи з моделями глибокого навчання є фреймворк PyTorch 2.2 [17], який забезпечує єдиний інтерфейс до різних обчислювальних backend-ів — CUDA для відеокарт NVIDIA, MPS для Apple Silicon і CPU — та підтримує обчислення з напівточністю (FP16), що дозволяє системі адаптуватися до різного апаратного забезпечення без зміни коду.

Безпосередньо моделі суперроздільності й відновлення облич підключено через спеціалізовані пакети: basicsr 1.4.2 (реалізація архітектури генератора RRDBNet), realesrgan 0.3.0 (клас RealESRGANer з підтримкою тайлованого інференсу) [20], gfpgan 1.3.8 (клас GFPGANer) [3] та facexlib 0.3.0 (допоміжне виявлення й вирівнювання облич). Для класичної обробки зображень і обчислення оптичного потоку використано OpenCV [12], для введення-виведення зображень — Pillow, для метрик якості — scikit-image [11], для числових операцій — NumPy

і SciPy.

Веб-інтерфейс реалізовано засобами Streamlit [18], що дає змогу побудувати інтерактивний застосунок мовою Python без окремої фронтенд-частини й значно прискорює розробку. Обробку відео забезпечують imageio з плагіном ffmpeg та FFmpeg [19]: декодування кадрів вхідного MP4 і кодування результату у форматі H.264 з домішуванням звукової доріжки. Для відтворюваного розгортання на GPU-хостах підготовлено образ Docker на базі офіційного образу PyTorch із CUDA [20].

Окремо варто відзначити технічну деталь сумісності: новіші версії torchvision вилучили модуль functional_tensor, на який спирається basicsr, тому в коді реалізовано легкий програмний місток, що відновлює потрібний шлях імпорту. Це типовий приклад інженерного компромісу під час інтеграції зовнішніх бібліотек різних версій.

Середовищем розробки слугувала IDE з підтримкою Python; для перевірки роботи на GPU застосовувалися хмарні середовища з відеокартами. Допоміжний інструментарій включає pytest (тестування), ruff і black (статичний аналіз і форматування).

2.6 Реалізація основних класів та методів

Менеджер моделей ModelManager реалізовано як потокобезпечний одинак: створення єдиного екземпляра захищене блокуванням, а ініціалізація виконується одноразово. Під час ініціалізації менеджер отримує оптимальні налаштування від GPUCapabilities та вмикає відповідні оптимізації PyTorch (TF32 і cudnn.benchmark на сумісних відеокартах). Скелет реалізації одинака наведено в лістингу 2.2.

Лістинг 2.2 – Реалізація одинака ModelManager (фрагмент)

```
class ModelManager:
    _instance = None
    _lock = threading.Lock()

    def __new__(cls, *args, **kwargs):
```

```
        if not cls._instance:
            with cls._lock:
                if not cls._instance:
                    cls._instance = super().__new__(cls)
        return cls._instance
```

Видача моделей відбувається через методи get_real_esrgan і get_gfpgan: за наявності моделі в кеші вона повертається з оновленням позиції в LRU, інакше — створюється нова, а за потреби найдавніша модель витісняється з вивільненням пам'яті відеокарти.

Детектор GPUCapabilities визначає оптимальні параметри обробки залежно від наявного обладнання. За відсутності CUDA перевіряється доступність Apple MPS, інакше застосовуються безпечні налаштування для CPU. За наявності CUDA параметри (увімкнення напівточності, розмір і відступ тайла, TF32, channels-last, розмір батчу) підбираються евристично за поколінням і обсягом пам'яті відеокарти — для архітектур Turing, Ampere та Ada Lovelace передбачено окремі набори значень. Логіку підбору схематично показано в лістингу 2.3.

Лістинг 2.3 – Евристика підбору налаштувань під відеокарту (фрагмент)

```
if any(x in gpu_name for x in ['rtx 40', 'ada', 'l40']):
    return {'use_half': True,
            'tile_size': 1536 if vram_gb >= 12 else 1024,
            'tile_pad': 32, 'enable_tf32': True,
            'enable_channels_last': True,
            'batch_size': 8 if vram_gb >= 24 else 4}
```

Обгортка RealESRGANModel завантажує генератор RRDBNet із параметрами, що відповідають обраному режиму (для режимів x2 і x4 — 23 залишкові блоки, для анімаційного — 6), і конфігурує клас RealESRGANer із тайлуванням. Напівточність вмикається лише на відеокартах із обчислювальною спроможністю не нижче 7.0, інакше система автоматично переходить на повну точність, щоб уникнути артефактів. Метод enhance_batch обробляє кілька однакових за розміром кадрів за один виклик моделі, що підвищує продуктивність на відео, а метод _tile_process_tensor реалізує потайлову обробку для контролю

використання відеопам'яті. Обгортка GFPGANModel ініціалізує GFPGANer із «чистою» архітектурою для відновлення облич без додаткового масштабування фону.

Клас CoreProcessor реалізує наскрізну логіку обробки. Метод run обирає гілку — зображення чи відео. Для зображень файли обробляються пакетами; зображення, що перевищують поріг внутрішнього розміру, скеровуються на шлях MemoryOptimizedProcessor. Для кожного зображення послідовно застосовуються відновлення облич (за потреби), один або два проходи апскейлінгу (для режиму x8) та конвеєр фільтрів, після чого відновлюється альфа-канал. Обробку відео ілюструє діаграма послідовності на рисунку 2.4: після зчитування метаданих кадри читаються лениво, обробляються батчами (відновлення облич, апскейлінг, фільтри, часова стабілізація) і записуються у вихідний файл; підтримуються індикація прогресу з оцінкою часу, що залишився, скасування обробки та ізоляція помилок окремих файлів.

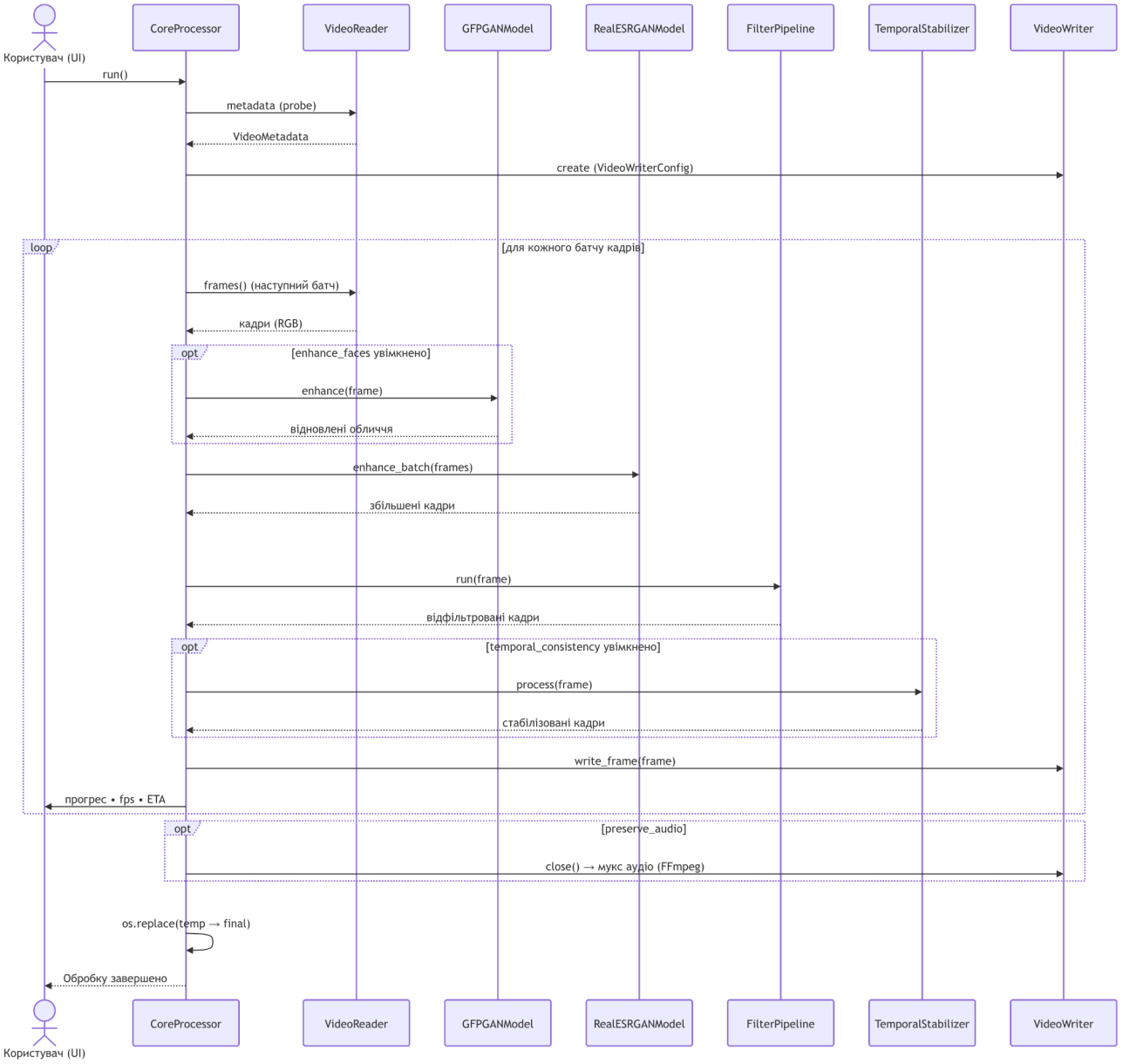

Рисунок 2.4 – Діаграма послідовності обробки відеокадру

Конвеєр постобробки реалізовано патерном «конвеєр»: фабрика build_filter_pipeline додає кроки залежно від прапорців конфігурації, а кожен крок виконується ізольовано — збій окремого фільтра логується, але не перериває обробку. Скорочено побудову конвеєра показано в лістингу 2.4.

Лістинг 2.4 – Побудова конвеєра фільтрів (фрагмент)

```
pipeline = FilterPipeline()
pipeline.add_step(config.remove_haze, remove_haze, "Dehaze")
pipeline.add_step(config.enhance_colors_flag, enhance_colors, "Color
```

```
Enhancement")
pipeline.add_step(config.sharpen_edges,
                  lambda img: sharpen_edges(img, factor=0.2), "Edge
Sharpening")
pipeline.add_step(config.postprocess, postprocess_image,
"Postprocessing")
```

Окремі фільтри реалізовано засобами OpenCV і Pillow: корекція кольору застосовує адаптивне вирівнювання гістограми з обмеженням контрасту (CLAHE) у просторі LAB [12], усунення серпанку — вирівнювання гістограми та гамма-корекцію, шумозаглушення — нелокальне усереднення, прибирання JPEG-артефактів — медіанну фільтрацію.

Оптимізатор пам'яті MemoryOptimizedProcessor розбиває велике зображення на тайли з перекриттям, обробляє їх по черзі та склеює результат із плавним змішуванням країв за допомогою градієнтних масок, що усуває видимі шви між тайлами.

Модуль часової узгодженості TemporalStabilizer реалізує стабілізацію відео як етап постобробки. Для кожного апскейленого кадру оцінюється щільний оптичний потік від попереднього кадру методом Фарнебека [12], попередній стабілізований кадр переноситься (warping) у систему координат поточного, після чого кадри змішуються лише в ділянках із надійною оцінкою руху. Надійність визначається перевіркою узгодженості прямого та зворотного потоку: ділянки з великою похибкою (оклюзії, швидкий рух) не змішуються, що запобігає двоїнню. Ключову операцію змішування наведено нижче.

```
weight = (self.blend * reliable_full)[..., None]
stabilized = (1.0 - weight) * current + weight * warped_prev
```

Модуль є відстановим лише в межах одного відео: він зберігає попередній кадр і скидає стан перед кожним новим кліпом, а кадри обробляються строго в порядку відтворення.

Введення-виведення відео інкапсульовано у класах VideoReader і VideoWriter. VideoReader лениво зчитує кадри MP4 через imageio/FFmpeg, зчитує

метадані та контролює обмеження за розміром, повертаючи кадри як зображення PIL. VideoWriter кодує оброблені кадри у потік raw RGB, що передається у FFmpeg для кодування libx264 (yuv420p, прапорець +faststart), записує результат у приватний тимчасовий файл і лише після успіху атомарно переміщує його у кінцеве розташування, що унеможливлює появу пошкоджених файлів при збої чи скасуванні. За увімкненої опції збереження звуку другий прохід FFmpeg копіює аудіодоріжку з оригіналу без перекодування; за відсутності доріжки вихід залишається без звуку, не перериваючи обробку.

Реалізація CoreProcessor побудована так, щоб одна й та сама логіка фільтрів використовувалася для зображень і для відеокадрів. Це зменшує дублювання: корекція кольору, постобробка, усунення JPEG-артефактів і підкреслення контурів підключаються фабрикою build_filter_pipeline залежно від ProcessingConfig. У разі помилки окремого фільтра система не повинна втрачати весь запуск; помилка фіксується, а користувач отримує підсумкове повідомлення. Така поведінка відповідає практичному сценарію пакетної обробки, де один проблемний файл не повинен блокувати решту результатів.

Для зображень з альфа-каналом передбачено вилучення прозорості перед обробкою та її повернення після завершення. Це важлива деталь для PNG-файлів із прозорим фоном: більшість моделей і фільтрів працюють з RGB-поданням, але втрата альфа-каналу була б помітним функціональним дефектом. Якщо масштабування змінює розмір зображення, альфа-канал також масштабується до нового розміру, після чого знову приєднується до результату.

Відеообробка реалізована потоково, а не через попереднє завантаження всіх кадрів у пам'ять. VideoReader надає ітератор кадрів, CoreProcessor обробляє кадри малими групами, а VideoWriter поступово передає результат у FFmpeg. Завдяки цьому пам'ять витрачається переважно на поточний батч, модель і буфери кодування, а не на весь відеофайл. Такий підхід особливо важливий для довших роликів, де повне розпакування кадрів могло б перевищити доступний обсяг оперативної пам'яті.

Модуль TemporalStabilizer підключений після збільшення та фільтрації

кадру, тобто працює з фінальним просторовим представленням. Це дає змогу оцінювати міжкадрові зміщення вже в тій роздільності, яку побачить користувач. Для першого кадру стабілізація не застосовується, оскільки немає попереднього кадру для оцінювання руху; наступні кадри узгоджуються з попереднім результатом за допомогою оптичного потоку. Параметр temporal_strength керує силою змішування та дає змогу балансувати між стабільністю і збереженням нових деталей.

Засоби журналювання доповнюють інтерфейсний статус. Streamlit показує короткі повідомлення про поточний файл, кадр і прогрес, а файл logs/app.log зберігає технічні подробиці для діагностики. Це розділення корисне під час супроводу: користувач не перевантажений службовою інформацією, але розробник може відтворити причину помилки за журналом, командою FFmpeg або повідомленням моделі.

2.7 Розробка інтерфейсу користувача

Інтерфейс користувача побудовано на Streamlit як односторінковий веб-застосунок. У бічній панелі згруповано всі налаштування: вибір типу входу (зображення або відео MP4), пресет (Speed, Balanced, Quality), режим масштабування, прапорці покращення (відновлення облич, попередня й завершальна обробка, корекція кольору), фільтри (усунення серпанку, прибирання JPEG-артефактів, стилізація, підкреслення контурів), повзунки насиченості й контрасту, налаштування продуктивності (напівточність, обмеження за мегапікселями) та параметри виводу. Для відео додатково доступні керування збереженням звуку та параметри часової узгодженості. Пресети задають типові набори значень: режим Speed вимикає додаткову обробку заради швидкості, Quality вмикає повний набір покращень, Balanced є проміжним.

Основна область відображає кількість завантажених файлів, розгортуваний блок стану системи (доступність CUDA, назва відеокарти й обсяг відеопам'яті, використання оперативної пам'яті, доступність MPS і середовища FFmpeg для

відео) та кнопку запуску обробки. Під час обробки відображаються індикатор прогресу й текстовий статус із поточною швидкістю в кадрах за секунду та оцінкою часу, що залишився. Результати показуються безпосередньо в інтерфейсі: для зображень — сіткою з кнопками завантаження кожного файлу, для відео — вбудованим програвачем із кнопкою завантаження. Загальний вигляд інтерфейсу та етапи роботи з ним наведено на рисунках 2.5–2.7.

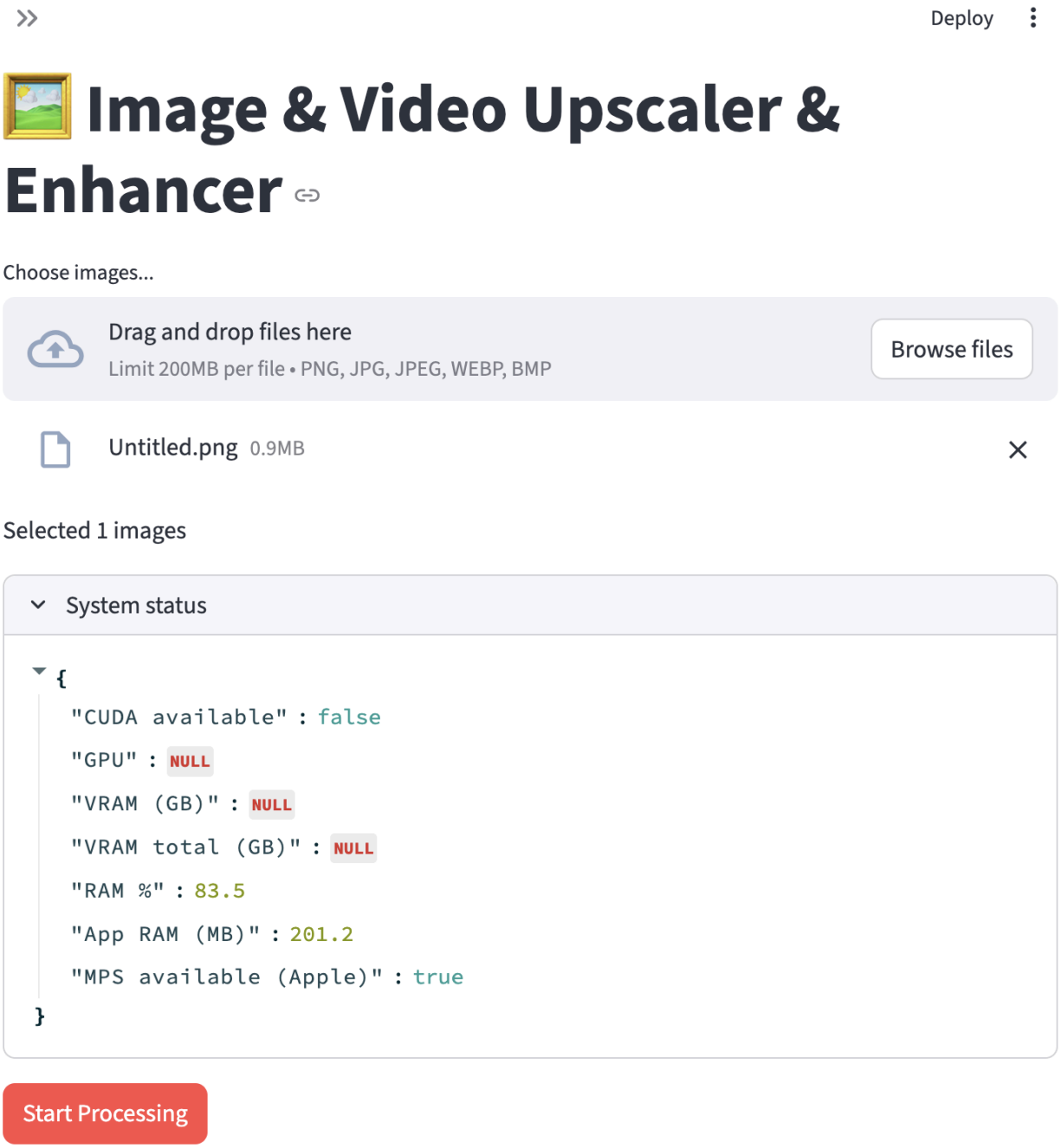

Рисунок 2.5 – Головне вікно інтерфейсу користувача

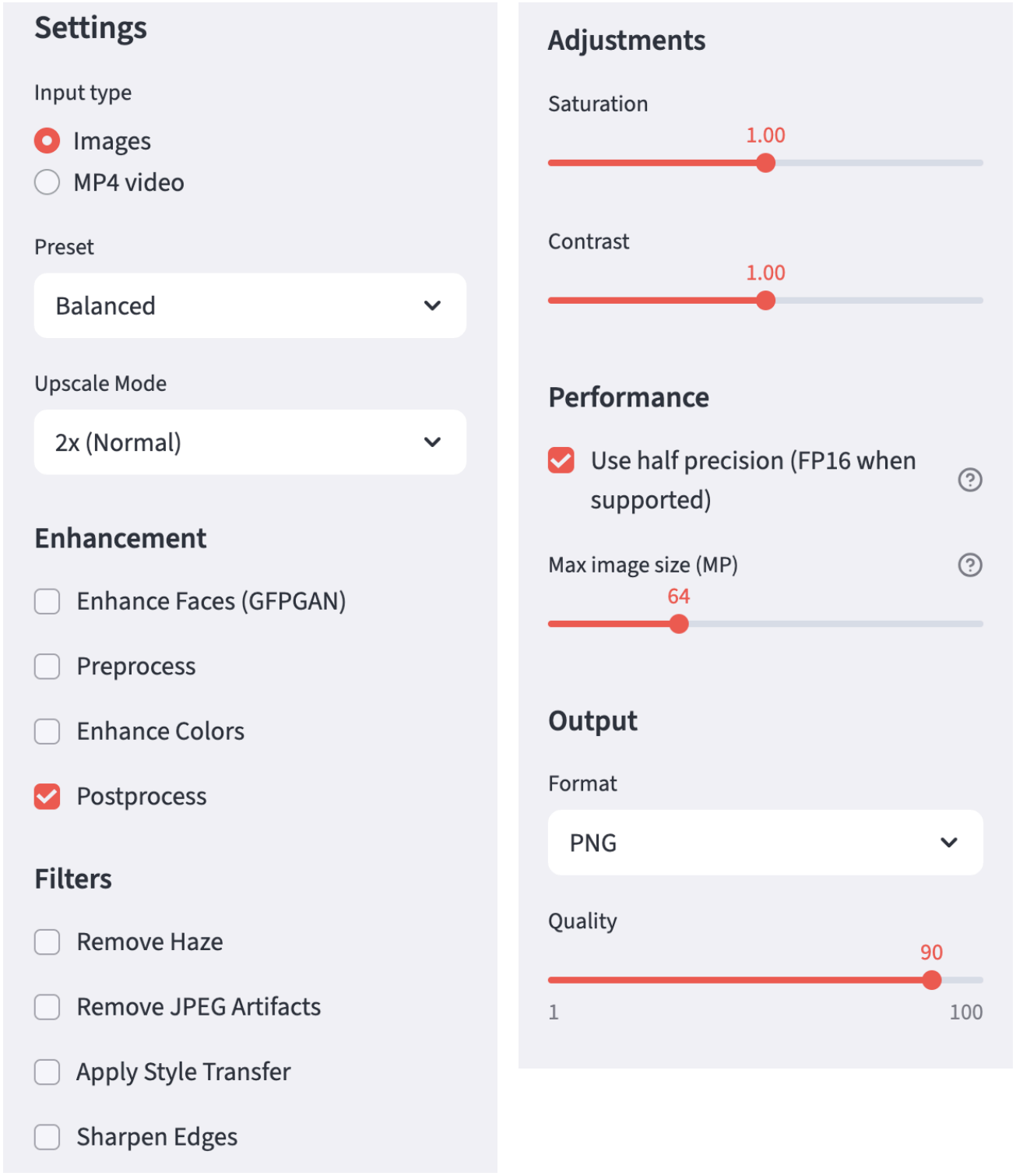

Рисунок 2.6 – Бічна панель налаштувань обробки

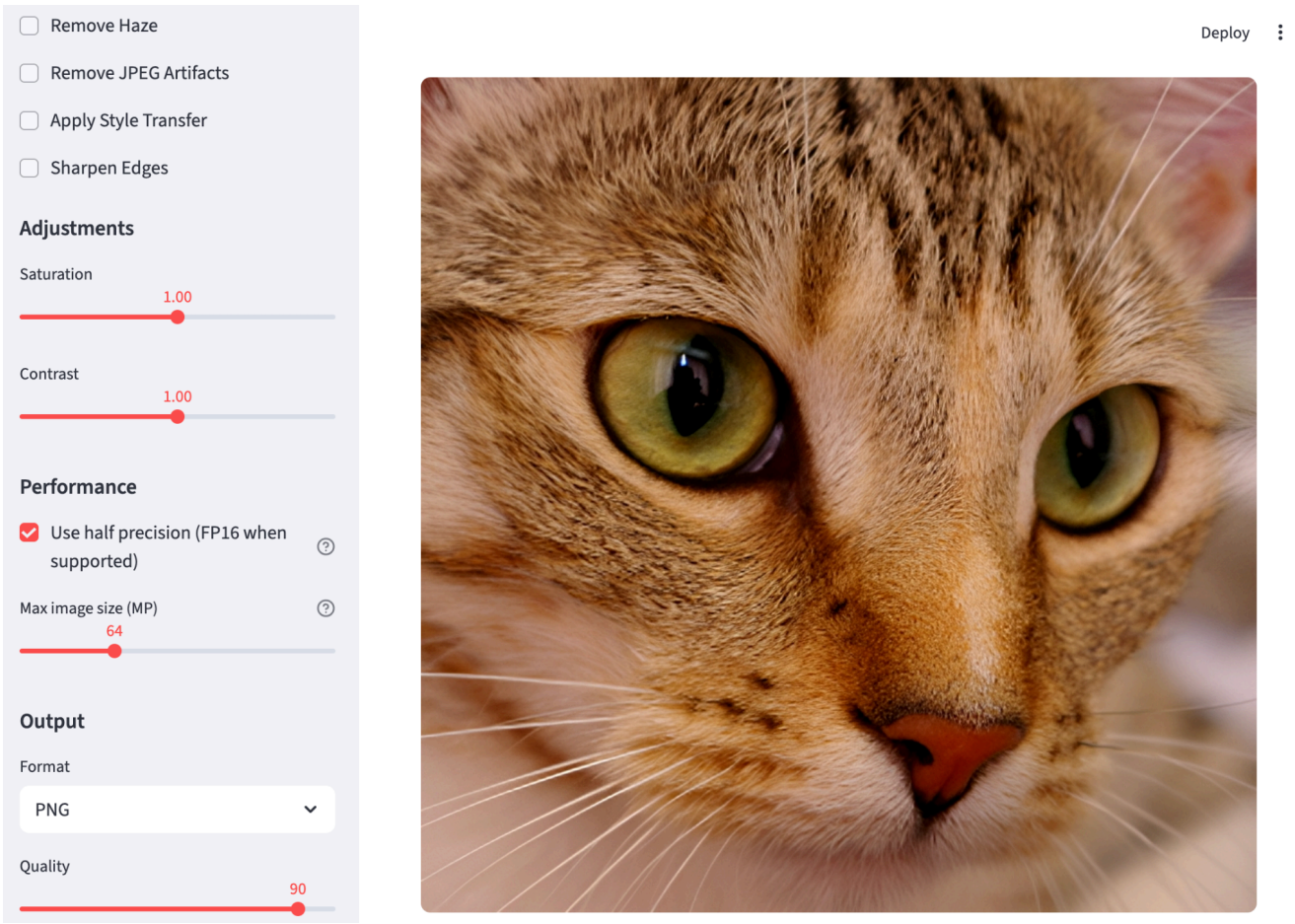

Рисунок 2.7 – Відображення результатів обробки

Інтерфейс не приховує технічні обмеження системи, але подає їх у прикладному вигляді. Користувач бачить обмеження розміру, доступність GPU/MPS, стан FFmpeg для відео та формат виводу. Кнопка запуску блокується, якщо для відео недоступне середовище кодування, а після завершення обробки результат відображається поряд із кнопкою завантаження. Це зменшує кількість помилкових сценаріїв, коли користувач запускає операцію, яка наперед не може бути виконана.

Для режимів Images і MP4 video використано спільну структуру панелі, але різні параметри виводу. У режимі зображень доступні PNG, JPEG і WEBP та повзунок якості, тоді як у відеорежимі фіксується MP4 і відкриваються параметри збереження аудіо та часової узгодженості. Така адаптивність інтерфейсу важлива для зручності: користувач не бачить налаштувань, які не мають сенсу для поточного типу медіа, і швидше знаходить релевантні параметри.

Показані на рисунках 2.5-2.7 екрани отримані з запущеного локального застосунку. Це дозволяє перевірити не лише задум інтерфейсу, а й реальний стан компонентів Streamlit: завантаження файлу, роботу бічної панелі, блок системного статусу, прогрес виконання, повідомлення про успішну обробку та область Results. Таким чином, графічний матеріал відповідає реалізованій програмній системі, а не є умовним макетом.

2.8 Висновки до другого розділу

У другому розділі обґрунтовано вибір ітеративно-інкрементного процесу розробки з опорою на практики контролю версій, автоматизованого тестування й контейнеризації. Спроєктовано багаторівневу архітектуру системи з інкапсуляцією компонентів глибокого навчання за фасадом ModelManager, реалізованим як одинак із LRU-кешуванням моделей. Описано структури даних системи та обґрунтовано відсутність бази даних з огляду на відстановий характер обробки, побудовано об'єктно-орієнтовану модель і UML-діаграму класів.

Обґрунтовано вибір мови Python і набору засобів: PyTorch, пакетів Real-ESRGAN, GFPGAN і basicsr, бібліотек OpenCV, Pillow і scikit-image, фреймворку Streamlit та FFmpeg для роботи з відео. Наведено реалізацію основних класів і методів, зокрема менеджера моделей, детектора апаратних можливостей, обгорток моделей, оркестратора обробки, конвеєра фільтрів, оптимізатора пам'яті, модуля часової узгодженості та засобів введення-виведення відео, а також розроблено веб-інтерфейс користувача. Це створює основу для тестування, оцінювання якості та розгортання системи, розглянутих у наступному розділі.

3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА ПРОГРАМНОЇ СИСТЕМИ

У цьому розділі описано види й план тестування програмної системи, наведено результати модульного та функціонального тестування, оцінено ефективність і виконано верифікацію за метриками якості та часової узгодженості, а також подано інструкцію з розгортання й підтримки системи.

3.1 Види та план тестування

Тестування системи організовано на кількох рівнях. Модульне тестування перевіряє коректність окремих класів і функцій в ізоляції за допомогою фреймворку pytest [22]; усі тести автоматично запускаються при кожному внесенні змін через систему безперервної інтеграції GitHub Actions [23], що унеможливлює непомітну появу регресій. Функціональне тестування підтверджує відповідність поведінки системи варіантам використання, визначеним у першому розділі. Окремо виконано оцінювання ефективності та верифікацію — кількісне вимірювання якості відновлення за метриками PSNR, SSIM і LPIPS, ефективності модуля часової узгодженості за похибкою перенесення та продуктивності за часом обробки й використанням пам'яті.

Усі вимірювання проводилися в такому середовищі: ноутбук MacBook Air (архітектура arm64, macOS), 8 ГБ оперативної пам'яті, без доступних прискорювачів CUDA та MPS, тобто у режимі обчислень на центральному процесорі. Відповідно, у результатах продуктивності відсутні показники відеопам'яті (їх позначено як недоступні), а замість них наведено споживання оперативної пам'яті процесу (RSS). Слід враховувати, що отримані часові показники відображають роботу на центральному процесорі; передбачена архітектурою апаратна акселерація на відеокартах (розділ 2) дозволяє істотно підвищити продуктивність, проте на цьому стенді вона не вимірювалася.

План тестування сформовано за принципом тестової піраміди: найбільшу

частину становлять швидкі модульні тести, меншу — інтеграційні перевірки роботи з відео та файловою системою, а функціональні сценарії виконуються вручну на рівні користувацького інтерфейсу. Такий розподіл є доцільним для системи з важкими моделями глибокого навчання: більшість логіки можна перевірити без фактичного інференсу, а дорогі запускі залишити для контрольних бенчмарків і демонстраційних сценаріїв.

Для кожного компонента визначено очікувані інваріанти. Для CoreProcessor це правильний порядок кроків, коректне оновлення прогресу, ізоляція помилок окремих файлів і збереження альфа-каналу. Для ModelManager — повторне використання моделей із кешу, витіснення за LRU та вибір пристрою. Для VideoWriter — атомарність фінального запису, коректна команда FFmpeg і збереження аудіо, якщо це дозволяє конфігурація. Для TemporalStabilizer — коректна обробка першого кадру, reset-стану та кадрів різного розміру.

Тестові дані підібрані так, щоб охопити не лише типовий шлях, а й крайові випадки: малі RGB-зображення, зображення з альфа-каналом, штучні відео з відомою кількістю кадрів, відсутні або некоректні ваги моделей, завеликі файли та помилки зовнішніх утиліт. Частина тестів використовує monkeypatch і mock-об'єкти, щоб відтворювати помилки без залежності від реальної відеокарти чи мережі. Це підвищує повторюваність результатів у локальному середовищі та в CI.

Критеріями приймання для модульних тестів є стабільне проходження всього набору без пропусків і без залежності від порядку запуску. Для функціональних сценаріїв критерієм є відповідність поведінки варіантам використання з першого розділу: файл має завантажуватися, параметри мають впливати на ProcessingConfig, результат має з'являтися в інтерфейсі, а користувач повинен отримувати повідомлення про помилки у зрозумілій формі. Для кількісної верифікації окремо оцінюються якість, час і пам'ять, оскільки жодна з цих характеристик не є достатньою сама по собі.

3.2 Модульне тестування

Систему покрито 89 автоматизованими модульними тестами, що охоплюють усі основні компоненти. Тести використовують підставні об'єкти (mock) для моделей і зовнішніх залежностей, що дозволяє перевіряти логіку обробки без завантаження ваг і без доступу до відеокарти. Розподіл тестів за компонентами наведено в таблиці 3.1.

Таблиця 3.1 – Розподіл модульних тестів за компонентами системи

	К-ть тестів	**Що перевіряється**
Ядро обробки (CoreProcessor)	18	розбір режимів, порядок кроків, збереження альфа-каналу, прогрес, відеообробка, скасування та ізоляція помилок
Завантаження ваг (download_weights)	11	перевірка розміру й SHA256, режим check-only, вибір ваг і символьні посилання
Писач відео (VideoWriter)	16	команда FFmpeg, аудіо, парність розмірів, атомарність запису, відмови й інтеграційний MP4
Читач відео (VideoReader)	8	метадані, потокова видача RGB-кадрів, обмеження розміру та помилки форматів
Детекція апаратури (GPUCapabilities)	6	евристики для CPU, MPS і CUDA-відеокарт різних поколінь
Оптимізатор пам'яті (MemoryOptimizedProcessor)	5	тайлування, геометрія, інкрементальна обробка і градієнтні маски
Шляхи до ваг (model_paths)	5	каталог ваг, змінна середовища, шляхи Real-ESRGAN і GFPGAN
Інференс Real-ESRGAN (enhance_batch)	5	батчева обробка, доповнення під кратність масштабу, завантаження чекпойнта
Часова узгодженість (TemporalStabilizer)	7	перший кадр, ідентичні кадри, форма виходу, reset, напрямок warping і зміна розміру
Менеджер моделей (ModelManager)	3	витіснення LRU, кеш і вибір пристрою
Метрики якості (quality_metrics)	2	перевірка мінімальної кількості кадрів і підсумкові temporal-метрики
Бенчмарк (benchmark_processing)	3	розбір розмірів і наскрізний smoke-прогін

Усі тести проходять успішно, що підтверджує коректність реалізації окремих компонентів. Успішний прогін набору тестів наведено на рисунку 3.1.

```
tests/test_benchmark_processing.py ...                    [  3%]
tests/test_core_processor_helpers.py .......              [ 11%]
tests/test_core_processor_pipeline.py ...                 [ 14%]
tests/test_core_processor_video.py ........               [ 23%]
tests/test_download_weights.py ...........                [ 35%]
tests/test_gpu_capabilities.py ......                     [ 42%]
tests/test_memory_optimizer.py .....                      [ 48%]
tests/test_model_manager.py ...                           [ 51%]
tests/test_model_paths.py .....                           [ 57%]
tests/test_quality_metrics.py ..                          [ 59%]
tests/test_real_esrgan_batch.py .....                     [ 65%]
tests/test_temporal_consistency.py .......                [ 73%]
tests/test_video_reader.py ........                       [ 82%]
tests/test_video_writer.py ................               [100%]

========================== 89 passed in 2.54s ==========================
```

Рисунок 3.1 – Успішний прогін автоматизованих тестів pytest

Фактичний набір тестів після доопрацювання містить 89 перевірок. Це не лише позитивні сценарії, а й негативні випадки: відмова відкриття файлу, перевищення max_megapixels, помилка FFmpeg, відсутність ваг, некоректний checkpoint, порушення парності розмірів кадру для H.264, скасування обробки та часткове завершення батчу. Наявність таких тестів важливо, бо саме обробка відмов визначає практичну надійність застосунку під час роботи з довільними користувацькими файлами.

Тести ядра обробки навмисно відокремлені від реальних моделей. Замість завантаження Real-ESRGAN і GFPGAN використовуються підставні класи, які повертають контрольовані зображення або штучно генерують винятки. Завдяки цьому перевіряється логіка оркестрації: чи викликаються потрібні кроки, чи коректно рахується прогрес, чи не втрачається альфа-канал і чи продовжується пакетна обробка після помилки одного файлу. Важкі моделі залишаються предметом окремих smoke- і benchmark-запусків.

Окрему групу становлять тести відео. Вони перевіряють метадані, потокову видачу кадрів, команду кодування FFmpeg, домішування аудіо, атомарне перейменування результату та видалення тимчасового файлу в разі помилки. Такі перевірки дають упевненість, що відеоконвеєр поводиться передбачувано не лише

тоді, коли обробка успішна, а й тоді, коли зовнішній процес завершується з помилкою.

Автоматизований прогін pytest є частиною регресійного контролю. Після зміни реалізації очікується, що весь набір тестів проходить однією командою *.venv/bin/python -m pytest -q tests*. Успішний результат 89 passed підтверджує, що внесені зміни до інтерфейсу, конфігурації, обробки відео та допоміжних модулів не зламали існуючу поведінку.

3.3 Функціональне тестування

Функціональне тестування виконано вручну за сценаріями, що відповідають варіантам використання. Для кожного сценарію перевірялася відповідність фактичного результату очікуваному. Результати наведено в таблиці 3.2.

Таблиця 3.2 – Результати функціонального тестування за варіантами використання

	Сценарій	**Очікуваний результат**	**Статус**
1	Пакетний апскейл зображень (2x/4x)	усі зображення збільшено, доступні для завантаження	пройдено
2	Апскейл відео MP4 зі збереженням звуку	вихідний MP4 (H.264) з оригінальною аудіодоріжкою	пройдено
3	Перемикання пресетів (Speed/Balanced/Quality)	набір фільтрів змінюється згідно з пресетом	пройдено
4	Відновлення облич (GFPGAN)	риси облич відновлено, артефактів не додано	пройдено
5	Конвеєр фільтрів (колір, серпанок, контури)	застосовано лише ввімкнені фільтри	пройдено
6	Часова узгодженість для відео	зменшення мерехтіння між кадрами	пройдено
7	Обробка завеликого зображення	файл пропущено з повідомленням, обробка триває	пройдено
8	Відео без аудіодоріжки	вихід без звуку, без аварійного завершення	пройдено

Окремо система безперервної інтеграції на кожній зміні перевіряє коректність конфігурацій Docker та шлях контролю відсутніх ваг, що підтверджує

готовність системи до розгортання.

3.4 Оцінювання ефективності та верифікація

3.4.1 Якість відновлення зображень

Для оцінювання якості сформовано перевірочний набір із 8 еталонних зображень високої роздільної здатності. З кожного бікубічним зменшенням у 4 рази створено вхід низької роздільної здатності, який потім відновлено двома способами: бікубічною інтерполяцією (базовий рівень) та моделлю Real-ESRGAN (x4). Відновлені зображення порівняно з еталоном за метриками PSNR, SSIM (більше — краще) і LPIPS (менше — краще). Усереднені результати наведено в таблиці 3.3.

Таблиця 3.3 – Якість відновлення: бікубічна інтерполяція проти Real-ESRGAN (x4), усереднення за 8 зображеннями

	PSNR, дБ ↑	**SSIM ↑**	**LPIPS ↓**
Бікубічна інтерполяція	28,01	0,824	0,290
Real-ESRGAN (x4)	24,55	0,736	0,203

Отримані результати потребують коректної інтерпретації. За піксельними метриками PSNR і SSIM вищі показники має бікубічна інтерполяція. Це закономірно: PSNR і SSIM оцінюють піксельну й структурну близькість до еталона, тоді як Real-ESRGAN — генеративна модель, що відновлює правдоподібні високочастотні деталі, які можуть не збігатися піксель-у-піксель із конкретним еталоном. Водночас за перцептивною метрикою LPIPS, яка краще корелює зі сприйняттям людини [13], Real-ESRGAN суттєво переважає бікубічну інтерполяцію (0,203 проти 0,290). Це узгоджується з відомим компромісом між перцептивною якістю та спотворенням (perception–distortion tradeoff): підвищення перцептивної реалістичності зображення, як правило, супроводжується погіршенням піксельних метрик відстані до еталона [14]. Тобто на синтетичному

перевірочному наборі бікубічна інтерполяція дає результат, формально ближчий до еталона за пікселями, але Real-ESRGAN дає результат, перцептивно ближчий до нього й візуально різкіший. Приклади відновлення наведено на рисунках 3.2–3.3.

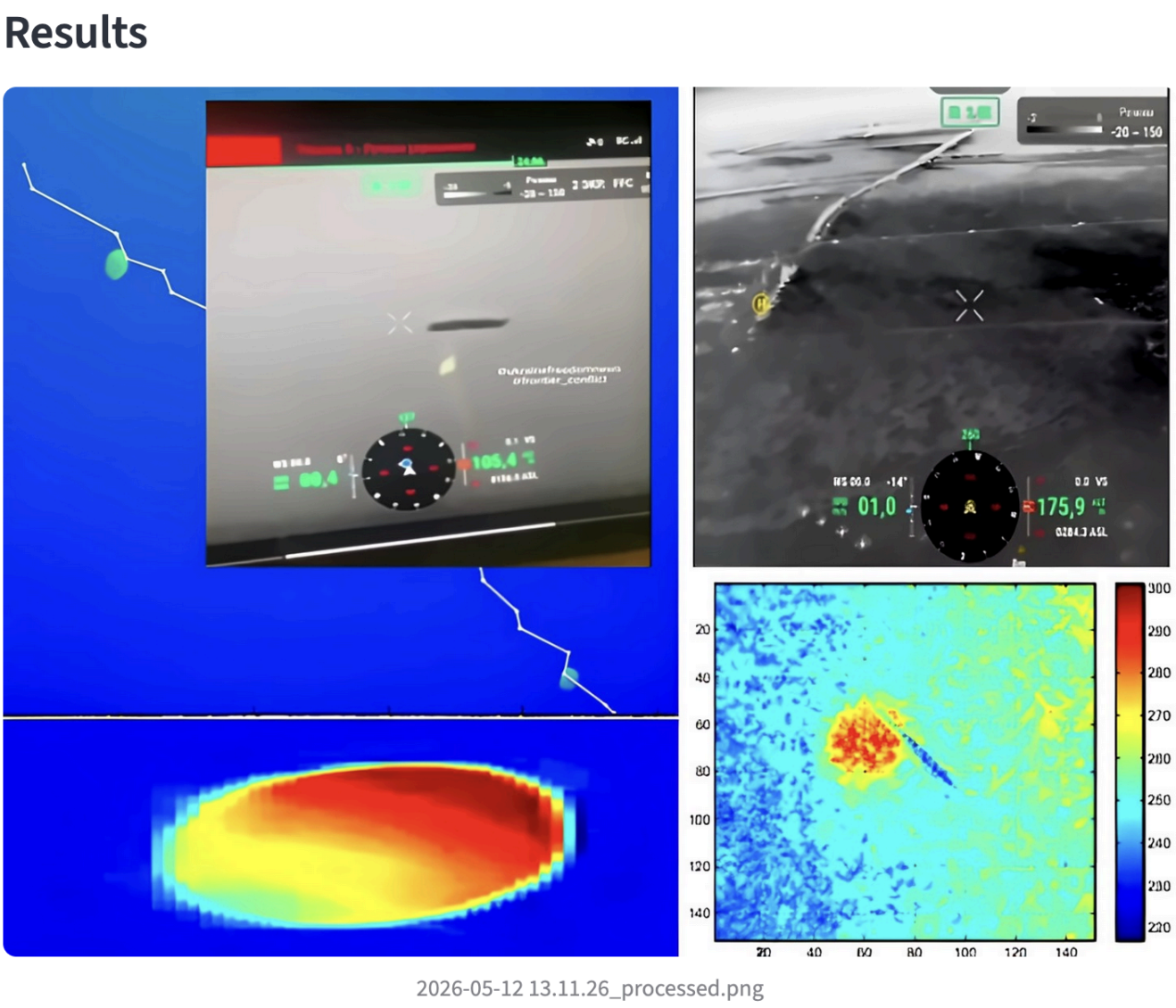

Рисунок 3.2 – Приклад відновлення зображення моделлю Real-ESRGAN

Рисунок 3.3 – Порівняння бікубічної інтерполяції та Real-ESRGAN

Аналіз за окремими зображеннями показує, що перевага LPIPS у Real-ESRGAN зберігається для більшості зображень із природними текстурами, тоді як на зображенні з текстом базовий рівень очікувано кращий за всіма метриками, оскільки генеративне відновлення гострих контурів тексту є складнішим випадком.

3.4.2 Ефективність модуля часової узгодженості

Ефективність модуля часової узгодженості оцінено на тестовій послідовності з 24 кадрів роздільної здатності 128x128, обробленій моделлю Real-ESRGAN (x2) із силою стабілізації 0,3. Показником слугувала похибка перенесення (warping error) між сусідніми кадрами, обчислена за оптичним потоком: що вона менша, то узгоджена послідовність у часі. Порівняння обробки з увімкненим і вимкненим модулем наведено в таблиці 3.4.

Таблиця 3.4 – Похибка перенесення між кадрами з вимкненим і ввімкненим модулем часової узгодженості

	Вимкнено	**Увімкнено**	**Покращення**
Середнє	33,62	20,23	39,81 %
Медіана	28,71	17,53	38,96 %
95-й перцентиль	55,65	38,18	31,41 %
Максимум	81,76	50,68	38,01 %

Увімкнення модуля зменшує середню похибку перенесення майже на 40 %, причому покращення спостерігається за всіма показниками — від медіани до максимуму. Це означає, що стабілізація не лише покращує окремі вдалі кадри, а системно зменшує розбіжності між сусідніми кадрами в усій тестовій послідовності. Зниження максимального та 95-го перцентиля також свідчить, що модуль зменшує не тільки середній рівень мерехтіння, а й найбільш помітні короткочасні стрибки якості, які зазвичай найсильніше сприймаються користувачем під час перегляду відео. Це безпосередньо підтверджує, що запропонований модуль часової стабілізації на основі оптичного потоку дієво зменшує міжкадрове мерехтіння, властиве покадровому апскейлінгу, тобто виконує задачу забезпечення часової узгодженості кадрів, винесену в тему роботи. Візуальну різницю між кадрами з увімкненим і вимкненим модулем наведено на рисунку 3.4.

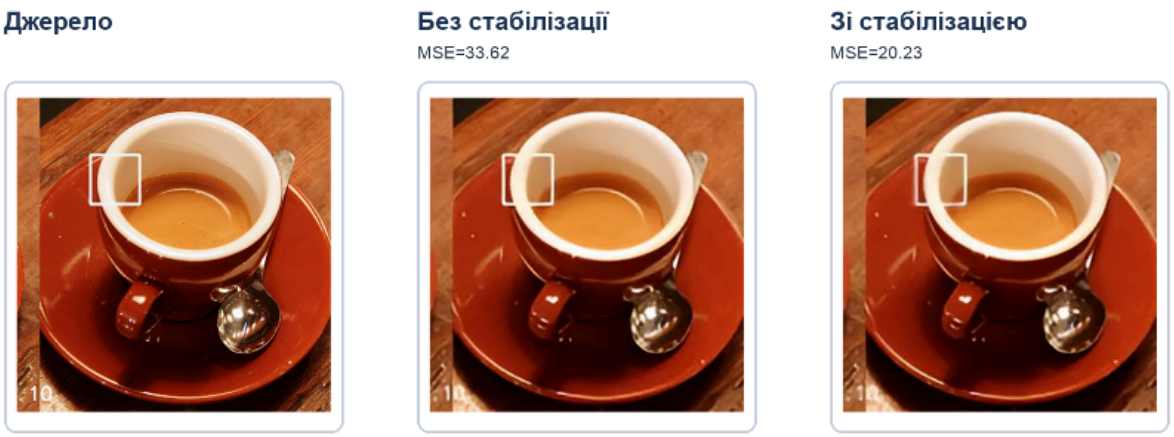

Рисунок 3.4 – Порівняння кадрів із вимкненим і ввімкненим модулем часової узгодженості

3.4.3 Продуктивність і використання пам'яті

Продуктивність виміряно вбудованим інструментом бенчмаркінгу на згенерованих зображеннях фіксованих розмірів (по три прогони на конфігурацію, наведено усереднені значення). Результати для центрального процесора подано в таблиці 3.5.

Таблиця 3.5 – Продуктивність обробки (середнє за 3 прогони, режим CPU)

	Розмір	**Середній час, с**	**Пропускна здатність, MP/s**	**Макс. RSS, МБ**	**Відеопам'ять**
Без апскейлу + фільтри	512x512	0,059	7,38	365	н/д
Без апскейлу + фільтри	1280x720	0,083	11,11	402	н/д
Real-ESRGAN x2	128x128	1,69	0,0102	475	н/д
Real-ESRGAN x2	256x256	6,43	0,0102	497	н/д
Real-ESRGAN x4	128x128	6,91	0,0024	344	н/д

Результати показують, що операції постобробки без апскейлінгу виконуються майже миттєво з високою пропускною здатністю, тоді як інференс Real-ESRGAN на центральному процесорі є обчислювально витратним: час обробки зростає зі збільшенням розміру входу та коефіцієнта масштабування. Споживання оперативної пам'яті залишається помірним (у межах приблизно 0,5 ГБ) і не перевищує можливостей тестового стенда з 8 ГБ. Перший прогін кожної конфігурації включає одноразові витрати на ініціалізацію, тому є дещо повільнішим за наступні. Наведені часові показники відповідають режиму центрального процесора; на відеокарті з підтримкою напівточності та більшого розміру батчу пропускна здатність очікувано буде значно вищою, що становить напрям подальшого вимірювання.

Метрики якості й продуктивності потрібно розглядати разом. Генеративна модель може програвати за PSNR або SSIM, але давати кращу перцептивну якість за LPIPS; навпаки, швидкий режим без апскейлінгу має високу продуктивність, але не розв'язує задачу підвищення роздільної здатності. Тому верифікація побудована як багатокритеріальна: окремо оцінено піксельні метрики, перцептивну метрику, часову узгодженість, час обробки та використання пам'яті.

Для відтворюваності бенчмарків зафіксовано розміри тестових зображень, кількість прогонів і режим обчислень. Перший запуск моделі може включати ініціалізацію ваг і кешів, тому для оцінювання продуктивності використано

усереднення за кількома прогонами. У звіті окремо зазначено, що наведені числа отримані на CPU-конфігурації; на GPU вони мають інший масштаб, але відносна тенденція між режимами зберігається: збільшення коефіцієнта масштабування й розміру входу підвищує час обробки.

Під час аналізу часової узгодженості warping error обчислюється між сусідніми кадрами після компенсації руху оптичним потоком. Це не повна заміна візуальної оцінки, але метрика добре показує, чи зменшується міжкадрове мерехтіння після ввімкнення стабілізації. Зменшення середнього значення майже на 40 % підтверджує, що модуль працює не лише формально, а й має вимірюваний ефект на відеопослідовності.

До обмежень проведеної верифікації належить невеликий тестовий набір і відсутність широкого порівняння на різних GPU. Це не знижує цінності результатів для кваліфікаційної роботи, але визначає напрям подальших досліджень: розширити набір відео, перевірити різні типи руху, додати суб'єктивне оцінювання користувачами та порівняти продуктивність на CUDA-пристроях з різним обсягом VRAM.

3.5 Інструкція користувача та розгортання системи

Для локального запуску потрібні Python 3.11, встановлені залежності з файлу requirements.txt (або requirements-cpu.txt для систем без відеокарти) та ваги моделей, які завантажуються сценарієм download_weights.py із перевіркою цілісності за SHA256. Застосунок запускається командою streamlit run streamlit_app.py, після чого інтерфейс стає доступним у браузері за локальною адресою. Користувач обирає тип входу (зображення або відео), завантажує файли, налаштовує режим і параметри обробки та запускає її кнопкою, а після завершення переглядає й завантажує результати.

Для відтворюваного розгортання на сервері з відеокартою NVIDIA підготовлено образ Docker на базі офіційного образу PyTorch із CUDA. Запуск виконується командою docker compose -f docker-compose.cuda.yml up --build; ваги

моделей зберігаються у постійному томі, а інтерфейс публікується на порту 8501. Для перевірки складання образу на системах без відеокарти (зокрема macOS) передбачено окрему «димову» конфігурацію docker-compose.smoke.yml. Образ містить системний FFmpeg для обробки відео та виконується від імені непривілейованого користувача, а перевірка стану (healthcheck) контролює доступність вебінтерфейсу.

Мінімальні системні вимоги для роботи без апаратної акселерації — 8 ГБ оперативної пам'яті; для прийнятної продуктивності апскейлінгу рекомендовано відеокарту NVIDIA з підтримкою CUDA та обсягом відеопам'яті, достатнім для обраного режиму. Підтримку системи спрощують ведення журналу подій у файл, ізоляція помилок окремих файлів і фільтрів, а також автоматичні тести, що виконуються при кожній зміні коду.

Супровід системи передбачає контроль трьох груп артефактів: програмного коду, ваг моделей і середовища виконання. Код змінюється через Git і перевіряється тестами; ваги моделей мають завантажуватися лише через сценарій із перевіркою розміру й SHA256; середовище виконання фіксується requirements-файлами або Docker-образом. Такий поділ дозволяє оновлювати кожну групу окремо та швидко локалізувати причину помилки після розгортання.

Для практичного впровадження важливо передбачити сценарій без GPU. Система може запускатися на CPU для перевірки інтерфейсу, функціональних сценаріїв і невеликих файлів, але для продуктивного використання апскейлінгу відео рекомендовано CUDA-сервер. Саме тому підготовлено окремі конфігурації: легший локальний запуск для розробника та Docker Compose для GPU-хоста. У документації до розгортання це потрібно розділяти, щоб користувач не очікував однакової швидкості на всіх пристроях.

Під час експлуатації бажано контролювати розмір вхідних файлів, кількість паралельних користувачів і вільний дисковий простір для тимчасових каталогів. Навіть якщо програма не зберігає історію запусків, під час активної обробки відео тимчасові файли можуть займати значний обсяг. Тому для серверного розгортання доцільно додати обмеження на тривалість відео, періодичне очищення тимчасових

каталогів і моніторинг журналів помилок.

3.6 Висновки до третього розділу

У третьому розділі описано багаторівневу стратегію тестування системи та наведено її результати. 89 автоматизованих модульних тестів, що виконуються у середовищі безперервної інтеграції, підтверджують коректність окремих компонентів, а функціональне тестування — відповідність поведінки системи варіантам використання.

Виконано кількісну верифікацію. За перцептивною метрикою LPIPS модель Real-ESRGAN переважає бікубічну інтерполяцію (0,203 проти 0,290), тоді як вищі значення PSNR і SSIM у бікубічної інтерполяції пояснюються компромісом між перцептивною якістю та піксельним спотворенням. Модуль часової узгодженості зменшує середню похибку перенесення між кадрами майже на 40 %, що безпосередньо підтверджує досягнення мети роботи в частині забезпечення часової узгодженості кадрів. Виміряно продуктивність і споживання пам'яті в режимі центрального процесора та подано інструкцію з локального й контейнеризованого розгортання систем

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

Розробка програмної системи виконується на робочому місці, обладнаному персональним комп'ютером, тому питання безпеки життєдіяльності та охорони праці безпосередньо стосуються процесу її створення й подальшого використання. У цьому розділі розглянуто ризик як кількісну оцінку небезпек та гігієнічні вимоги до організації й обладнання робочих місць з відеодисплейними терміналами стосовно робочого місця розробника програмного забезпечення.

4.1 Ризик як кількісна оцінка небезпек

Небезпека — це властивість процесів, явищ і об'єктів завдавати шкоди здоров'ю або життю людини. Оскільки повністю безпечної діяльності не існує, для зіставлення й порівняння небезпек використовують кількісну міру — ризик. Ризик визначають як міру очікуваної небезпеки, що враховує як імовірність (частоту) реалізації небажаної події, так і тяжкість її наслідків. У загальному вигляді ризик подають як добуток імовірності настання події на величину її наслідків.

Частотний (статистичний) індивідуальний ризик обчислюють як відношення кількості несприятливих подій певного виду до загальної кількості осіб, що зазнавали відповідного впливу за визначений період. Розрізняють індивідуальний ризик, що характеризує небезпеку для окремої людини, та колективний (соціальний) ризик, що характеризує небезпеку для групи людей. Залежно від величини ризик поділяють на знехтуваний, прийнятний (допустимий), гранично допустимий і неприйнятний. Концепція прийнятного ризику виходить із того, що абсолютної безпеки досягти неможливо, тому встановлюється такий рівень ризику, який суспільство готове прийняти з огляду на технічні та економічні можливості; для індивідуального ризику орієнтовним рівнем прийнятності вважають значення близько 10^{-6} на рік.

Для кількісного оцінювання ризику застосовують кілька методів: інженерний, що спирається на статистику відмов та ймовірнісні розрахунки;

модельний, що ґрунтується на побудові моделей впливу небезпек; експертний, що використовує оцінки фахівців; та соціологічний, що базується на опитуваннях. Аналіз ризику може бути апріорним (до настання події, на основі прогнозу) або апостеріорним (за наслідками подій, що вже сталися). Процес управління ризиком, відповідно до сучасних підходів до систем управління охороною здоров'я та безпекою праці [25], охоплює ідентифікацію небезпек, оцінювання ризику та впровадження заходів його зниження.

Стосовно робочого місця розробника програмного забезпечення основними джерелами небезпеки є значне зорове навантаження під час тривалої роботи з екраном, статичне навантаження на опорно-руховий апарат через тривале перебування в сидячому положенні, небезпека ураження електричним струмом під час роботи з електрообладнанням, вплив електромагнітних випромінювань та психоемоційне напруження. Більшість із цих небезпек характеризуються високою ймовірністю тривалого впливу за відносно невисокої тяжкості окремої події, однак їхній кумулятивний характер може призводити до хронічних порушень здоров'я. Тому сукупний професійний ризик розробника оцінюється як помірний, а основним засобом його зниження до прийнятного рівня є дотримання гігієнічних вимог до організації робочого місця, розглянутих далі.

4.2 Гігієнічні вимоги до організації та обладнання робочих місць з відеодисплейними терміналами

Вимоги до робочих місць, обладнаних відеодисплейними терміналами та електронно-обчислювальними машинами, регламентуються чинними нормативними документами з охорони праці [24-25], зокрема вимогами щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями [24] та державними санітарними правилами і нормами роботи з візуальними дисплейними терміналами. Їх дотримання спрямоване на зниження розглянутих вище професійних ризиків до прийнятного рівня.

Вимоги до приміщення та простору. Площа на одне робоче місце з

відеодисплейним терміналом має становити не менше 6 м², а об'єм — не менше 20 м³. Приміщення повинні мати природне й штучне освітлення; робочі місця розташовують так, щоб природне світло падало переважно збоку, з лівого боку. Не допускається розміщення робочих місць упритул до стін з вікнами без відповідного захисту від відблисків.

Вимоги до освітлення. Освітленість на поверхні робочого столу в зоні розміщення документів має становити в межах 300–500 лк. Слід уникати прямого потрапляння яскравого світла в очі та відблисків на екрані; для цього застосовують розсіяне освітлення, жалюзі на вікнах та відповідне розташування джерел світла. Нерівномірність освітлення й засліплювальна яскравість у полі зору не допускаються.

Вимоги до мікроклімату. Оптимальні параметри мікроклімату передбачають температуру повітря в межах приблизно 22–24 °С у холодний період і 23–25 °С у теплий, відносну вологість повітря 40–60 % та швидкість руху повітря не більше 0,1 м/с. Рівень шуму на робочому місці не повинен перевищувати 50 дБА.

Вимоги до організації робочого місця. Конструкція робочого місця має забезпечувати раціональну робочу позу. Відстань від очей до екрана має становити приблизно 600–700 мм. Висота робочої поверхні столу повинна відповідати зросту працівника й забезпечувати достатній простір для ніг. Робоче крісло має бути підйомно-поворотним із регулюванням висоти сидіння та кута нахилу спинки для підтримання правильного положення хребта. Екран розташовують так, щоб верхній край був приблизно на рівні очей або трохи нижче.

Вимоги до екрана та режиму праці. Екран має забезпечувати стабільне зображення без помітного мерехтіння, достатню яскравість і контраст, а також мати антивідблискові властивості. Для запобігання перевтомі встановлюють регламентовані перерви: під час роботи, що передбачає тривалу творчу взаємодію з комп'ютером (до якої належить і розробка програмного забезпечення), доцільно влаштовувати короткі перерви тривалістю 10–15 хвилин через кожну годину роботи. Під час перерв рекомендовано виконувати вправи для очей та

опорно-рухового апарату. Дотримання цих вимог зменшує зорове й статичне навантаження та знижує професійний ризик розробника до прийнятного рівня.

Для програмної системи, що розробляється, безпека праці має дві взаємопов'язані площини: організацію робочого місця розробника та керування ризиками під час експлуатації програмного забезпечення. Робота виконується переважно за комп'ютером, тому основними чинниками є тривале статичне навантаження, зорове напруження, інформаційне перевантаження, електробезпека периферійного обладнання та мікроклімат приміщення. Вимоги до роботи з екранними пристроями доцільно узгоджувати з чинними нормами щодо безпеки й захисту здоров'я працівників [24].

Таблиця 4.1 – Ідентифікація ризиків робочого місця розробника

	Можливий наслідок	**Профілактичний захід**
Тривала статична поза	перенапруження м'язів шиї, спини та плечового поясу	регульоване крісло, чергування роботи й коротких перерв
Зорове навантаження	втома очей, зниження концентрації	відстань до екрана 50-70 см, відсутність відблисків, достатня освітленість
Інформаційне перевантаження	помилки в коді, зниження продуктивності	планування задач, обмеження паралельних контекстів, контрольні списки
Електричне обладнання	ризик ураження електричним струмом	справні кабелі, заземлення, заборона перевантаження подовжувачів
Недостатній мікроклімат	дискомфорт, головний біль, швидка втома	провітрювання, температура 20-24 °C, вологість 40-60 %

Окрему увагу слід приділити ергономіці розміщення обладнання. Монітор потрібно встановлювати так, щоб верхня межа екрана була приблизно на рівні очей або трохи нижче, а площина екрана не створювала відблисків від вікон чи світильників. Клавіатура й маніпулятор мають розташовуватися на одній висоті, щоб кисті рук не перебували у вимушеному положенні. Для роботи з кодом і документацією бажано використовувати достатній розмір шрифту, контрастну тему редактора та режим автоматичного форматування, оскільки дрібний шрифт і низький контраст підвищують зорове навантаження.

Під час тестування моделей глибокого навчання додатково виникають організаційні ризики, пов'язані з тривалою роботою обчислювального обладнання. Навіть якщо розробка виконується на персональному комп'ютері, обробка відео або пакетів зображень може суттєво завантажувати центральний процесор, графічний процесор і систему охолодження. Тому небажано залишати експериментальні запуски без нагляду на робочому місці з поганою вентиляцією; потрібно контролювати температуру компонентів, стан кабелів живлення та наявність вільного простору для відведення тепла.

Таблиця 4.2 – Заходи керування ризиками під час розробки й тестування системи

	Заходи керування	**Очікуваний результат**
Організація праці	перерви 5-10 хв після кожної години інтенсивної роботи, чергування програмування, тестування й документування	зменшення втоми та кількості помилок
Ергономіка	правильна висота крісла, підтримка попереку, розміщення монітора без відблисків	зниження навантаження на опорно-руховий апарат і зір
Електробезпека	перевірка кабелів, використання справних блоків живлення, недопущення перевантаження розеток	зменшення ризику короткого замикання й ураження струмом
Тепловий режим	контроль температур CPU/GPU, провітрювання, недопущення перекриття вентиляційних отворів	стабільна робота обладнання під час тривалих тестів
Якість процесу	CI-перевірки, повторювані бенчмарки, журналювання помилок	раннє виявлення регресій і зменшення ризику некоректних результатів

Система керування охороною праці повинна базуватися на циклі виявлення небезпек, оцінювання ризиків, планування дій і перевірки результативності. Такий підхід відповідає логіці ДСТУ ISO 45001:2019, де акцент зроблено не лише на усуненні наслідків, а й на попередженні небезпечних ситуацій [25]. Для розробника програмного забезпечення це означає, що робоче місце, режим праці, обчислювальні експерименти й супровід проєкту потрібно розглядати як єдину

систему ризиків.

У межах кваліфікаційної роботи практичне застосування цих вимог полягає у виборі безпечного режиму локального запуску: обмеження максимального розміру вхідних зображень, контроль параметра max_megapixels, використання тайлової обробки для великих файлів, обробка помилок без аварійного завершення застосунку та журналювання стану системи. Такі програмні механізми не замінюють організаційні заходи охорони праці, але зменшують імовірність перевантаження обладнання та втрати результатів під час експлуатації.

4.3 Висновки до четвертого розділу

У четвертому розділі розглянуто ризик як кількісну міру небезпеки, наведено способи його обчислення та класифікації, методи оцінювання й концепцію прийнятного ризику. Проаналізовано основні джерела професійного ризику на робочому місці розробника програмного забезпечення та визначено, що основним засобом його зниження є дотримання гігієнічних вимог. Розглянуто гігієнічні вимоги до організації та обладнання робочих місць з відеодисплейними терміналами — до приміщення, освітлення, мікроклімату, організації робочого місця, екрана та режиму праці й відпочинку, дотримання яких забезпечує безпечні та комфортні умови праці й знижує професійний ризик до прийнятного рівня

ВИСНОВКИ

У кваліфікаційній роботі розв'язано актуальну задачу розробки програмної системи для підвищення роздільної здатності зображень і відеопотоку методами глибокого навчання із забезпеченням часової узгодженості кадрів.

Проаналізовано предметну область і обмеження класичних методів інтерполяції, розглянуто сучасні методи глибокого навчання для суперроздільності та проблему часової узгодженості при покадровій обробці відео. Огляд наявних рішень показав, що потужні комерційні системи є закритими й платними, а доступні відкриті інструменти переважно обробляють відео покадрово без забезпечення часової узгодженості, що обґрунтувало доцільність розробки.

Спроєктовано багаторівневу архітектуру системи з інкапсуляцією компонентів глибокого навчання за фасадом-одинаком ModelManager та реалізовано систему, яка поєднує апскейлінг на основі Real-ESRGAN, відновлення облич GFPGAN, конвеєр постобробки, тайлування великих зображень і модуль часової узгодженості на основі оптичного потоку. Система має простий вебінтерфейс і автоматично адаптується до апаратного забезпечення (CUDA, Apple MPS або центральний процесор).

Систему перевірено 89 автоматизованими тестами в середовищі безперервної інтеграції та функціональним тестуванням за варіантами використання. Кількісне оцінювання показало, що за перцептивною метрикою LPIPS модель Real-ESRGAN переважає бікубічну інтерполяцію (0,203 проти 0,290), а розроблений модуль часової узгодженості зменшує середню похибку перенесення між кадрами майже на 40 %, що безпосередньо підтверджує досягнення мети роботи в частині забезпечення часової узгодженості кадрів.

Практичне значення роботи полягає у створенні відкритого й доступного інструмента апскейлінгу зображень і відео з вебінтерфейсом, який, на відміну від поширених відкритих аналогів, забезпечує часову узгодженість кадрів і може використовуватися для відновлення контенту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wang x. et al. ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. Proc. European Conf. on Computer Vision (ECCV) Workshops. 2018.

2. Wang x., xie L., Dong C., Shan Y. Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data. Proc. IEEE/CVF ICCV Workshops. 2021. P. 1905-1914.

3. Wang x., Li Y., Zhang H., Shan Y. Towards Real-World Blind Face Restoration with Generative Facial Prior (GFPGAN). Proc. IEEE/CVF CVPR. 2021.

4. Liang J. et al. SwinIR: Image Restoration Using Swin Transformer. Proc. IEEE/CVF ICCV Workshops. 2021.

5. Chan K. C. K. et al. BasicVSR: The Search for Essential Components in Video Super-Resolution and Beyond. Proc. IEEE/CVF CVPR. 2021.

6. Chan K. C. K. et al. BasicVSR++: Improving Video Super-Resolution with Enhanced Propagation and Alignment. Proc. IEEE/CVF CVPR. 2022.

7. Lai W.-S. et al. Learning Blind Video Temporal Consistency. Proc. European Conf. on Computer Vision (ECCV). 2018.

8. Li J., Pei Z., Zeng T. A Systematic Survey of Deep Learning-Based Single-Image Super-Resolution. ACM Computing Surveys. 2024. DOI: 10.1145/3659100.

9. Chen H., He x., Qing L., Wu Y., Ren C., Sheriff R. E., Zhu C. Real-world single image super-resolution: A brief review. Information Fusion. 2022. Vol. 79. P. 124-145.

10. Liu H., Ruan Z., Zhao P. et al. Video super-resolution based on deep learning: a comprehensive survey. Artificial Intelligence Review. 2024. DOI: 10.1007/s10462-022-10147-y.

11. The scikit-image development team. Module skimage.metrics: structural_similarity (SSIM), peak_signal_noise_ratio (PSNR). scikit-image documentation. URL: https://scikit-image.org/docs/stable/api/skimage.metrics.html (дата звернення: 13.06.2026).

12. OpenCV team. Optical Flow: calcOpticalFlowFarneback. OpenCV documentation. URL: https://docs.opencv.org/4.x/d4/dee/tutorial_optical_flow.html (дата звернення: 13.06.2026).

13. Zhang R., Isola P., Efros A. A., Shechtman E., Wang O. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric (LPIPS). Proc. IEEE/CVF CVPR. 2018.

14. Blau Y., Michaeli T. The Perception-Distortion Tradeoff. Proc. IEEE/CVF CVPR. 2018. P. 6228-6237.

15. Topaz Labs. Topaz Video AI video enhancement software. URL: https://www.topazlabs.com/topaz-video (дата звернення: 13.06.2026).

16. k4yt3x. Video2x: a lossless video/GIF/image upscaler. GitHub. URL: https://github.com/k4yt3x/video2x (дата звернення: 13.06.2026).

17. Paszke A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems (NeurIPS). 2019.

18. Streamlit Inc. Streamlit documentation. URL: https://docs.streamlit.io (дата звернення: 13.06.2026).

19. FFmpeg project. FFmpeg documentation. URL: https://ffmpeg.org/documentation.html (дата звернення: 13.06.2026).

20. xinntao. Real-ESRGAN: practical algorithms for general image/video restoration. GitHub. URL: https://github.com/xinntao/Real-ESRGAN (дата звернення: 13.06.2026).

21. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide). Version 4.0 / ed. H. Washizaki. IEEE Computer Society, 2024. 411 p.

22. pytest project. pytest documentation. URL: https://docs.pytest.org (дата звернення: 13.06.2026).

23. GitHub Inc. GitHub Actions documentation. URL: https://docs.github.com/actions (дата звернення: 13.06.2026).

24. Желібо Є.П. Безпека життєдіяльності : підручник / В. В. Зацарний. Київ : Каравела, 2023. 344 с.

25. Жидецький В.Ц. Охорона праці користувачів комп’ютерів : підручник. Львів : Афіша, 2020. 176 с.

ДОДАТКИ

ДОДАТОК А

Тези конференції

IX Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 004.932.2:004.8
Духній В.–ст. гр. СП-41
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ ДЛЯ АПСКЕЙЛІНГУ ВІДЕОПОТОКУ З ВИКОРИСТАННЯМ МЕТОДІВ ГЛИБОКОГО НАВЧАННЯ ТА ЧАСОВОЇ УЗГОДЖЕНОСТІ КАДРІВ

Науковий керівник: к.т.н., доц. каф. ПІ Стоянов Ю. М.

Dukhnii V.
Ternopil Ivan Puluj National Technical University

DEVELOPMENT OF A SOFTWARE SYSTEM FOR VIDEO UPSCALING USING DEEP LEARNING AND TEMPORAL FRAME COHERENCE METHODS

Supervisor: PhD, Assoc. Prof. Stoyanov Yu. M.

Ключові слова: апскейлінг відео, глибоке навчання, Real-ESRGAN, часова узгодженість, обробка відеопотоку.
Keywords: video upscaling, deep learning, Real-ESRGAN, temporal coherence, video stream processing.

Вступ. Відео низької роздільної здатності є поширеною проблемою в архівній обробці, системах відеоспостереження та стримінгових платформах. На відміну від обробки окремих зображень, апскейлінг відеопотоку має специфічну складність: незалежна покадрова обробка породжує часові артефакти — мерехтіння, нестабільність текстур та непослідовність деталей між суміжними кадрами. Ці ефекти суттєво знижують суб'єктивну якість відтворення навіть при формально високій роздільній здатності кожного кадру [1].

Мета роботи – розробка програмної системи для підвищення роздільної здатності відеопотоку на основі методів глибокого навчання з урахуванням часової узгодженості між кадрами.

Основна частина. Розроблена система реалізує конвеєрну архітектуру обробки відеопотоку, яка складається з трьох основних компонентів: модуля зчитування відео, ядра попіксельної обробки кадрів та модуля кодування вихідного відео.

Модуль зчитування відео (video_reader) реалізований як генератор кадрів — він послідовно витягує кадри у форматі PIL-зображень та передає метадані потоку: частоту кадрів, загальну кількість кадрів, тривалість і роздільну здатність. Генераторний підхід дозволяє уникнути повного розпакування відео на диск, що критично для тривалих кліпів і обмежених ресурсів пам'яті.

Ядром системи підвищення роздільної здатності є модель Real-ESRGAN [2] на базі RRDB-генератора (Residual-in-Residual Dense Block) з підтримкою масштабування 2×, 4× та 8×. Для відео з людьми до конвеєру підключається модуль GFPGAN [3], який відновлює деталі обличчя з використанням генеративних пріорів StyleGAN2. Фреймворк PyTorch забезпечує інференс на трьох апаратних бекендах: CUDA, MPS та

CPU. Для відео великої роздільної здатності застосовується тайлінг — покадрова обробка по фрагментах з перекриттям із подальшим безшовним зшиванням.

Часова узгодженість між кадрами забезпечується на рівні постобробки: суміжні кадри аналізуються на наявність різких змін яскравості та структури текстур, після чого застосовується адаптивне змішування результатів. Це пригнічує мерехтіння, характерне для незалежної покадрової обробки, і забезпечує плавніший відеоряд без необхідності використання окремих мереж оптичного потоку.

Модуль запису відео (video_writer) кодує оброблені кадри у формат MP4 з кодеком H.264 засобами бібліотеки imageio-ffmpeg. На етапі фіналізації оригінальний аудіопотік мультиплексується у вихідний файл без перекодування, що зберігає якість звуку незалежно від коефіцієнта масштабування відеоряду. Передбачена коректна обробка тимчасових файлів при скасуванні або аварійному завершенні обробки.

Взаємодія з системою реалізована через графічний інтерфейс на базі Streamlit. Користувач може задати діапазон кадрів, крок пропуску кадрів, коефіцієнт масштабування, політику частоти кадрів на виході та перемикач збереження аудіо. Інтерфейс також відображає прогрес обробки з індикацією поточного кадру і загальної кількості, а при відсутності встановленого FFmpeg виводить діагностичне повідомлення з інструкцією зі встановлення.

Результати. Система забезпечує повний цикл обробки відеофайлів: від зчитування MP4-потоку до отримання MP4-виходу з підвищеною роздільною здатністю та збереженим аудіотреком. Генераторна потокова обробка дозволяє опрацьовувати відеокліпи довільної тривалості без зростання пікового споживання оперативної пам'яті. Застосування часової узгодженості між кадрами усуває мерехтіння, характерне для незалежної покадрової обробки.

Висновки. Розроблено програмну систему для апскейлінгу відеопотоку на основі глибокого навчання. Тонкошарова архітектура відео-адаптерів дозволила повністю перевикористати існуючий конвеєр обробки зображень без переписування моделей та фільтрів. Реалізована підтримка часової узгодженості кадрів підвищує суб'єктивну якість вихідного відео порівняно з незалежною покадровою обробкою. Подальший розвиток системи передбачає підтримку форматів MOV та MKV, а також додавання вихідного формату WebM.

Список використаних джерел:

1. Caballero J. et al. Real-Time Video Super-Resolution with Spatio-Temporal Networks and Motion Compensation // Proceedings of the IEEE CVPR. – 2017.
2. Wang X. et al. Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data // Proceedings of the IEEE/CVF ICCV Workshops. – 2021.
3. Wang X. et al. Towards Real-World Blind Face Restoration with Generative Facial Prior // Proceedings of the IEEE/CVF CVPR. – 2021.

ДОДАТОК Б

Посилання на репозиторій GitHub

https://github.com/fofanich/upscaler_mvp