

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ НЕЙРОННИХ
МЕРЕЖ ДЛЯ ЗБІЛЬШЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ ЗОБРАЖЕНЬ
RESEARCH AND SOFTWARE IMPLEMENTATION OF NEURAL NETWORKS
FOR IMAGE SUPER-RESOLUTION ENHANCEMENT

Виконав: студент 4 курсу, групи КНТ-113сп

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ШОСТАК А.К.

(ПРИЗВИЩЕ та ініціали)

Керівник СУББОТІН С.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент КОРОТКИЙ О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(КОД І НАЙМЕНУВАННЯ)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ШОСТАКА Андрія Костянтиновича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація нейронних мереж для збільшення роздільної здатності зображень. Research and Software Implementation of Neural Networks for Image Super-Resolution Enhancement

керівник проєкту (роботи) д.т.н., проф., СУББОТІН Сергій Олександрович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “07” квітня 2026 року № 139

2. Строк подання студентом проєкту (роботи) 10 червня 2026 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Опис програми. 3. Розробка архітектури програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СУББОТІН С.О., проф.		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання «20» квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Опис програми.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Експлуатація, тестування та експериментальне дослідження програми.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів. до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент

(підпис) Андрій ШОСТАК
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Сергій СУББОТІН
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 105 с., 3 табл., 29 рис., 2 дод., 34 джерела.

АНАЛІЗ ПРОДУКТИВНОСТІ, ГЛИБОКЕ НАВЧАННЯ, ЗБІЛЬШЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ, МЕТРИКИ ЯКОСТІ, НЕЙРОННА МЕРЕЖА, BENCHMARK, PYTORCH, SUPER-RESOLUTION.

Об'єкт дослідження – процес інженерії програмного забезпечення для інтелектуального опрацювання цифрових зображень.

Предмет дослідження – методи, алгоритми та програмні засоби архітектур нейронних мереж для збільшення роздільної здатності зображень.

Мета роботи – зменшення витрат часу та підвищення точності проведення порівняльного аналізу архітектур нейронних мереж шляхом розроблення програмного комплексу для автоматизованого тестування та візуалізації результатів збільшення роздільної здатності зображень.

Задачі дослідження:

- проаналізувати існуючі методи та архітектури нейронних мереж для збільшення роздільної здатності зображень;
- спроектувати модульну архітектуру програмного забезпечення для стандартизованого бенчмаркінгу;
- розробити програмні модулі для обробки даних, розрахунку метрик якості та створення графіків;
- здійснити тестування розробленого програмного комплексу та провести порівняльний аналіз обраних архітектур.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мова програмування Python, фреймворк глибокого навчання PyTorch, персональний комп'ютер з графічною картою NVIDIA RTX 4060 Ti 8GB під управлінням операційної системи Arch Linux.

Результати. Спроектовано та реалізовано програмний комплекс з розширюваною архітектурою, що дозволяє інтегрувати та порівнювати довільні моделі глибокого навчання через стандартизований інтерфейс.

Висновки. Створено систему, що забезпечує комплексне оцінювання нейромережових моделей збільшення роздільної здатності на основі математичних, перцептивних та апаратних метрик.

Галузь використання – розроблення, тестування та порівняльний аналіз моделей інтелектуальної обробки зображень у галузі комп’ютерного зору.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 105 pages, 3 tables, 29 figures, 2 appendixes, 34 sources.

BENCHMARK, DEEP LEARNING, IMAGE ENHANCEMENT, NEURAL NETWORK, PERFORMANCE ANALYSIS, PSNR, PYTORCH, QUALITY METRICS, SSIM, SUPER-RESOLUTION.

Object of study is the software engineering process for intelligent digital image processing.

Subject of study is methods, algorithms, and software tools of neural network architectures for image super-resolution.

Purpose of the work is the reduction of time costs and increasing the accuracy of comparative analysis of neural network architectures by developing a software package for automated testing and visualization of image super-resolution results.

Research tasks:

- to analyze existing methods and neural network architectures for image super-resolution;
- to design the modular software architecture for standardized benchmarking;
- to develop software modules for data processing, quality metric calculation, and plot generation;
- to perform testing of the developed software package and conduct a comparative analysis of the selected architectures.

Materials, methods, and tools: object-oriented programming, Python programming language, PyTorch deep learning framework, personal computer with NVIDIA RTX 4060 Ti 8GB graphics card running the Arch Linux operating system.

Results. A software package with an extensible architecture has been designed and implemented, allowing for the integration and comparison of arbitrary deep learning models through a standardized interface.

Conclusions. A system has been created that provides comprehensive evaluation of neural network models for super-resolution based on mathematical, perceptual, and hardware metrics.

Field of use is the development, testing, and comparative analysis of intelligent image processing models in the field of computer vision.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	10
Вступ.....	12
1 Аналіз предметної області.....	15
1.1 Огляд задачі збільшення роздільної здатності зображень.....	15
1.2 Аналіз архітектур нейронних мереж для обробки зображень.....	17
1.2.1 Вплив функцій втрат на якість генерації.....	17
1.2.2 Згорткові нейронні мережі (CNN).....	19
1.2.3 Генеративно-змагальні мережі (GAN).....	20
1.2.4 Трансформерні архітектури	22
1.2.5 Дифузійні моделі.....	24
1.3 Огляд методів та метрик оцінювання якості та наборів даних	25
1.4 Постановка завдань роботи.....	28
2 Опис програми.....	31
2.1 Схема проєктування структури системи	31
2.2 Схема функціонування системи	32
2.3 Обґрунтування вибору засобів розробки та технологій	34
2.4 Програмні рішення інтеграції моделей та оптимізації обчислень.....	37
2.5 Підсумки розділу та постановка завдань для наступного етапу	38
3 Розробка архітектури програми.....	40
3.1 Створення головного керуючого модуля та системи маршрутизації....	40
3.2 Розробка алгоритмів координації.....	41
3.3 Алгоритм тайлінгового інференсу для оптимізації використання пам'яті	43
3.4 Проєктування підсистем обробки графічних даних та розрахунку метрик	44
3.4.1 Процес підготовки та стандартизації вхідних даних	44
3.4.2 Алгоритми оцінювання якості	45

3.5	Механізми збору апаратної статистики та логування процесів	46
3.6	Створення інструментарію для візуалізації результатів та генерації звітів	48
3.7	Підготовка та налаштування нейромережових архітектур.....	49
3.7.1	Планування процесу тренування моделей	50
3.7.2	Оптимізація тензорних обчислень для споживчих графічних процесорів	51
3.8	Підсумки розділу та постановка завдань для наступного етапу.....	52
4	Експлуатація, тестування та експериментальне дослідження програми	53
4.1	Вимоги до програмно-апаратного середовища та експлуатація системи	53
4.2	Інструкція до встановлення та використання програми	58
4.3	Методика проведення експериментальних досліджень.....	60
4.4	Аналіз якості відновлення зображень за математичними та перцептивними метриками	63
4.5	Дослідження апаратної ефективності та швидкодії моделей.....	69
4.6	Оцінка балансу між якістю генерації та обчислювальною складністю	72
4.7	Підсумки розділу.....	76
	Висновки	78
	Перелік джерел посилання	80
	Додаток А Текст програми.....	84
	Додаток Б Слайди презентації	97

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CLIPQA	– CLIP-based Image Quality Assessment;
CNN	– Convolutional Neural Network;
CUDA	– Compute Unified Device Architecture;
ESRGAN	– Enhanced Super-Resolution GAN;
FLOPs	– Floating Point Operations per Second;
GAN	– Generative Adversarial Network;
GPU	– Graphics Processing Unit;
HAT	– Hybrid Attention Transformer;
HR	– High Resolution;
IQA	– Image Quality Assessment;
LPIPS	– Learned Perceptual Image Patch Similarity;
LR	– Low Resolution;
MACs	– Multiply-Accumulate Operations;
MSE	– Mean Squared Error;
MUSIQ	– Multi-scale Image Quality transformer;
PSNR	– Peak Signal-to-Noise Ratio;
RCAN	– Residual Channel Attention Network;
Real-ESRGAN	– Real-world ESRGAN;
ResShift	– Residual Shift;
SR	– Super-Resolution;
SRGAN	– Super-Resolution Generative Adversarial Network;
SRResNet	– Super-Resolution Residual Network;
SSIM	– Structural Similarity Index Measure;
SwinIR	– Swin Transformer for Image Restoration;
VGG	– Visual Geometry Group;
VRAM	– Video Random Access Memory;
ГП	– графічний процесор;

ОС	– операційна система;
ПЗ	– програмне забезпечення;
ЦП	– центральний процесор.

ВСТУП

На сучасному етапі розвитку інформаційних технологій візуальний контент становить основу світового обміну даними, що висуває безперервно зростаючі вимоги до якості та роздільної здатності цифрових зображень. Проблема збільшення роздільної здатності зображень (Super-Resolution, SR) є однією з найбільш фундаментальних та складних задач комп'ютерного зору, оскільки вона належить до класу некоректно поставлених обернених задач (ill-posed inverse problem). Суть проблеми полягає у необхідності відновлення втрачених високочастотних деталей на основі апріорно недостатньої інформації, що міститься у вихідному зображенні низької якості.

Актуальність цієї тематики у 2026 році підсилюється декількома глобальними трендами. З одного боку, спостерігається масовий перехід до стандартів надвисокої чіткості у побутовій електроніці та системах відеоспостереження. З іншого боку, у критичних галузях, таких як медична діагностика та дистанційне зондування Землі, фізична роздільна здатність сенсорів часто залишається обмеженою, що робить програмне покращення якості ключовим шляхом до отримання деталізованої інформації.

Традиційні методи математичної інтерполяції (білінійна [1], бікубічна [2]) є неефективними для сучасних вимог, оскільки вони не здатні реконструювати втрачені текстури та неминуче генерують артефакти розмиття. Натомість передовим стандартом стали методи глибокого навчання, які дозволяють нейронним мережам синтезувати відсутні деталі з високим ступенем вірогідності. Однак стрімкий розвиток нейромережових архітектур (від згорткових та генеративно-змагальних мереж до трансформерів і дифузійних моделей) створив нову проблему: відсутність уніфікованих засобів для їх об'єктивного порівняння.

Існуючі дослідження часто демонструють результати, отримані в різних умовах тестування, використовуючи застарілі метрики (наприклад, виключно PSNR [3]), які погано корелюють із людським візуальним

сприйняттям. Це суттєво ускладнює вибір оптимальної моделі для практичного впровадження. Саме тому розроблення універсального програмного комплексу для автоматизованого бенчмаркінгу планується як ефективне рішення цієї проблеми.

Об'єкт дослідження – процес інженерії програмного забезпечення для інтелектуального опрацювання цифрових зображень.

Предмет дослідження – методи, алгоритми та програмні засоби архітектур нейронних мереж для збільшення роздільної здатності зображень.

Мета роботи – зменшення витрат часу та підвищення точності проведення порівняльного аналізу архітектур нейронних мереж шляхом розроблення програмного комплексу для автоматизованого тестування та візуалізації результатів збільшення роздільної здатності зображень.

Для досягнення поставленої мети передбачається вирішити такі задачі дослідження:

- проаналізувати існуючі методи та архітектури нейронних мереж для збільшення роздільної здатності зображень;
- спроектувати модульну архітектуру програмного забезпечення для стандартизованого бенчмаркінгу;
- розробити програмні модулі для обробки даних, розрахунку метрик якості та створення графіків;
- здійснити тестування розробленого програмного комплексу та провести порівняльний аналіз обраних архітектур.

Методи дослідження. Для розв'язання поставлених задач планується використати методи об'єктно-орієнтованого проєктування програмного забезпечення, методи машинного та глибокого навчання для побудови конвеєрів обробки зображень, а також методи статистичного аналізу для обробки результатів обчислення математичних (PSNR [3], SSIM [4]) та перцептивних (LPIPS [5], CLIPQA [6], MUSIQ [7]) метрик.

Практичне значення отриманих результатів полягатиме у створенні функціонального програмного інструментарію (бенчмарку), який дозволить

дослідникам та інженерам автоматизувати процес тестування нейромереж. Завдяки модульній архітектурі, програмний комплекс забезпечить можливість легкої інтеграції нових моделей для визначення їхньої візуальної ефективності та апаратних вимог (споживання відеопам'яті, швидкодія).

Таким чином, логічним початком роботи є детальне вивчення теоретичного підґрунтя задачі, огляд існуючих математичних методів відновлення сигналів та аналіз передових нейромережеских архітектур, що дозволить сформулювати обґрунтовані вимоги до майбутнього програмного продукту. Це дослідження буде наведено у першому розділі пояснювальної записки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд задачі збільшення роздільної здатності зображень

Задача збільшення роздільної здатності зображень є однією з фундаментальних і найбільш досліджуваних проблем у галузі комп'ютерного зору та цифрової обробки сигналів. З математичної точки зору, процес отримання зображення низької роздільної здатності (LR) з оригінального зображення високої роздільної здатності (HR) можна формалізувати як процес деградації. Зазвичай цей процес моделюється рівнянням, що включає згортку з ядром розмиття (наприклад, гауссівським фільтром), операцію субдискретизації (даунсемплінгу) та додавання адитивного шуму. Основна мета алгоритмів Super-Resolution полягає в знаходженні функції зворотного відображення, яка здатна реконструювати зображення, що є максимально наближеним до оригінального, маючи на вході лише зменшене оригінальне зображення.

Однак, оскільки операції просторового розмиття та зменшення розмірності призводять до значної втрати високочастотних просторових компонентів, одному й тому ж зображенню низької якості може відповідати нескінченна множина потенційних високороздільних оригіналів. Отже, математичного розв'язку, який би гарантував однозначне і точне відновлення вихідного сигналу, не існує. Саме тому задача SR класифікується як некоректно поставлена обернена задача (ill-posed inverse problem).

Історично для вирішення цієї проблеми застосовувалися традиційні методи просторової інтерполяції, серед яких найбільш поширеними є інтерполяція найближчого сусіда, білінійна [1] та бікубічна [2] інтерполяції. Механізм дії цих алгоритмів базується виключно на локальних математичних апроксимаціях: значення нових пікселів розраховуються як зважена сума значень сусідніх пікселів вихідного зображення за допомогою поліноміальних функцій. Відповідно, традиційні методи здатні лише перерозподіляти наявну енергію сигналу та виконувати згладжування переходів між пікселями. Вони

апріорі не можуть додати жодної нової інформації або відновити втрачені текстури, через що результати інтерполяції характеризуються вираженим розмиттям, наявністю артефактів "драбинки" (aliasing) та ефектом ореолу (ringing) навколо контрастних меж.

У зв'язку з цим, критично важливо розуміти фундаментальні обмеження процесу реконструкції з позицій теорії інформації. Необхідно наголосити, що інформація, втрачена під час даунсемплінгу або стиснення із втратами, зникає безповоротно. У масовій культурі та кінематографі глибоко вкорінився міф про можливість магічного "покращення" зображень, коли з невеликої групи розмитих пікселів програма з абсолютною точністю відновлює номерні знаки автомобіля або детальні риси обличчя підозрюваного. З наукової точки зору такі сценарії є фізично та математично неможливими. Жоден алгоритм не здатен детерміновано витягти специфічні ідентифікаційні дані, якщо вони були знищені на етапі квантування або субдискретизації сенсором камери.

Сучасні підходи на основі машинного та глибокого навчання докорінно змінили парадигму вирішення задачі SR, проте вони не порушують законів теорії інформації. Нейронні мережі не "відновлюють" справжню історичну реальність, яка була перед об'єктивом у момент зйомки. Замість цього вони "галюцинують" (синтезують) високочастотні деталі. Під час тривалого процесу навчання на колосальних масивах парних зображень (LR та HR), нейромережі засвоюють складні статистичні закономірності візуального світу (природні апріорні знання, або natural image priors). Отримуючи на вхід розмитий контур, модель генерує ту текстуру (наприклад, шерсть тварини, листя дерева чи пори шкіри), яка є статистично найбільш вірогідною для даного семантичного контексту. Таким чином, синтезоване зображення виглядає фотореалістичним і переконливим для людської зорової системи, але згенеровані мікродеталі є результатом імовірнісної апроксимації, а не точним відтворенням первинного об'єкта.

Для забезпечення такого складного нелінійного відображення між просторами низької та високої роздільної здатності необхідні потужні архітектурні рішення. Ефективність синтезу вірогідних текстур прямо залежить від топології моделі, її здатності захоплювати глобальний контекст та обраних оптимізаційних критеріїв. Відповідно, логічним кроком для подальшого дослідження є детальний розгляд еволюції та принципів функціонування різноманітних сімейств нейронних мереж, що застосовуються для інтелектуальної обробки зображень.

1.2 Аналіз архітектур нейронних мереж для обробки зображень

Еволюція методів глибокого навчання у задачі збільшення роздільної здатності зображень пройшла шлях від простих неглибоких згорткових мереж до складних ієрархічних систем, що базуються на механізмах уваги та дифузійних процесах.

1.2.1 Вплив функцій втрат на якість генерації

Процес навчання нейронних мереж для відновлення зображень полягає у мінімізації функції втрат, яка визначає розбіжність між реконструйованим зображенням та еталоном високої роздільної здатності. Вибір конкретної функції втрат безпосередньо впливає на візуальні характеристики вихідного контенту, оскільки різні математичні критерії по-різному інтерпретують помилки відновлення.

Традиційним підходом у задачах збільшення роздільної здатності є використання функції втрат L_2 (середньоквадратична помилка, MSE). Встановлено, що мінімізація L_2 еквівалентна максимізації пікового відношення сигналу до шуму (PSNR), що робить її зручною для досягнення високих значень цієї метрики. Проте, згідно з дослідженням [8], використання L_2 часто призводить до надмірного розмиття (blurring) високочастотних

деталей. Це пояснюється тим, що функція L_2 сильніше штрафує великі відхилення, ніж малі, що стимулює модель генерувати статистично "середні" значення пікселів, фактично усереднюючи всі можливі варіанти відновлення текстур.

Альтернативою є функція втрат L_1 (середня абсолютна помилка, MAE), яка демонструє кращі результати при відновленні чітких країв та дрібних деталей. На відміну від L_2 , функція L_1 не так агресивно реагує на викиди, що дозволяє нейронній мережі краще зберігати контрастність зображення. Дослідним шляхом встановлено, що моделі, навчені за допомогою L_1 , забезпечують вищу суб'єктивну якість генерації порівняно з моделями на базі L_2 за однакових архітектурних умов.

Окрему категорію становлять структурні функції втрат, що базуються на індексі структурної схожості (SSIM) та його багатомасштабній версії (MS-SSIM). Ці функції враховують особливості людської зорової системи, зокрема сприйняття яскравості, контрасту та структури об'єктів. Дослідження [8] доводить, що MS-SSIM дозволяє ефективніше зберігати високочастотні компоненти у темних та контрастних зонах зображення, де класичні попиксельні метрики (L_1 , L_2) виявляються малоефективними.

Перцептивні функції втрати (Perceptual Losses), що базуються на порівнянні ознак у просторі глибоких нейронних мереж (наприклад, VGG), дозволяють моделі фокусуватися на семантичному змісті зображення, а не на точному збігу значень пікселів. Це критично важливо для генеративно-змагальних мереж (GAN), де метою є досягнення фотореалістичності. Проте часто найкращі результати досягаються шляхом комбінування декількох функцій втрат, наприклад, поєднання L_1 для забезпечення базової точності та MS-SSIM або перцептивної втрати для відновлення текстурної достовірності.

1.2.2 Згорткові нейронні мережі (CNN)

Згорткові нейронні мережі (CNN) стали фундаментом для розвитку сучасних методів збільшення роздільної здатності зображень завдяки їхній здатності автоматично видобувати ієрархічні просторові ознаки та моделювати складні нелінійні відображення між просторами низької та високої роздільної здатності.

Першою успішною спробою застосування глибокого навчання для вирішення задачі SR стала архітектура SRCNN [9]. Дана модель концептуально довела, що традиційні методи на основі розрідженого кодування можуть бути представлені у вигляді глибокої згорткової мережі. Архітектура SRCNN складається з трьох ключових етапів: екстракції та представлення патчів, нелінійного відображення та фінальної реконструкції. Незважаючи на відносно малу глибину (три шари), встановлено, що SRCNN суттєво перевершує класичні методи інтерполяції за показниками PSNR та SSIM.

Подальший розвиток CNN у галузі SR пов'язаний із впровадженням концепції залишкового навчання (residual learning). Встановлено, що використання залишкових зв'язків (skip-connections) дозволяє будувати значно глибші мережі, вирішуючи проблему згасання градієнтів та полегшуючи процес оптимізації. У задачах Super-Resolution це дозволяє мережі фокусуватися на вивченні лише високочастотних деталей (залишку між LR та HR зображеннями), а не на відновленні всього масиву пікселів [10].

Для забезпечення стабільної збіжності глибоких архітектур критично важливим є вибір функцій активації та методів ініціалізації ваг. Зокрема, використання параметричного випрямляча (PReLU) та методу ініціалізації, адаптованого для нелінійних активацій, дозволяє досягати вищої точності порівняно з традиційною функцією ReLU та ініціалізацією Ксав'є [11].

Важливим аспектом проєктування CNN є використання шарів пакетної нормалізації (Batch Normalization). Хоча Batch Normalization сприяє

прискоренню навчання шляхом зменшення внутрішнього коваріатного зсуву, у пізніших дослідженнях SR було встановлено, що видалення цих шарів може покращити гнучкість мережі щодо діапазону значень пікселів та зменшити кількість артефактів [12].

Еволюція залишкових архітектур призвела до появи моделі RCAN [13]. У цій архітектурі запропоновано механізм каналної уваги (Channel Attention) у поєднанні з дуже глибокими структурами типу "залишок у залишку" (residual in residual). Встановлено, що такий підхід дозволяє мережі адаптивно перерозподіляти вагу між різними каналами ознак, фокусуючись на найбільш інформативних високочастотних компонентах, що забезпечує значне покращення деталізації.

На фінальному етапі реконструкції в CNN часто застосовуються шари транспонованої згортки (deconvolution) або субпіксельної згортки для збільшення розмірності карт ознак. Слід зазначити, що при використанні транспонованої згортки існує ризик виникнення артефактів "шахової дошки" (checkerboard artifacts), спричинених нерівномірним перекриттям ядер згортки. Для запобігання таким дефектам обґрунтовано використання операції Resize-Convolution, де збільшення розмірності виконується за допомогою інтерполяції з наступною стандартною згорткою [14].

Таким чином, згорткові нейронні мережі забезпечують високу математичну точність відновлення контурів, проте їхня ефективність безпосередньо залежить від глибини архітектури та обраної стратегії нормалізації й активації.

1.2.3 Генеративно-змагальні мережі (GAN)

Генеративно-змагальні мережі (Generative Adversarial Networks) стали фундаментальним проривом у вирішенні задачі відновлення текстур та дрібних деталей, які зазвичай втрачаються при використанні класичних згорткових архітектур, орієнтованих на мінімізацію середньоквадратичної

помилки. Основна концепція GAN базується на змаганні двох мереж: генератора (G), який намагається створити зображення високої роздільної здатності, та дискримінатора (D), завданням якого є відрізнити згенерованих даних від реальних еталонних зразків.

Першою успішною реалізацією цього підходу в галузі Super-Resolution стала архітектура SRGAN [15]. У даній моделі було вперше запропоновано використання перцептивної функції втрат (perceptual loss), яка обчислюється не в просторі пікселів, а у просторі ознак глибокої мережі VGG. Це дозволило змістити фокус із чистої математичної точності (PSNR) у бік візуальної фотореалістичності. Генератор SRGAN використовує залишкові блоки (Residual Blocks), що дозволяє будувати глибші архітектури без втрати градієнта.

Подальшим розвитком технології стала мережа ESRGAN (Enhanced SRGAN) [16], яка впроваджує декілька ключових вдосконалень:

- вилучення шарів пакетної нормалізації (Batch Normalization), що дозволило зменшити кількість обчислювальних артефактів та підвищити стабільність навчання;
- впровадження блоку RRDB (Residual-in-Residual Dense Block), який поєднує залишкові зв'язки з щільними блоками, забезпечуючи вищу пропускну здатність ознак;
- використання релятивістського дискримінатора (Relativistic Discriminator), який оцінює не просто ймовірність реальності зображення, а те, наскільки воно "більш реальне", ніж згенероване.

Для вирішення задач відновлення зображень у реальних умовах ("Blind SR"), де параметри деградації невідомі, було розроблено архітектуру Real-ESRGAN [17]. Особливістю даної моделі є використання складного процесу синтезу даних (higher-order degradation model), що імітує багаторазове розмиття, шум та стиснення. У Real-ESRGAN застосовано дискримінатор на базі архітектури U-Net [18] зі спектральною нормалізацією. Використання структури U-Net у дискримінаторі дозволяє мережі аналізувати зображення на

різних масштабах, що є критично важливим для відновлення складних міських текстур та усунення артефактів на межах об'єктів.

Досліджено, що застосування GAN-архітектур у програмному комплексі SRBenchmark дозволяє досягати значно вищих показників за перцептивними метриками (LPIPS, CLIPQA), попри певне зниження показників PSNR. Це підтверджує ефективність змагального навчання для задач, де пріоритетом є суб'єктивна візуальна якість та природність відновлених текстур.

1.2.4 Трансформерні архітектури

Поява архітектури Transformer, яка спочатку здійснила революцію в обробці природної мови, стимулювала її адаптацію для задач комп'ютерного зору. Фундаментальним кроком стала модель Vision Transformer (ViT), яка довела, що зображення можна обробляти як послідовність патчів (фрагментів), аналогічно до слів у тексті [19]. Замість використання локальних згорткових фільтрів, які обмежені невеликим рецептивним полем, ViT застосовує механізм глобального внутрішнього зосередження уваги (Self-Attention). Це дозволяє моделювати довготривалі залежності між будь-якими просторовими ділянками зображення з перших шарів мережі.

Проте застосування стандартного механізму глобальної уваги до задач збільшення роздільної здатності стикається з проблемою квадратичної обчислювальної складності відносно просторового розміру зображення. Для вирішення цієї проблеми було розроблено ієрархічний трансформер зі зсунутими вікнами – Swin Transformer [20]. Основна ідея методу полягає в обчисленні уваги локально в межах вікон, що не перетинаються, зі зсувом цих вікон на наступних шарах. Застосування такого підходу забезпечило лінійну обчислювальну складність та дозволило ефективно обробляти зображення високої роздільної здатності.

Успіх архітектури Swin у задачах класифікації та сегментації був адаптований для відновлення зображень у моделі SwinIR (Swin Transformer for Image Restoration) [21]. Досліджено, що SwinIR складається з модулів неглибокого виділення ознак, глибокого виділення ознак (на основі залишкових блоків Swin Transformer – RSTB) та модуля високоякісної реконструкції. Використання механізму віконної уваги дозволяє моделі ефективно враховувати локальні текстури, тоді як глобальні структурні зв'язки підтримуються завдяки залишковим з'єднанням між блоками. Встановлено, що такий підхід забезпечує значну перевагу над класичними згортковими мережами за повноеталонними метриками PSNR та SSIM.

Незважаючи на ефективність віконного механізму уваги, подальші дослідження виявили його структурні обмеження у задачах реставрації контенту. Встановлено, що використання непересічних вікон суттєво звужує фактичне рецептивне поле мережі, не дозволяючи активувати достатню кількість пікселів для формування повноцінного глобального контексту [22]. Це призводить до втрати частини корисної інформації з сусідніх ділянок зображення, яка могла б бути використана для якіснішої реконструкції високочастотних деталей та складних текстур.

Для подолання вказаних обмежень було розроблено гібридний трансформер уваги (HAT) [23]. Ця архітектура інтегрує механізм каналної уваги (Channel Attention) у класичний віконний трансформер, що дозволяє моделі глобально агрегувати інформацію вздовж каналів і краще використовувати міжпіксельні взаємозв'язки. Крім того, в HAT було впроваджено блоки перехресної уваги з перекриттям (Overlapping Cross-Attention Blocks, OCAB), які забезпечують пряму взаємодію ознак між сусідніми вікнами. Обґрунтовано, що така гібридна конструкція дозволяє значно розширити рецептивне поле моделі та використати ширший візуальний контекст для синтезу зображень.

В межах даної роботи та розробленого програмного комплексу SRBenchmark сімейство трансформерних архітектур представлено моделями

SwinIR та HAT. Їхня програмна реалізація та інтеграція в стандартизоване середовище тестування дозволяє на практиці дослідити компроміс між надвисокою математичною точністю реконструкції та значними обчислювальними витратами. Як свідчать результати експериментальних досліджень, хоча механізми уваги забезпечують лідерство за метриками структурної схожості, вони висувають жорсткі вимоги до обсягу відеопам'яті (VRAM) та кількості операцій з рухомою комою (FLOPs) у порівнянні з класичними згортковими рішеннями.

1.2.5 Дифузійні моделі

Протягом останніх двох років у галузі генеративного моделювання та розв'язання обернених задач комп'ютерного зору спостерігається зміна парадигми від генеративно-змагальних мереж до дифузійних імовірнісних моделей (Diffusion Probabilistic Models). Теоретичний фундамент даного класу нейронних мереж базується на принципах нерівноважної термодинаміки, що вперше було сформульовано у 2015 році [24]. Основна ідея полягає у моделюванні процесу поступового руйнування структури даних шляхом додавання гауссівського шуму (прямий процес) та подальшому навчанні нейронної мережі відновлювати вихідний сигнал із зашумленого стану (зворотний процес).

Ключовим етапом розвитку технології стала розробка алгоритму Denoising Diffusion Probabilistic Models (DDPM) [25], де було запропоновано використовувати марковські ланцюги для поступового видалення шуму. На відміну від GAN, дифузійні моделі забезпечують вищу стабільність навчання та дозволяють уникнути проблеми колапсу мод (mode collapse), що забезпечує більшу варіативність та фотореалістичність згенерованих текстур. Проте покрокова природа DDPM зумовлює низьку швидкість інференсу, що було частково вирішено в архітектурі Denoising Diffusion Implicit Models (DDIM) [26] через впровадження недетермінованого прискореного семплювання.

Для подолання проблеми високої обчислювальної складності при роботі з зображеннями високої роздільної здатності було розроблено концепцію латентних дифузійних моделей (Latent Diffusion Models, LDM) [27]-[28]. У даному підході дифузійний процес переноситься з простору пікселів у стиснений латентний простір, сформований попередньо навченим автоенкодером (VQGAN). Це дозволяє моделі оперувати семантичними ознаками зображення, що радикально зменшує вимоги до обсягу відеопам'яті та часу обробки, зберігаючи при цьому здатність до відновлення дрібних деталей.

Особливу увагу в контексті задачі збільшення роздільної здатності приділено архітектурі ResShift [29], яка інтегрована до розробленого програмного комплексу SRBenchmark. Модель базується на механізмі залишкового зсунення (residual shifting), що дозволяє ефективно пов'язувати зображення низької роздільної здатності з цільовим станом високої якості. На відміну від стандартних дифузійних моделей, ResShift потребує значно меншої кількості кроків денойзингу (від 4 до 15), що робить її однією з найбільш продуктивних дифузійних архітектур для SR-задач станом на 2026 рік. У межах дослідження встановлено, що використання дифузійного підходу дозволяє досягти найвищих показників за перцептивними метриками (CLIP-IQA [6], MUSIQ [7]), забезпечуючи синтез природних текстур, які важко відрізнити від оригінальних фотографій.

1.3 Огляд методів та метрик оцінювання якості та наборів даних

Проблема об'єктивного оцінювання результатів інтелектуального збільшення роздільної здатності зображень залишається однією з найбільш дискусійних у сучасному комп'ютерному зорі. Складність полягає у відсутності універсального математичного дескриптора, який міг би повною мірою апроксимувати людське сприйняття візуальної інформації. Слід зауважити, що кількісні метрики використовуються переважно для

формалізованого порівняння ефективності архітектур у межах наукових досліджень та публікацій. Однак на практиці фінальним та найбільш значущим критерієм якості залишається суб'єктивне оцінювання, що базується на особливостях людської зорової системи (Human Visual System, HVS). Оскільки метою задачі SR часто є отримання візуально привабливого контенту, математична близькість до еталона не завжди корелює з реальною чіткістю та деталізацією згенерованого кадру.

Традиційно для оцінювання моделей SR застосовують класичні повноеталонні метрики (Full-Reference IQA), які базуються на безпосередньому порівнянні відновленого зображення з оригіналом високої якості (Ground Truth). Найбільш розповсюдженими серед них є пікове відношення сигналу до шуму (PSNR [3]) та індекс структурної схожості (SSIM [4]).

Метрика PSNR обчислюється через логарифмічне відношення максимально можливої потужності сигналу до потужності шуму, що вноситься процесом деградації та подальшого відновлення. Математично вона базується на середньоквадратичній помилці (MSE). Проаналізовано, що PSNR є ефективним індикатором при мінімізації попиксельних розбіжностей, що робить її ідеальною для оцінювання моделей, навчених за допомогою функцій втрат L1 або L2. Проте в задачі SR ця метрика має фундаментальний недолік: вона стимулює отримання усереднених, розмитих результатів. Моделі, орієнтовані на PSNR, генерують так зване "мило", оскільки усереднення значень пікселів дозволяє досягти меншої статистичної помилки, ніж спроба синтезувати чітку текстуру, яка може не збігатися з оригіналом на піксельному рівні.

Індекс структурної схожості (SSIM) є більш просунутим підходом, оскільки він враховує не лише інтенсивність сигналу, а й зміну яскравості, контрасту та структури об'єктів. SSIM розглядає зображення як сукупність структурних елементів, що краще відповідає особливостям HVS, ніж попиксельне порівняння. Проте обидві метрики PSNR та SSIM демонструють

неспроможність адекватно оцінювати генеративно-змагальні мережі (GAN). Слід підкреслити, що GAN-моделі часто отримують низькі значення PSNR/SSIM через незначні просторові зміщення згенерованих текстур, хоча візуально вони виглядають значно чіткішими та реалістичнішими за еталон.

Для подолання описаного розриву між математичною точністю та візуальною якістю було розроблено перцептивні метрики. Однією з найбільш репрезентативних є LPIPS [5]. Дана метрика обчислює відстань між зображеннями у просторі глибоких ознак нейронних мереж (наприклад, VGG або AlexNet). Оскільки глибокі шари мереж виділяють семантичні ознаки, LPIPS дозволяє оцінювати схожість текстур і форм так, як це робить людина, ігноруючи дрібні попиксельні відхилення.

Окрему категорію складають методи оцінювання без використання еталонного зображення (No-Reference IQA), що є критично важливим для реальних сценаріїв використання SR, де оригінал високої якості відсутній. Сучасні алгоритми, такі як CLIPQA [6] та MUSIQ [7], базуються на глибокому вивченні ознак природних зображень. CLIPQA використовує потужний візуально-мовний простір моделі CLIP для оцінки природності та естетичності зображення. У свою чергу, MUSIQ використовує трансформерну архітектуру для аналізу зображення на різних масштабах, що дозволяє враховувати як глобальну композицію, так і локальні мікротекстури. Обґрунтовано, що використання лише однієї групи метрик не дозволяє отримати повної картини продуктивності моделі. Тому в сучасних дослідженнях необхідним є комплексний бенчмаркінг, який поєднує класичні (PSNR, SSIM), перцептивні (LPIPS) та безреференсні (CLIPQA, MUSIQ) підходи для всебічного аналізу якості генерації.

Якість навчання та достовірність тестування моделей Super-Resolution безпосередньо залежать від обраних наборів даних. У сучасній індустрії стандартом для навчання є набори з великою кількістю варіативних сцен та високою деталізацією:

- DIV2K [30] – містить 1000 зображень роздільної здатності 2K, що характеризуються різноманітністю об'єктів та текстур;
- DF2K [31] – комбінований набір, що об'єднує DIV2K та Flickr2K, забезпечуючи ще більшу кількість прикладів для навчання глибоких архітектур;
- ImageNet [32] – широкомасштабна база даних, що використовується для попереднього навчання моделей для покращення їхньої здатності до узагальнення.

Для етапу тестування та валідації (testing) зазвичай використовується набір класичних датасетів, кожен з яких спрямований на перевірку специфічних характеристик моделі:

- Set5 та Set14 [33] – невеликі набори з популярними тестовими зображеннями для базової оцінки;
- BSDS100 [33] – вибірка з набору Berkeley Segmentation Dataset, що містить різноманітні природні сцени;
- Urban100 [33] – спеціалізований набір, що складається зі складних міських архітектурних структур, де критично важливим є збереження прямих ліній та повторюваних візерунків;
- Manga109 [34] – набір японських коміксів, що використовується для оцінювання здатності моделі відновлювати чіткі лінії, текст та специфічні графічні елементи.

1.4 Постановка завдань роботи

На основі проведеного аналізу сучасного стану технологій інтелектуальної обробки сигналів та виявлених проблем, таких як методологічна розрізненість існуючих нейромережевих архітектур, складність їх об'єктивного та всебічного порівняння, а також гостра потреба у знаходженні оптимального балансу між візуальною якістю генерації та реальними апаратними витратами, виникає необхідність у створенні

спеціалізованого програмного середовища для бенчмаркінгу. Традиційні підходи до оцінювання моделей збільшення роздільної здатності часто виявляються обмеженими через використання розрізнених скриптів та ігнорування показників обчислювальної складності, що суттєво ускладнює вибір конкретного рішення для практичного впровадження.

Таким чином, актуальність дослідження зумовлена потребою у проєктуванні єдиного інструментарію, здатного забезпечити прозоре та автоматизоване порівняння різнорідних моделей глибокого навчання.

Мета роботи – зменшення витрат часу та підвищення точності проведення порівняльного аналізу архітектур нейронних мереж шляхом розроблення концепції та проєктування універсального програмного комплексу для автоматизованого тестування, профілювання апаратних витрат та візуалізації результатів збільшення роздільної здатності зображень.

Для досягнення поставленої мети планується вирішити такі конкретні завдання:

- проаналізувати предметну область, методи, алгоритми та існуючі підходи до архітектур нейронних мереж для збільшення роздільної здатності зображень;
- спроектувати модульну структуру програмного забезпечення, що забезпечить можливість легкого інтегрування та стандартизованого порівняння різних сімейств моделей, включаючи згорткові мережі, генеративно-змагальні мережі, трансформери та дифузійні моделі;
- продумати логіку та алгоритми майбутніх модулів для автоматизованої обробки наборів даних, розрахунку класичних і сучасних перцептивних метрик якості зображення, а також інструментарію для збору детальної статистики споживання апаратних ресурсів;
- спланувати та провести комплексне тестування майбутнього програмного забезпечення, здійснити порівняльний аналіз продуктивності та якості роботи обраних архітектур нейронних мереж у межах створеного бенчмарку та сформулювати обґрунтовані висновки.

У першому розділі було розглянуто фундаментальні аспекти проблеми збільшення роздільної здатності зображень та проаналізовано еволюцію методів глибокого навчання: від класичних згорткових мереж до сучасних дифузійних моделей та трансформерів. Встановлено, що ключовою проблемою сучасної галузі комп'ютерного зору є відсутність стандартизованих програмних рішень для комплексного оцінювання нейромереж, які б враховували не лише математичну точність (PSNR, SSIM), а й перцептивну якість (LPIPS, CLIPQA) та апаратну ефективність.

З огляду на сформульовану мету та конкретизовані завдання, логічним продовженням дослідження є перехід до етапу проєктування архітектури майбутнього програмного забезпечення. У наступному розділі буде детально розібрано схему структури та функціонування комплексу, обґрунтовано вибір мови програмування та визначено ключові технічні рішення, необхідні для успішної реалізації поставлених завдань.

2 ОПИС ПРОГРАМИ

2.1 Схема проєктування структури системи

Для ефективного розв'язання задачі автоматизованого тестування нейронних мереж структуру майбутнього програмного забезпечення планується побудувати за модульним принципом. Такий підхід забезпечить чіткий розподіл відповідальності між компонентами та ізоляцію важких обчислювальних процесів від логіки відображення та збереження результатів. Згідно із запропонованою концепцією, архітектура системи буде відкритою для розширення, але закритою для модифікації свого базового ядра.

Відповідно до розробленої схеми проєктування, життєвий цикл програми розпочинатиметься із запуску та послідовно проходитиме через наступні логічні модулі (рисунок 2.1):

- модуль обробки параметрів запуску – відповідатиме за прийом вхідних даних від користувача (введення з командного рядка або зчитування конфігураційних файлів), їх валідацію та передачу до центрального вузла;
- модуль координатор – виступатиме головним маршрутизатором системи, який ініціалізуватиме необхідні робочі класи та визначатиме загальний вектор роботи (режим тестування, одиничного збільшення або візуального порівняння);
- модуль опрацювання зображень використовуючи нейронні мережі та збору статистичних даних – спеціалізоване середовище, де планується реалізувати алгоритми поетапного завантаження графічних файлів, виконання прямого проходження тензорів через моделі штучного інтелекту, а також паралельну фіксацію апаратних метрик і показників візуальної якості;
- модуль опрацювання статистичних даних, створення і збереження зображень на диск – фінальний етап, на якому отримані сирі масиви інформації перетворюватимуться у репрезентативні звіти, будуватимуться графіки залежностей та формуватимуться порівняльні колажі перед остаточним завершенням програми.

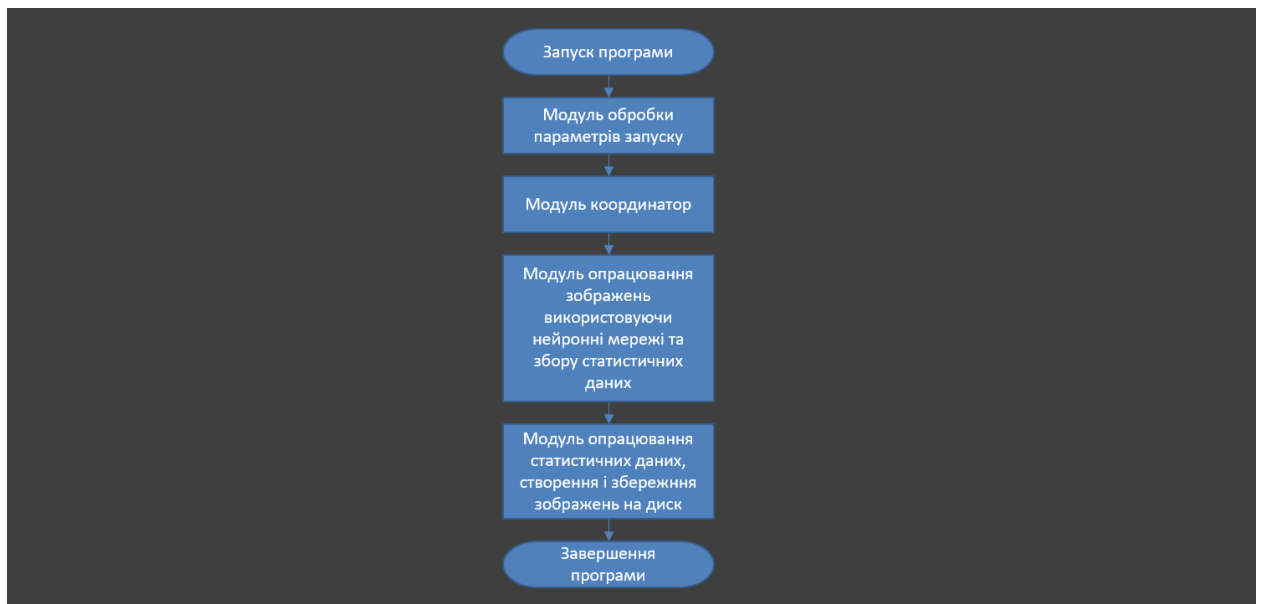


Рисунок 2.1 – Схема проєктування програмного забезпечення

Запропонована схема проєктування чітко окреслює модульну структуру майбутнього програмного комплексу, що гарантує його гнучкість та здатність до безперешкодної інтеграції нових нейромережевих архітектур у майбутньому. Визначивши статичну структуру системи та її основні будівельні блоки, логічним наступним кроком є розгляд динамічних аспектів її роботи, а саме – детальної схеми функціонування, яка опише алгоритмічну послідовність взаємодії цих модулів під час генерації зображень та збору аналітики.

2.2 Схема функціонування системи

Після визначення модульної структури наступним етапом є розроблення динамічної моделі поведінки програмного забезпечення. Схема функціонування системи (рисунок 2.2) визначатиме алгоритмічну послідовність дій від моменту запуску до формування кінцевого результату залежно від обраного користувачем сценарію.

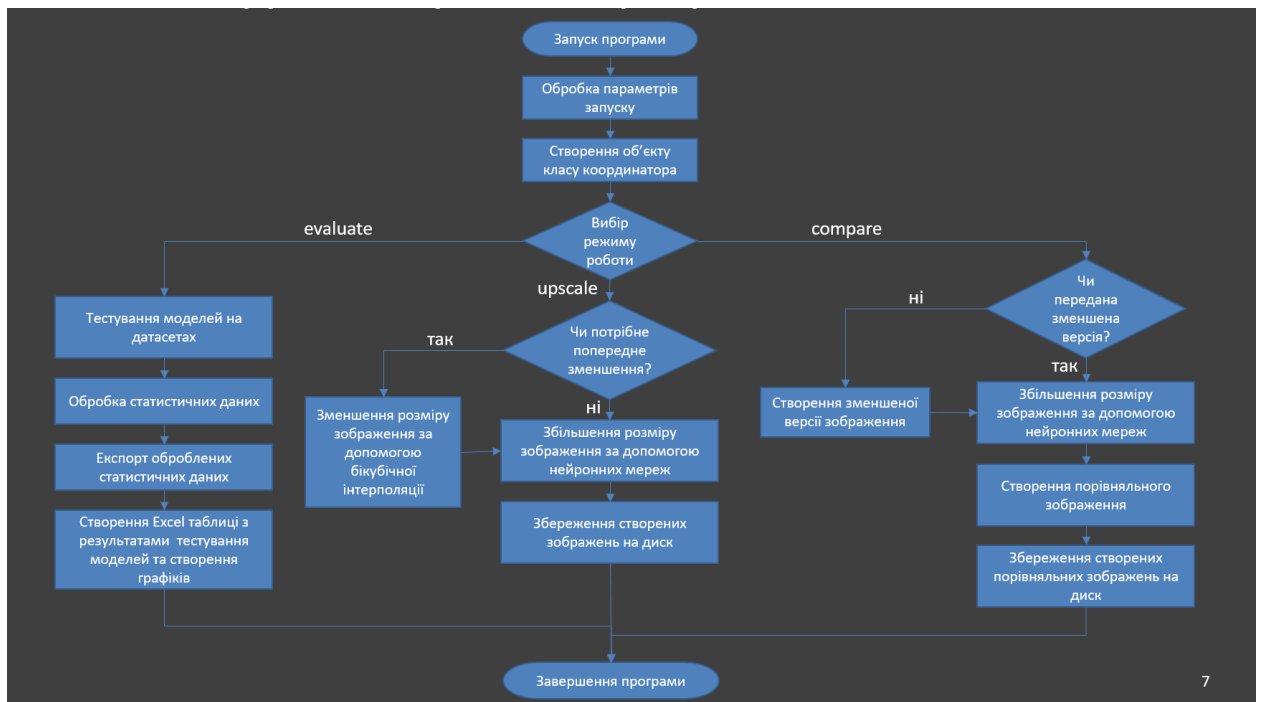


Рисунок 2.2 – Схема функціонування програмного забезпечення

Передбачається, що життєвий цикл програми розпочинатиметься із загального етапу запуску та обробки вхідних параметрів. Після їх успішної валідації відбудуватиметься створення об'єкта класу координатора, який перебиратиме на себе подальше управління виконанням. На цьому етапі алгоритм передбачає розгалуження на три незалежні режими роботи: тестування (evaluate), збільшення (upscale) та порівняння (compare).

У режимі тестування (evaluate) послідовність дій буде наступною:

- ініціалізація та тестування обраних моделей на заданих наборах даних (датасетах);
- збір та опрацювання статистичних даних, що включатимуть показники якості та апаратних витрат;
- експорт оброблених статистичних даних;
- автоматичне створення Excel таблиці з результатами тестування моделей та генерація відповідних графіків.

У режимі збільшення (upscale) алгоритм працюватиме з зображеннями:

- система виконуватиме перевірку: чи потрібне попереднє зменшення вхідного зображення;

- якщо так, відбуватиметься зменшення розміру зображення за допомогою бікубічної інтерполяції;
- далі ініціюватиметься безпосереднє збільшення розміру зображення за допомогою нейронних мереж;
- збереження створених зображень на диск.

У режимі порівняння (compare) процес функціонуватиме за схожим принципом, але з фокусом на візуальне зіставлення:

- перевірка: чи передана зменшена версія зображення користувачем;
- якщо зменшена версія відсутня, система автоматично ініціюватиме створення зменшеної версії зображення;
- збільшення розміру зображення за допомогою набору обраних нейронних мереж;
- створення зведеного порівняльного зображення (колажу) для наочної оцінки;
- збереження створених порівняльних зображень на диск.

Розроблена схема функціонування чітко регламентує поведінку системи у різних сценаріях використання, гарантуючи передбачуваність та стабільність обчислювального процесу. Відпрацьована логіка маршрутизації дозволить ефективно керувати як масовим тестуванням, так і роботою з одиничними файлами. Для того щоб успішно реалізувати такий складний алгоритмічний конвеєр та забезпечити високопродуктивну взаємодію з нейромережевими моделями, необхідно ретельно підійти до вибору технологічного стека. Відповідно, наступним кроком стане обґрунтування вибору мови програмування та супутніх інструментів розробки.

2.3 Обґрунтування вибору засобів розробки та технологій

Процес розроблення програмного комплексу для бенчмаркінгу нейромережових архітектур потребуватиме ретельного підбору технологічного інструментарію, який би забезпечував баланс між високою

швидкістю обчислень, гнучкістю архітектури та безпекою обробки даних. Основним середовищем розроблення планується обрати мову програмування Python. Цей вибір продиктований статусом мови як абсолютного стандарту для задач комп'ютерного зору. Для прийняття остаточного та об'єктивного рішення було здійснено порівняльний аналіз екосистем Python, R та MATLAB за ключовими для проєкту критеріями, результати якого наведено у таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз мов програмування

Критерій порівняння	Python	R	MATLAB
Наявність сучасних фреймворків для створення нейронних мереж	Так	Ні	Ні
Рівень підтримки апаратного прискорення GPU (CUDA)	Високий	Низький	Середній
Наявність готових SOTA архітектур	Так	Ні	Ні
Ефективність інструментів обробки зображень	Висока	Низька	Висока
Відкритий вихідний код	Так	Так	Ні

Аналіз даних таблиці 2.1 беззаперечно доводить, що Python є єдиним доцільним вибором для реалізації поставлених завдань, оскільки лише він повною мірою задовольняє вимоги щодо нативної роботи з глибокими нейронними мережами та апаратним прискоренням.

Фундаментальним обчислювальним фреймворком майбутнього проєкту стане PyTorch. На відміну від альтернативних рішень, ця бібліотека базується на парадигмі динамічного графа обчислень (eager execution), що дозволить будувати та модифікувати архітектуру нейронної мережі

безпосередньо під час проходження сигналу. Це рішення є критичним для підсистеми оцінювання, оскільки обрані перцептивні метрики самі по собі є нейронними мережами, що вимагатимуть динамічного управління пам'яттю на графічному процесорі.

Важливим аспектом проєктування системи буде вибір формату збереження параметрів (ваг) навчених моделей. У розробці передбачається відмовитися від традиційного формату `.pth`, що базується на десеріалізації бібліотеки `pickle` і містить потенційну вразливість для виконання довільного шкідливого коду під час завантаження файлу. Натомість буде впроваджено підтримку формату `.safetensors`, який гарантує математичну структурну безпеку та забезпечує вищу швидкість завантаження завдяки механізму відображення пам'яті (`memory mapping`) безпосередньо у відеопам'ять графічного процесора.

Для управління залежностями та забезпечення повної відтворюваності середовища буде застосовано сучасний пакетний менеджер `uv` та стандарт конфігурації `pyproject.toml`. Наявність файлу фіксації стану `uv.lock` гарантуватиме абсолютну детермінованість середовища розроблення, що є базовою вимогою наукової достовірності отриманих результатів. Крім того, інфраструктура програми доповнюватиметься бібліотекою `NumPy` для оптимізованих матричних операцій та `OpenCV` для багатокритеріальної попередньої обробки зображень у відповідних модулях.

Обґрунтований вибір мови програмування `Python`, фреймворку `PyTorch` та прогресивних засобів управління пакетами сформує надійний технологічний фундамент для розроблення системи. Визначивши інструментарій та інтеграційні формати, що гарантують безпеку і швидкодію, подальша увага буде зосереджена безпосередньо на конкретних технічних та архітектурних рішеннях, які перетворюють створені алгоритмічні схеми на повноцінний програмний продукт.

2.4 Програмні рішення інтеграції моделей та оптимізації обчислень

Для забезпечення заявленої гнучкості та масштабованості системи планується застосувати низку сучасних архітектурних та алгоритмічних підходів. Оскільки в індустрії комп'ютерного зору існує проблема кардинальних відмінностей між архітектурами нейронних мереж, пряме їх вбудовування у ядро бенчмарку є недоцільним. Замість цього передбачається розробити систему стандартизованих класів-обгорт (wrappers). Кожна нейромережева модель буде інкапсульована у власний ізольований пакет, який взаємодіятиме з головним координатором системи виключно через єдиний програмний інтерфейс.

Управління гіперпараметрами та налаштуваннями моделей буде винесено за межі програмного коду у спеціалізовані конфігураційні файли формату YAML. Такий підхід дозволить досліднику змінювати коефіцієнт масштабування (scale factor), специфікації шарів та шляхи до файлів без будь-якого втручання у внутрішню логіку системи. З міркувань інформаційної безпеки та для підвищення швидкості ініціалізації планується повністю відмовитися від застарілих форматів збереження ваг мережі на користь формату безпечних тензорів. Це рішення гарантуватиме захист від виконання шкідливого коду під час завантаження даних та забезпечить пряме відображення масивів у відеопам'ять графічного процесора.

Окремим вагомим програмним рішенням стане впровадження алгоритму тайлінгу (tiling) для обробки зображень. З огляду на те, що трансформерні та дифузійні архітектури вимагають експоненційного збільшення обсягу відеопам'яті при зростанні розміру вхідного кадру, опрацювання цілих зображень високої роздільної здатності на споживчих відеокартах неминуче призведе до критичної помилки переповнення пам'яті. Для нівелювання цього ризику буде розроблено механізм просторової декомпозиції: велике зображення автоматично розбиватиметься на менші фрагменти (тайли), які по черзі проходитимуть через нейронну мережу.

Для уникнення появи візуальних швів та артефактів на межах зшивання фрагментів, алгоритм тайлінгу буде використовувати математичне перекриття зон. Крайові пікселі кожного фрагмента, що зазвичай страждають від нестачі контекстної інформації, будуть відкидатися або усереднюватися, залишаючи у фінальному результаті лише високоякісні центральні ділянки. Таке рішення перетворить проблему апаратних обмежень обсягу пам'яті на керовану задачу часової складності обчислень.

Оцінювання візуальної якості результатів базуватиметься на перенесенні всіх математичних розрахунків (таких як визначення метрик структурної схожості та пікового відношення сигналу до шуму) безпосередньо на тензорні ядра графічного процесора. Це дозволить уникнути ресурсномістких операцій копіювання даних між оперативною та відеопам'яттю. Для обчислення перцептивних метрик передбачається динамічне підключення допоміжних нейромережових моделей, які функціонуватимуть у суворому режимі інференсу без обчислення градієнтів, що мінімізує фонове навантаження на систему під час бенчмаркінгу.

2.5 Підсумки розділу та постановка завдань для наступного етапу

У межах поточного розділу було сформовано повноцінний теоретико-прикладний фундамент для створення системи порівняльного аналізу нейронних мереж. Розроблено схему проєктування структури програмного забезпечення, яка гарантує ізоляцію обчислювального ядра від логіки обробки та візуалізації даних. Спроектовано динамічну схему функціонування, що чітко регламентує алгоритми маршрутизації для режимів тестування, одиничного збільшення та візуального порівняння зображень.

Обґрунтовано вибір мови Python та фреймворку глибокого навчання PyTorch як єдино доцільних інструментів, що здатні забезпечити апаратне прискорення та безпосередню роботу з сучасними архітектурами машинного зору. Крім того, визначено ключові програмні рішення, такі як модульна

ізоляція, алгоритми тайлінгу та використання безпечних форматів збереження ваг, які дозволять обійти апаратні обмеження та забезпечити високу стабільність майбутнього комплексу.

Затвердивши архітектурні концепції та обравши технологічний інструментарій, наступним логічним етапом є детальне алгоритмічне проєктування внутрішніх підсистем. Наступний розділ буде присвячено плануванню розробки конкретних обчислювальних модулів, проєктуванню логіки збору апаратної статистики та алгоритмам побудови репрезентативних аналітичних графіків.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

3.1 Створення головного керуючого модуля та системи маршрутизації

Головний керуючий модуль виступатиме центральним вузлом майбутнього програмного комплексу. Його основне призначення – надання користувачеві зручного інтерфейсу командного рядка та перенаправлення потоку виконання програми до відповідних підсистем залежно від поставленого завдання.

Життєвий цикл роботи цього модуля планується ініціювати розбором аргументів командного рядка. Для забезпечення максимальної гнучкості та відтворюваності експериментів у системі буде впроваджено підтримку конфігураційних файлів формату YAML. Алгоритм роботи передбачатиме, що користувач зможе передати шлях до файлу конфігурації через відповідний параметр виклику. У такому випадку модуль зчитуватиме всі налаштування з файлу, що дозволить легко дублювати складні конфігурації для повторних запусків без необхідності ручного введення великої кількості аргументів у терміналі.

Програмна логіка керуючого модуля розділятиме виконання на три ізольовані режими:

- режим тестування (evaluate) – ініціюватиме створення об'єкта координатора для проведення комплексного бенчмаркінгу моделей на вказаних наборах даних зі збором відповідної статистики;
- режим збільшення (upscale) – завантажуватиме обрану модель та виконуватиме збільшення роздільної здатності для вказаних користувачем зображень (додатково передбачається можливість попереднього застосування математичної інтерполяції для штучного зменшення зображень перед їхнім відновленням нейромережею);
- режим порівняння (compare) – ініціюватиме виклик методів підсистеми візуалізації для генерації порівняльних колажів, що дозволить

візуально зіставити роботу декількох архітектур на одному графічному полотні.

Таким чином, спроектований головний керуючий модуль забезпечить надійну маршрутизацію та зручне управління всіма функціями системи через термінал і конфігураційні файли. Оскільки цей модуль буде відповідати виключно за розподіл завдань, для безпосереднього управління процесом масового тестування та взаємодії з нейронними мережами необхідно розробити спеціалізований внутрішній вузол. З огляду на це, логічним наступним кроком є розроблення алгоритмів координації та управління тестуванням, які візьмуть на себе роль структурного посередника між точкою входу програми та її обчислювальними підсистемами.

3.2 Розробка алгоритмів координації

Модуль координації відіграватиме роль структурного посередника між точкою входу програми та її обчислювальними підсистемами. Всупереч традиційним монолітним підходам, даний компонент не міститиме низькорівневої логіки прямого проходження тензорів через нейронні мережі або управління графами обчислень. Його головне функціональне призначення полягатиме в оркестрації високорівневих процесів тестування та делегуванні вузькоспеціалізованих завдань відповідним класам-виконавцям.

Життєвий цикл роботи цього алгоритму починатиметься після того, як головний керуючий скрипт передасть йому оброблені параметри. Отримавши ці вхідні дані (перелік моделей, наборів даних, цільовий пристрій обчислень та шляхи до цільових директорій), координатор здійснюватиме ініціалізацію робочого середовища та створюватиме екземпляри ключових робочих класів: підсистеми оцінювання та підсистеми візуалізації.

Алгоритм виконання тестування буде побудований на послідовній взаємодії та передачі даних між зазначеними класами:

- на першому етапі координатор ініціюватиме виклик обчислювальних методів підсистеми оцінювання для конкретної обраної моделі;
- після успішного завершення обчислень підсистема оцінювання повертатиме структурований масив статистичних даних, що включатиме значення математичних та перцептивних метрик, апаратні показники швидкодії, а також показники теоретичної складності архітектури (кількість параметрів та операцій з рухомою комою);
- критично важливим етапом алгоритму стане примусове очищення оперативної пам'яті та кешу графічного процесора після кожної протестованої моделі, що дозволить уникнути переповнення пам'яті при масовому тестуванні важких архітектур;
- на фінальному етапі координатор виконуватиме передачу отриманих агрегованих результатів до підсистеми візуалізації, яка автоматично генеруватиме аналітичні діаграми, графіки співвідношення якості та продуктивності, а також зберігатиме загальні результати у Excel файл.

Для режимів одиничного збільшення та візуального порівняння логіка роботи координатора передбачатиме керування процесом зчитування зображень, їхньою передачею до моделей та подальшим збереженням реконструйованих файлів або сформованих порівняльних колажів на диск. У разі відсутності зображень низької якості для порівняння, алгоритм автоматично ініціюватиме їх створення за допомогою методу бікубічної інтерполяції.

Використання такого архітектурного рішення забезпечить суворе дотримання принципу єдиної відповідальності. Модуль координації залишатиметься повністю абстрагованим від математики фреймворку глибокого навчання та інструментів малювання графічних бібліотек. Завдяки цьому програмний комплекс характеризуватиметься високим рівнем підтримуваності. Водночас для успішного виконання обчислень безпосередньо всередині підсистеми оцінювання, особливо при роботі зі складними трансформерами та дифузійними моделями на зображеннях

високої роздільної здатності, необхідно розв'язати проблему експоненційного зростання споживання відеопам'яті. З огляду на це, логічним наступним кроком є розроблення спеціального алгоритму просторової декомпозиції зображень (тайлінгу) для оптимізації тензорних обчислень.

3.3 Алгоритм тайлінгового інференсу для оптимізації використання пам'яті

Для забезпечення працездатності сучасних нейромережових архітектур, зокрема візуальних трансформерів та дифузійних моделей, на споживчому апаратному забезпеченні планується розробити та впровадити спеціалізований алгоритм тайлінгового інференсу (tiling inference). Необхідність такого підходу зумовлена експоненційним зростанням обсягів споживання відеопам'яті (VRAM) при збільшенні просторової роздільної здатності вхідних тензорів, що часто призводить до критичних помилок переповнення ресурсів графічного процесора.

Алгоритмічно цей процес буде реалізовано на основі принципу просторової декомпозиції, де вихідне зображення низької якості автоматично розбиватиметься на сітку менших фрагментів – тайлів. Ключовими параметрами керування цим процесом стануть розмір фрагмента (tile size) та величина математичного перекриття (tile overlap). Останній параметр має критичне значення для збереження візуальної цілісності контенту: він дозволить нейронній мережі отримувати достатній просторовий контекст на межах кожного фрагмента, що запобігатиме появі характерних артефактів "швів" та розривів ліній під час фінальної реконструкції.

Процес обробки планується побудувати за наступною схемою:

- кожен тайл проходитиме через нейронну мережу незалежно, що дозволить ефективно використовувати обмежені ресурси VRAM;
- для усунення крайових дефектів передбачається використання механізму реплікації пікселів на кордонах патчів (replicate padding);

– фінальне зшивання реконструйованого зображення здійснюватиметься шляхом накопичення значень у глобальній ваговій карті з наступним усередненням зон перекриття через метод блендингу.

Така програмна оптимізація має забезпечити можливість виконання збільшення роздільної здатності для зображень практично будь-якого формату навіть на графічних картах із невеликим обсягом пам'яті.

Впровадження алгоритму тайлінгового інференсу дозволить нівелювати апаратні обмеження пам'яті та забезпечить стабільну роботу найбільш обчислювально складних архітектур. Після вирішення проблеми обробки великих масивів пікселів, наступним завданням є планування логіки підготовки вхідної інформації та розрахунку кількісних показників якості. Таким чином, логічним продовженням є проєктування підсистем обробки графічних даних та розрахунку метрик.

3.4 Проєктування підсистем обробки графічних даних та розрахунку метрик

Центральним елементом будь-якої системи оцінювання є надійна підсистема обробки даних та точного розрахунку метрик. Планується, що ця підсистема складатиметься з двох логічних блоків: перший відповідатиме за підготовку зображень, а другий – за їх об'єктивне оцінювання.

3.4.1 Процес підготовки та стандартизації вхідних даних

Для забезпечення стабільного завантаження даних у відеопам'ять планується спроектувати строгу послідовність обробки зображень. Процес обробки ініціюватиметься зчитуванням графічних файлів із диска. Використання стандарту RGB є обов'язковим для забезпечення відповідності форматам тренування більшості досліджуваних моделей, тому передбачається автоматична конвертація кольорових просторів у разі потреби.

Для оптимізації тензорних обчислень буде реалізовано переведення цілочисельних значень інтенсивності пікселів із діапазону $[0, 255]$ у формат з рухомою комою в діапазоні $[0, 1]$. Це дозволить стабілізувати процес обчислення градієнтів та запобігатиме переповненню під час генерації.

Для роботи з зображеннями надвисокої роздільної здатності передбачається створення алгоритму розбиття на патчі за допомогою методу ковзного вікна із заданим кроком. Такий алгоритм дозволить обробляти зображення частинами та зшивати результати в єдине полотно без видимих швів на межах фрагментів.

Крім того, для сумісності з архітектурами розроблюваних моделей, передбачено зміну розмірності вхідних даних із формату висота-ширина-канали у формат канали-висота-ширина. Такий підхід дозволить паралелізувати обробку кольорних каналів на апаратному рівні.

3.4.2 Алгоритми оцінювання якості

З метою забезпечення комплексного оцінювання результатів роботи нейромережових моделей, передбачається реалізувати розрахунок як класичних статистичних метрик, так і перцептивних нейромережових методів.

Обчислення метрики PSNR [3] базуватиметься на попередньому розрахунку середньоквадратичної помилки (MSE). Цей підхід є надійним математичним інструментом для вимірювання абсолютної різниці між пікселями. Після отримання значення MSE розраховуватиметься PSNR у децибелах, спираючись на логарифмічне відношення максимально можливої потужності сигналу до потужності шуму.

Алгоритм SSIM [4] буде реалізовано через аналіз трьох компонентів: яскравості, контрасту та структури. Використання механізму ковзного вікна гауссівського розподілу дозволить обчислювати локальні середні значення та дисперсії, що значно наблизить метрику до особливостей людського сприйняття.

Для оцінювання перцептивної якості зображень передбачається імплементувати алгоритми виклику зовнішніх моделей LPIPS [5] та CLIPQA [6]. Передача тензорів у попередньо навчену нейронну мережу дозволить видобувати карти ознак із різних шарів мережі та математично обчислювати відстань між ними. Модель CLIPQA планується використовувати для порівняння вектора ознак зображення з векторами текстових описів, що дасть змогу отримати достовірну оцінку без використання еталонного зображення. Для розрахунку метрики MUSIQ [7] передбачається застосування трансформерної архітектури для аналізу зображення на різних масштабах, що дозволить враховувати як глобальну композицію, так і локальні мікротектури.

Усі обчислення в межах даної підсистеми будуть виконуватися із вимкненим розрахунком градієнтів. Це є критично необхідним заходом для мінімізації споживання обчислювальних ресурсів під час масштабного тестування.

3.5 Механізми збору апаратної статистики та логування процесів

Процес порівняльного аналізу нейромережових моделей передбачає не лише оцінку якості реконструйованих зображень, а й детальне відстеження використання системних ресурсів. Для забезпечення об'єктивності результатів планується реалізувати алгоритми розрахунку теоретичної обчислювальної складності та фактичного навантаження на апаратне забезпечення.

Розрахунок теоретичної обчислювальної складності моделей базуватиметься на визначенні кількості операцій множення-накопичення (MACs) та кількості операцій з рухомою комою (FLOPs). Показник FLOPs розраховуватиметься як подвоєне значення MACs, враховуючи окремо операції множення та додавання для кожного елемента тензора. Передбачається впровадження алгоритму обходу графа моделі, що дозволить підсумовувати ці показники для кожного шару архітектури. Це забезпечить

отримання точної теоретичної оцінки складності незалежно від конфігурації конкретного обладнання.

Для моніторингу споживання відеопам'яті (VRAM) під час інференсу планується використання спеціалізованого інструментарію фреймворку глибокого навчання. Механізм фіксації передбачатиме реєстрацію максимального обсягу пам'яті, який зарезервовано менеджером пам'яті графічного процесора під час роботи алгоритму. Використання саме показника пікового резервування дозволить найточніше оцінити реальні апаратні вимоги моделі, оскільки він враховує внутрішню фрагментацію та кешування даних системним алокатором.

Вимірювання часу виконання буде спроектовано з урахуванням асинхронного характеру обчислень на графічних прискорювачах. Для уникнення похибок, спричинених черговістю виконання команд у потоках, замість стандартних системних таймерів центрального процесора планується застосувати апаратні події. Алгоритм передбачатиме розміщення маркерів початку та завершення операції безпосередньо в черзі виконання пристрою. Перед обчисленням фінальної різниці в часі ініціюватиметься примусова синхронізація процесів, що гарантуватиме завершення всіх паралельних обчислень та отримання достовірного часу інференсу. У разі виконання на центральному процесорі використовуватимуться стандартні високоточні лічильники продуктивності операційної системи.

Для забезпечення цілісності отриманих даних передбачається розробка механізму логування. Збереження результатів здійснюватиметься паралельно у структуровані текстові файли та виводитиметься в консоль. Форматований вивід включатиме ітераційні показники якості, завантаженість VRAM та поточну швидкість обробки, що дозволить оперативно виявляти аномалії в роботі нейронних мереж безпосередньо під час тестування.

3.6 Створення інструментарію для візуалізації результатів та генерації звітів

Функціональне призначення підсистеми візуалізації полягатиме у перетворенні накопичених кількісних даних та бінарних результатів роботи нейронних мереж у репрезентативні графічні форми для подальшого аналітичного дослідження. Планується, що логіка роботи з графічними елементами базуватиметься на дворівневому підході: базове керування графічними примітивами та накладання статистичних абстракцій для підвищення загальної інформативності графіків при порівнянні декількох архітектур одночасно.

Алгоритм побудови діаграм порівняння передбачатиме автоматичне формування кількох типів залежностей на основі масивів даних, зібраних під час тестування. По-перше, здійснюватиметься візуалізація компромісу між швидкістю інференсу та показниками якості (PSNR, MUSIQ, тощо) у вигляді бульбашкових діаграм розсіювання, де позиція елемента визначатиме його ефективність, а розмір – обчислювальну складність (кількість параметрів моделі). По-друге, ініціюватиметься генерація багатовимірних пелюсткових (радарних) діаграм для комплексного оцінювання балансу характеристик кожної архітектури. Усі отримані статистичні показники також автоматично форматоватимуться та експортуватимуться у зведені електронні таблиці для забезпечення зручного доступу до сирих метрик.

Окрему і найбільш складну частину підсистеми становитиме механізм візуального порівняння згенерованих зображень. Для детального якісного аналізу артефактів та ступеня реконструкції мікротекстур планується розробити алгоритм створення порівняльних колажів (side-by-side comparison) із виділенням локальних областей інтересу (Region of Interest, ROI). Процес просторового зіставлення виконуватиметься у декілька етапів:

- визначення координат цільової області інтересу на еталонному зображенні високої якості;

- виконання операції вирізання фрагмента (кропінгу) ідентичного розміру для оригінального зображення, його деградованої копії (результату бікубічної інтерполяції) та результатів реконструкції від усіх досліджуваних нейромережових моделей;
- масштабування виділеного фрагмента за допомогою математичної інтерполяції найближчого сусіда (Nearest Neighbor);
- розміщення збільшених патчів у єдиній сітці з накладанням контурних рамок, колір яких відповідає конкретній моделі на графіках, для зручного візуального зіставлення.

Використання алгоритму найближчого сусіда на етапі масштабування фрагментів є принциповим інженерним рішенням. Такий підхід дозволяє зберегти оригінальну піксельну структуру та виявити специфічні артефакти нейронних мереж (наприклад, ефекти ореолу, геометричні викривлення чи пересвітлення) без внесення додаткового стороннього згладжування, яке неминуче виникає при застосуванні лінійних методів збільшення.

3.7 Підготовка та налаштування нейромережових архітектур

Архітектурна підготовка досліджуваних моделей, включаючи SRResNet, RCAN, SRGAN, ESRGAN, Real-ESRGAN, SwinIR, HAT та ResShift, базується на адаптації їхньої структури для майбутньої інтеграції у запланований програмний комплекс. Для забезпечення коректної роботи майбутньої системи кожна архітектура інкапсулюватиметься в окремий підпакет, де зберігатимуться відповідні параметри.

Отже, зважаючи на те, що теоретична база для моделей сформована, необхідно детально розглянути етап їхнього попереднього тренування та показники затрат часу, що стануть основою для інтеграції у заплановану систему.

3.7.1 Планування процесу тренування моделей

На відміну від підходів з уніфікованим навчанням, кожна архітектура тренувалася на оптимальних для неї вибірках (таких як DIV2K [30] або DF2K [31]). Таке рішення обґрунтоване власним баченням необхідності тестування моделей у їхньому найкращому стані. Час, витрачений на відтворення та валідацію циклів навчання на апаратних потужностях графічного процесора NVIDIA RTX 4060 Ti 8GB, наведено у таблиці 3.1.

Таблиця 3.1 – Характеристики та час тренування нейромережових моделей

Назва моделі	Час тренування, г.	Кількість параметрів, М.	Кількість FLOPs (для зображення 64 × 64 пікселі)
SRResNet	59	3,45	39,41
SRGAN	15	3,45	39,41
ESRGAN	35,5	16,91	144,16
Real-ESRGAN	78	16,91	144,16
RCAN	24	15,89	129,83
SwinIR	24	13,16	107,84
HAT	322	20,77	172,05

Аналіз даних таблиці 3.1 демонструє суттєву розбіжність у часових витратах на навчання: від 15 годин для моделі SRGAN до 322 годин для гібридного трансформера HAT. Така різниця в часі не є критичним фактором для майбутнього об'єктивного оцінювання ефективності архітектур, оскільки у запланованому бенчмарку ці моделі використовуватимуться виключно в режимі інференсу. Основний внесок конкретної архітектури у якість відновлення високочастотних деталей стає очевидним ще до моменту повного насичення процесу навчання. Розрахунок кількості параметрів та

обчислювальної складності для генеративно-змагальних архітектур проведено виключно для компонентів генератора, оскільки дискримінатор у майбутній системі не задіюватиметься.

Підсумовуючи результати тренування, можна стверджувати, що всі необхідні моделі підготовлені та мають відповідні набори вагових коефіцієнтів. Оскільки математична база готова, наступним кроком є розроблення стратегії оптимізації тензорних обчислень для забезпечення безперебійної роботи цих архітектур у майбутній програмі.

3.7.2 Оптимізація тензорних обчислень для споживчих графічних процесорів

При проєктуванні програмної системи особливу увагу приділено розробці оптимізацій, які будуть спрямовані на максимально ефективне використання ресурсів споживчих графічних процесорів. Для майбутнього прискорення матричного множення та операцій згортки планується активувати режим TensorFloat-32 (TF32). Використання тензорними ядрами 19-бітного формату обчислень дозволить значно підвищити пропускну здатність обчислювального блоку без суттєвої втрати візуальної точності, що є критично важливим для запланованої роботи з глибокими архітектурами.

З метою економії відеопам'яті під час експлуатації комплексу передбачається впровадження технології змішаної точності (Mixed Precision) з використанням типу даних `bfloat16`. Такий підхід нівелює проблему переповнення пам'яті на масивних зображеннях, зберігаючи при цьому стабільність обчислень. Для оптимізації обміну даними між оперативною та відеопам'яттю у майбутньому коді буде реалізовано механізм неблокуючої передачі тензорів з попереднім закріпленням пам'яті.

Крім того, планується автоматичне переведення моделей та вхідних тензорів у формат пам'яті Channels Last (NHWC). Це дозволить апаратному забезпеченню ефективніше звертатися до даних у кеші при паралельній

обробці колірних каналів. Налаштування підсистеми завантаження даних також перебуває на стадії проєктування: кількість потоків дорівнюватиме кількості фізичних ядер, а параметр пре-вибірки дозволить створити постійну чергу готових пакетів для уникнення простоїв графічного процесора.

3.8 Підсумки розділу та постановка завдань для наступного етапу

У межах поточного розділу було сформовано детальну логіку функціонування внутрішніх підсистем майбутнього програмного комплексу. Спроектовано систему маршрутизації та головний керуючий вузол, що забезпечить ізоляцію високорівневої логіки управління від безпосередніх математичних обчислень.

Для подолання апаратних обмежень відеопам'яті буде розроблено алгоритм просторової декомпозиції зображень (тайлінгу), який дозволить обробляти зображення надвисокої роздільної здатності навіть на споживчих графічних процесорах. Крім того, сплановано механізми підготовки графічних даних та розрахунку як класичних повноеталонних, так і сучасних перцептивних метрик оцінювання якості. Окрему увагу було приділено проєктуванню підсистем збору апаратної статистики та візуалізації результатів. Передбачається, що розроблені алгоритми генерації порівняльних колажів та побудови графіків дозволять здійснювати глибокий аналітичний розбір ефективності кожної досліджуваної моделі. Також було сплановано процес попереднього тренування та підготовки вагових коефіцієнтів для обраних архітектур.

Завершивши етап проєктування та підготувавши теоретичне й алгоритмічне підґрунтя для всіх модулів, наступним логічним кроком є перехід до розгортання та практичної експлуатації створеного інструментарію.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Вимоги до програмно-апаратного середовища та експлуатація системи

Для забезпечення коректної роботи розробленого програмного комплексу SRBenchmark, а також для проведення об'єктивних, швидких та безперебійних експериментальних досліджень було сформовано спеціалізований високопродуктивний апаратний стенд. Основною обчислювальною одиницею виступає графічний процесор NVIDIA RTX 4060 Ti (8 Гб) з підтримкою технології паралельних обчислень CUDA. Використання сучасного графічного прискорювача є критично важливою умовою для задач комп'ютерного зору, оскільки інференс глибоких нейронних мереж (зокрема складних трансформерних та дифузійних архітектур) вимагає значної пропускної здатності пам'яті та апаратного прискорення матричних множень. Апаратний комплекс додатково оснащено 32 Гб оперативної пам'яті (RAM), що гарантує відсутність "вузьких місць" при попередній обробці великих пакетів зображень (батчів), та твердотільним накопичувачем об'ємом 1 ТБ формату NVMe M.2 SSD. Застосування NVMe-накопичувача дозволило радикально мінімізувати затримки вводу-виводу (I/O bottlenecks) під час ітеративного зчитування масивних тестових датасетів, забезпечуючи безперервне надходження даних до конвеєра графічного процесора.

Розробку, налагодження та розгортання програмного комплексу було реалізовано в унікальному, глибоко оптимізованому середовищі на базі операційної системи Arch Linux. Вибір цього дистрибутиву обґрунтовано можливістю гнучкого налаштування ядра та відсутністю надлишкових фонових процесів, що дозволило вивільнити максимальний обсяг системних ресурсів виключно для потреб машинного навчання. Як основний інструментарій розробника використовувався сучасний апаратно-прискорений термінальний емулятор Ghostty у поєднанні з термінальним

мультиплексором `tmux`. Це забезпечило ефективне управління ізольованими робочими просторами, дозволяючи одночасно відстежувати системні ресурси, виконувати скрипти та редагувати конфігурації. Написання коду та його рефакторинг здійснювалися за допомогою текстового редактора `Neovim`. Такий консольно-орієнтований підхід виключає необхідність використання важковагових графічних середовищ розробки (IDE), що додатково підвищує загальну швидкодію системи під час тривалих обчислювальних сесій.

Основним технологічним стеком для реалізації алгоритмів збільшення роздільної здатності зображень обрано мову програмування `Python` версії 3.13 та фреймворк глибокого навчання `PyTorch`. Фундаментальним аспектом будь-якого наукового дослідження є гарантія 100% відтворюваності експериментального середовища. Для досягнення цієї мети та розв'язання проблеми конфліктів версій було реалізовано використання сучасного пакетного менеджера `uv`, розробленого на мові `Rust`. На відміну від традиційних інструментів (таких як `pip`), `uv` забезпечує безпрецедентну швидкість резолвінгу залежностей та високу надійність ізоляції середовища. Управління конфігурацією проєкту здійснюється через стандартизований файл `pyproject.toml`, який описує всі метадані та архітектурні вимоги розробленого бенчмарку. Водночас файл фіксації стану `uv.lock` гарантує абсолютну детермінованість середовища: він зберігає точні криптографічні хеші та версії кожної встановленої бібліотеки та її підзалежностей. Це дає гарантію того, що при перенесенні системи на інший апаратний стенд поведінка моделей та результати обчислення метрик залишаться ідентичними.

Процес експлуатації програмного комплексу `SRBenchmark` спроектовано з урахуванням найкращих практик програмної інженерії, зокрема принципу відкритості-закритості (`Open-Closed Principle`). Точкою входу та головним оркестратором системи є файл `main.py`, запуск якого через інтерфейс командного рядка ініціює повний цикл тестування: завантаження даних, ініціалізацію вибраних моделей, розрахунок метрик та логування результатів.

Архітектура комплексу є суворо модульною. Усі досліджувані архітектури нейронних мереж ізольовані у відповідних піддиректоріях всередині директорії `models` (наприклад, `models/ESRGAN`, `models/HAT` тощо). Завдяки цьому додавання нової моделі для дослідження не потребує втручання в ядро системи. Користувачеві достатньо створити нову директорію, що міститиме клас архітектури та конфігураційний файл `config.yaml`. Цей файл відіграє ключову роль в управлінні експериментом: він дозволяє декларативно задавати гіперпараметри мережі, коефіцієнт масштабування (*scale factor*) та вказувати шляхи до попередньо навчених ваг, не змінюючи при цьому програмний код.

Особливу увагу під час розроблення та експлуатації системи було приділено питанням інформаційної безпеки та оптимізації пам'яті. У розробленому бенчмарку суворо регламентовано використання формату `.safetensors` для збереження та завантаження параметрів (ваг) нейронних мереж замість традиційного формату `.pth`. Відомо, що формат `.pth` базується на серіалізаторі Python `pickle`, який під час завантаження даних здатен виконувати довільний (зокрема шкідливий) системний код. Натомість `.safetensors` є математично та структурно безпечним форматом, що містить виключно тензорні масиви та JSON-метадані. Крім абсолютних гарантій безпеки, цей формат підтримує механізм відображення пам'яті (*memory mapping*), що дозволяє завантажувати великі обсяги ваг (характерні для моделей SwinIR або HAT) безпосередньо у відеопам'ять графічного процесора, мінаючи етап попереднього копіювання в оперативну пам'ять, що суттєво прискорює ініціалізацію системи.

Експлуатація програмного комплексу здійснюється виключно через інтерфейс командного рядка (CLI), що дозволяє легко інтегрувати систему в автоматизовані скрипти, конвеєри тестування (CI/CD) та виконувати дослідження на віддалених серверах без графічного інтерфейсу. Для отримання вичерпної довідки щодо доступних аргументів та параметрів запуску користувачеві достатньо виконати команду: `python main.py --help`.

Фундаментальною перевагою інтерфейсу є підтримка конфігураційних файлів. Замість введення довгого списку аргументів у терміналі, користувач може передати прапорець `-c` або `--config` із вказівкою шляху до файлу формату YAML. Це дозволяє зберегти складні налаштування експерименту в одному файлі та забезпечити абсолютну зручність для повторних запусків досліджень.

Повний перелік параметрів командного рядка, що підтримуються системою, наведено у таблиці 4.1.

Таблиця 4.1 – Параметри інтерфейсу командного рядка програмного комплексу

Назва параметра	Скорочення	Опис та призначення
1	2	3
tasks	—	Позиційний аргумент, що визначає перелік завдань для виконання: benchmark (тестування), upscale (збільшення роздільної здатності), compare (створення колажів).
--config	-c	Шлях до файлу конфігурації формату YAML, що дозволяє автоматизувати запуск, зчитуючи всі параметри з одного файлу.
--models	-m	Перелік нейромережових моделей для використання. Доступні варіанти генеруються автоматично на основі папки models.
--datasets	-d	Шляхи до кореневих папок із наборами даних. Є обов'язковим параметром для режиму benchmark.

Продовження таблиці 4.1

Назва параметра	Скорочення	Опис та призначення
1	2	3
--input	-i	Шляхи до вхідних зображень високої якості (HR). Є обов'язковим параметром для режимів upscale та compare.
--output	-o	Директорія для збереження вихідних файлів: реконструйованих зображень, аналітичних звітів або графіків.
--lr-input	-lri	Шляхи до зображень низької якості (LR) для режиму порівняння. Значення None ініціює автоматичну генерацію LR-копій.
--crop-size	—	Розмір фрагмента (в пікселях) для створення колажів у режимі compare, що дозволяє детально дослідити мікротекстури.
--downscale	—	Прапорець, що активує попереднє зменшення зображення перед збільшенням для перевірки точності відновлення в режимі upscale.
--tile-size	—	Розмір тайла (фрагмента) для поетапної обробки великих зображень з метою економії відеопам'яті (VRAM).
--tile-overlap	—	Значення перекриття тайлів у пікселях для запобігання появі видимих швів на реконструйованому зображенні.

Кінець таблиці 4.1

Назва параметра	Скорочення	Опис та призначення
1	2	3
--num_workers	—	Кількість паралельних потоків для багатопотокового завантаження даних із диска до оперативної пам'яті.
--prefetch_factor	—	Кількість пакетів даних, які заздалегідь підвантажуються кожним потоком для мінімізації простоїв ГП.

Використання вказаних параметрів дозволяє адаптувати роботу програми під конкретні апаратні можливості користувача, зокрема шляхом регулювання розміру тайлів та кількості обчислювальних потоків. Для повторних експериментів рекомендовано використовувати прапорець --config для забезпечення детермінованості умов тестування.

4.2 Інструкція до встановлення та використання програми

Для початку роботи з розробленим програмним комплексом необхідно підготувати робоче середовище. Базовою вимогою до апаратного забезпечення є наявність персонального комп'ютера з операційною системою сімейства Linux або Windows та бажано графічного прискорювача з підтримкою технології паралельних обчислень.

Оскільки система побудована з використанням сучасного стеку технологій, процес встановлення є максимально автоматизованим. На цільовому пристрої має бути встановлений інтерпретатор мови Python версії 3.13. Управління залежностями здійснюється за допомогою пакетного менеджера uv. Для розгортання проєкту достатньо завантажити директорію з вихідним кодом та виконати команду синхронізації в терміналі. Ця дія автоматично створить ізольоване віртуальне середовище та встановить усі

необхідні бібліотеки, спираючись на зафіксовані версії у файлі блокування, що гарантує стовідсоткову відтворюваність середовища на будь-якій машині

Після встановлення залежностей виконується підготовка нейромережових моделей. Для кожної архітектури необхідно завантажити файли попередньо навчених ваг у форматі `safetensors` або `pth` та розмістити їх у відповідних внутрішніх каталогах моделей. За потреби гіперпараметри мереж можна змінити у відповідних файлах конфігурації формату `YAML`, що дозволяє уникнути втручання в основний програмний код.

Експлуатація системи здійснюється через інтерфейс командного рядка, що забезпечує швидкий доступ до всіх передбачених функцій. Головний модуль підтримує три основні режими роботи: тестування (`evaluate`), збільшення (`upscale`) та порівняння (`compare`).

Режим тестування призначений для виконання комплексного оцінювання архітектур. Для його запуску необхідно вказати відповідну команду, передати конфігураційний файл та надати шлях до директорії з еталонними наборами даних. Програма автоматично обробить усі зображення та згенерує підсумкові звіти з розрахованими метриками продуктивності та якості.

Режим збільшення дозволяє здійснювати безпосереднє збільшення роздільної здатності користувацьких зображень. У параметрах командного рядка необхідно вказати шлях до вихідних файлів та цільову папку для збереження результатів. Система також підтримує додаткові аргументи, такі як активація алгоритму тайлінгу (просторового розбиття на фрагменти), що дозволяє уникнути переповнення відеопам'яті при обробці зображень надвисокої роздільної здатності на споживчих графічних процесорах.

Режим порівняння створено для візуального аналізу. Під час запуску цього режиму користувач отримує можливість згенерувати зведені колажі, на яких відображаються результати роботи різних архітектур разом із виділеною областю інтересу. Це дозволяє наочно оцінити здатність кожної нейронної

мережі до відновлення мікротекстур та виявити специфічні візуальні артефакти або геометричні викривлення.

Підсумовуючи, розроблена інструкція чітко регламентує послідовність дій для швидкого розгортання та ефективної експлуатації програмного комплексу. Завдяки використанню новітніх інструментів управління залежностями та зручному інтерфейсу командного рядка, процес налаштування займає мінімум часу і повністю виключає виникнення конфліктів версій програмного забезпечення. Гнучкі параметри запуску дозволяють легко адаптувати систему під різноманітні дослідницькі задачі та наявні апаратні обмеження. Успішне встановлення програмного інструментарію та підтвердження його стабільної роботи в базових режимах відкриває шлях до практичного використання системи. Відповідно, наступним логічним кроком є формування чіткої методології тестування, вибір еталонних наборів даних та визначення об'єктивних критеріїв для проведення повноцінного експериментального порівняння архітектур нейронних мереж.

4.3 Методика проведення експериментальних досліджень

Для забезпечення репрезентативності, об'єктивності та відтворюваності результатів тестування архітектур нейронних мереж було розроблено комплексну методологію проведення експериментальних досліджень. Враховуючи, що задача збільшення роздільної здатності зображень (Super-Resolution) належить до класу некоректно поставлених обернених задач, оцінювання продуктивності алгоритмів вимагає багатовимірного підходу. Такий підхід має враховувати не лише математичну точність реконструкції сигналу, а й особливості людського візуального сприйняття, а також апаратну ефективність моделей.

Для верифікації здатності нейронних мереж генерувати високоякісний контент було обрано п'ять стандартизованих наборів даних (датасетів), які є загальновизнаним індустріальним стандартом у галузі комп'ютерного зору:

– Set5 та Set14 [33] – класичні тестові вибірки, що містять зображення з різноманітними частотними характеристиками (обличчя, флора, фауна), які використовуються для базового оцінювання здатності моделі відновлювати чіткі контури та загальні текстури без внесення грубих артефактів;

– BSDS100 [33] – вибірка зі 100 зображень з Berkeley Segmentation Dataset, що характеризується широким розмаїттям природних сцен, завдяки чому використання цього датасету дозволяє перевірити здатність моделей до генералізації на складних природних текстурах, таких як листя, вода або пісок;

– Urban100 [33] – спеціалізований набір даних, що складається виключно із зображень міських пейзажів та архітектури і містить велику кількість повторюваних геометричних патернів, ліній та структурних решіток., що є критично важливим для перевірки здатності мережі коректно відновлювати високочастотні структури без ефекту накладання частот (аліасингу) та муарових візерунків;

– Manga109 [34] – датасет, що складається з відсканованих сторінок японських коміксів (манги) і кардинально відрізняється від природних фотографій, що дозволяє оцінити, наскільки добре архітектура справляється з відновленням штучних ліній, різких градієнтів, тексту та специфічних графічних патернів (скрінтонів).

Оскільки не існує єдиної універсальної функції оцінювання якості візуального контенту, методикою дослідження передбачено класифікацію та використання трьох різних груп метрик:

Перша група – класичні повноеталонні метрики (Full-Reference IQA), які вимагають наявності оригінального зображення (Ground Truth) для попиксельного порівняння:

– PSNR [3] вимірює абсолютну математичну точність реконструкції на основі середньоквадратичної помилки, але хоча ця метрика є стандартом де-факто, вона має відомий недолік – схильність високо оцінювати розмиті зображення, оскільки усереднення пікселів мінімізує статистичну помилку;

– SSIM [4] фокусується на вимірюванні деградації структурної інформації, порівнюючи яскравість, контраст та взаємозв'язки між сусідніми пікселями, що є ближчим до людського сприйняття геометрії об'єктів.

Друга група – перцептивні метрики:

– LPIPS [5] вирішує так званий "парадокс розмиття" класичних метрик, обчислюючи відстань між відновленим та еталонним зображеннями не в просторі пікселів, а у просторі глибоких ознак (feature space), витягнутих за допомогою попередньо навченої нейронної мережі (наприклад, VGG), що імітує когнітивне сприйняття людини: навіть якщо згенерована текстура (наприклад, пори шкіри) дещо зміщена просторово відносно оригіналу, LPIPS розпізнає її як високоякісну та фотореалістичну.

Третя група – безреференсні метрики (No-Reference IQA), які дозволяють оцінити зображення автономно:

– CLIP-IQA [6] використовує потужний мультимодальний простір моделі CLIP для оцінювання природності та естетичної привабливості контенту;

– MUSIQ [7] застосовує трансформерну архітектуру для аналізу зображення на різних масштабах, одночасно фіксуючи локальні артефакти генерації та оцінюючи глобальну композицію кадру.

Окрім оцінювання візуальної якості, повноцінне інженерне дослідження вимагає ретельного профілювання обчислювальної ефективності. Для цього методикою передбачено паралельний збір апаратних метрик. Вимірювався час інференсу (Latency, у секундах), що демонструє практичну придатність моделей для систем реального часу. Також фіксувалося максимальне резервування відеопам'яті (VRAM Max Reserved, у ГБ) за допомогою вбудованих профайлерів PyTorch, що дозволяє визначити мінімальні апаратні вимоги для успішного розгортання конкретної архітектури в експлуатаційних умовах.

Застосування описаної багатокритеріальної методики формує надійний статистичний фундамент. Агрегація даних за всіма трьома групами візуальних

метрик у поєднанні з показниками апаратного навантаження створює передумови для об'єктивного та глибоко аналітичного порівняння архітектур. Детальний розгляд результатів цього комплексного експерименту, їх візуалізація та інтерпретація математичних і перцептивних показників наведено у наступному підрозділі.

4.4 Аналіз якості відновлення зображень за математичними та перцептивними метриками

На основі проведених експериментальних досліджень було здійснено комплексне оцінювання восьми архітектур нейронних мереж: ESRGAN, HAT, RCAN, Real-ESRGAN, ResShift, SRGAN, SRResNet та SwinIR. Порівняльний аналіз виконувався за допомогою п'яти репрезентативних метрик на п'яти еталонних наборах даних (Set5, Set14, BSDS100, Urban100 та Manga109), що дозволило отримати вичерпну картину ефективності різних підходів до задачі Super-Resolution (SR).

Аналіз результатів за класичними повноеталонними метриками PSNR (рисунок 4.1) та SSIM (рисунок 4.2) продемонстрував безумовне лідерство архітектур, що базуються на механізмах візуальних трансформерів та розширеної уваги. Зокрема, модель HAT (Hybrid Attention Transformer) показала найвищі результати на наборі даних Set5, досягнувши показника PSNR у 32,74 дБ та індексу SSIM на рівні 0,904. Подібну ефективність демонструють архітектура SwinIR (32,63 дБ PSNR та 0,901 SSIM на Set5) та CNN з механізмом уваги RCAN (32,20 дБ PSNR та 0,897 SSIM).

Наукове обґрунтування такої переваги в математичній точності полягає в особливостях цільових функцій втрат, що використовуються при навчанні даних моделей. Орієнтація на мінімізацію попиксельної помилки (функції втрат L1 або L2/MSE) призводить до того, що мережа прагне знайти статистичне середнє значення для кожного пікселя. Це забезпечує високі показники PSNR та структурну стабільність (SSIM), проте спричиняє ефект

"розмиття" або "замилювання" високочастотних текстур, оскільки алгоритм уникає ризику генерації помилкових деталей, які могли б суттєво погіршити математичну метрику.

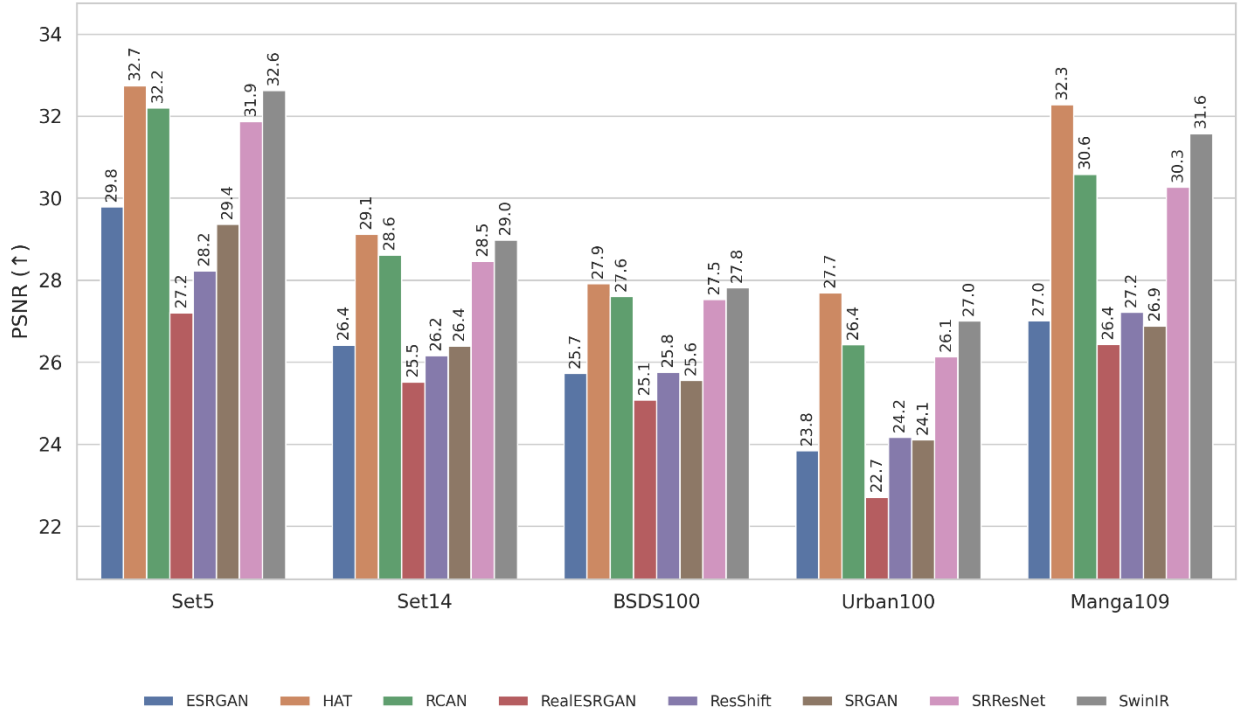


Рисунок 4.1 – Показники PSNR для різних архітектур та датасетів

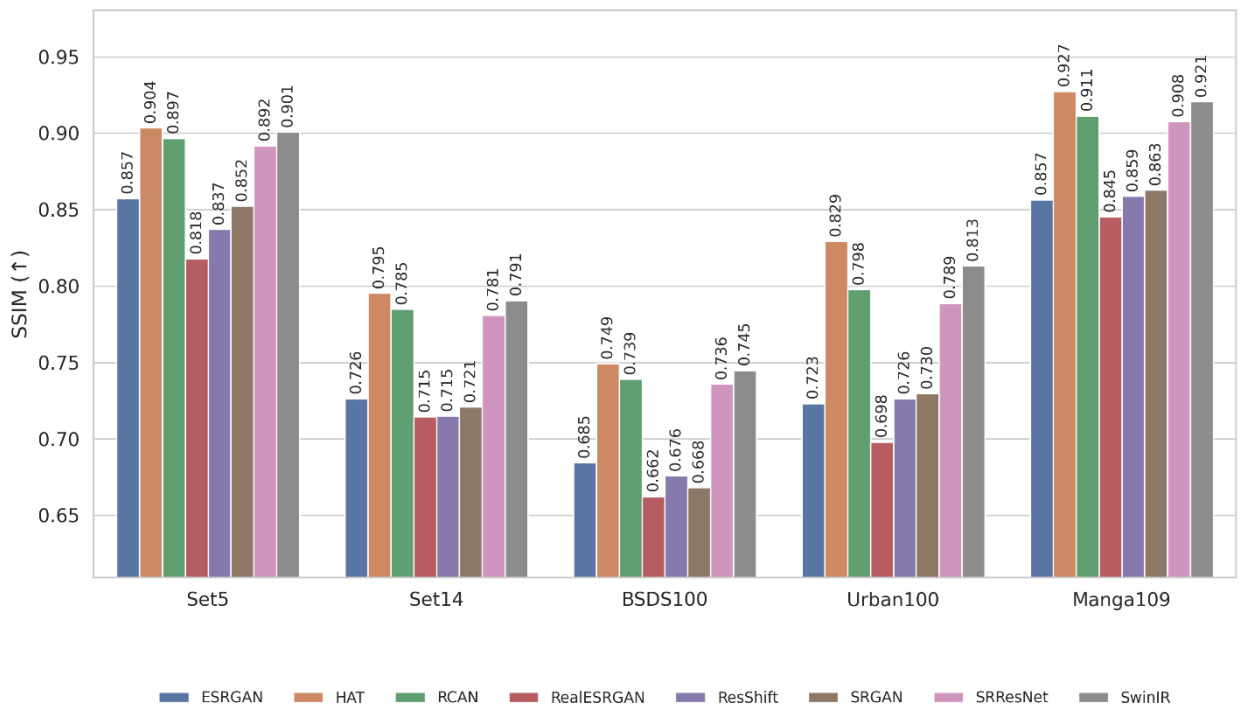


Рисунок 4.2 – Показники SSIM для різних архітектур та датасетів

На противагу математичним методам, аналіз перцептивної метрики LPIPS (рисунок 4.3), яка вимірює відстань у просторі ознак глибоких мереж, виявив домінування генеративно-змагальних мереж (GAN). На наборі Set5 найкращі (найменші) значення LPIPS продемонстрували моделі SRGAN (0,094) та ESRGAN (0,097). Слід зазначити, що хоча ці моделі значно поступаються трансформерам у PSNR (наприклад, SRGAN має лише 29,37 дБ), вони забезпечують набагато вищу візуальну якість. Наявність дискримінатора в архітектурі GAN змушує генератор синтезувати фотореалістичні мікротекстури та "галюцинувати" високі частоти, що сприймається людським зором як чітке та деталізоване зображення, навіть якщо згенеровані пікселі не ідеально збігаються з оригіналом.

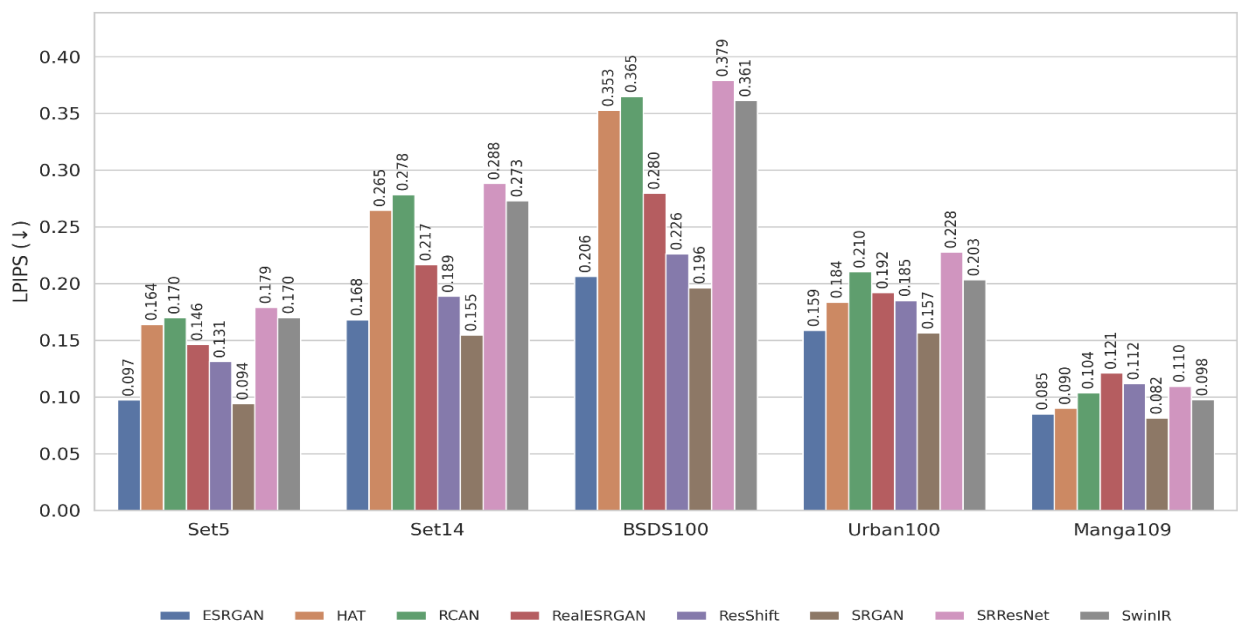


Рисунок 4.3 – Показники LPIPS для різних архітектур та датасетів

Найбільш прогресивні результати в категоріях естетичної якості та природності було зафіксовано при аналізі безреференсних метрик CLIPQA (рисунок 4.4) та MUSIQ (рисунок 4.5). Тут лідируючу позицію займає дифузійна архітектура ResShift. На наборі Set5 модель ResShift досягла показника CLIPQA 0,8 та MUSIQ 67,4, що є найвищим результатом серед усіх

досліджуваних зразків. Для порівняння, орієнтована на PSNR модель HAT має значно нижчі бали за цими критеріями (CLIPQA 0,6 та MUSIQ 61,1).

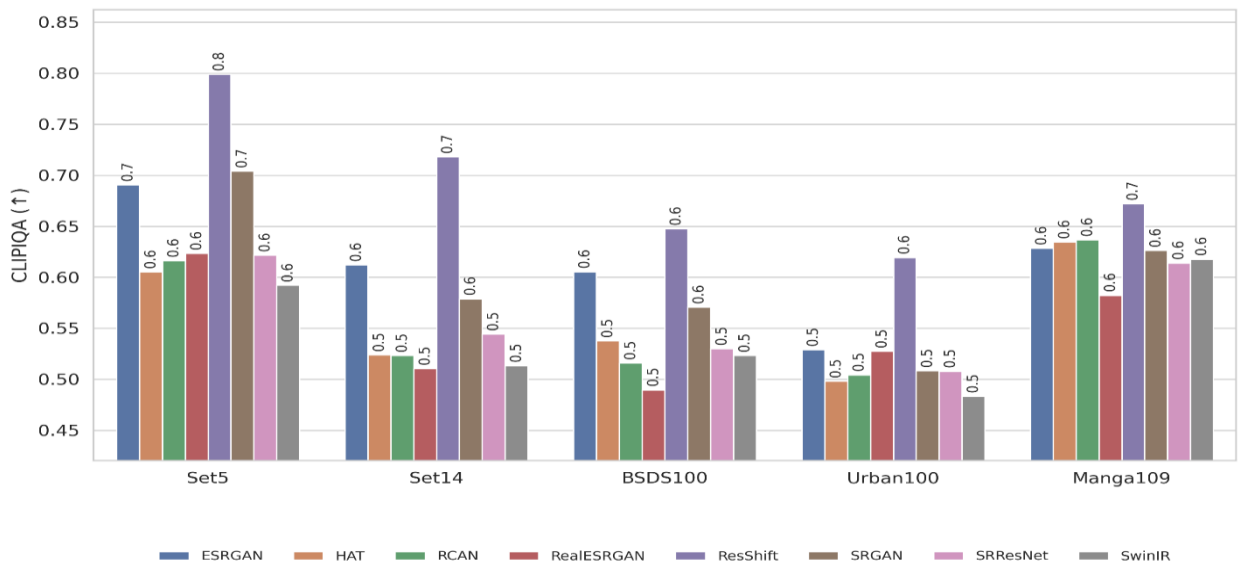


Рисунок 4.4 – Показники CLIPQA для різних архітектур та датасетів

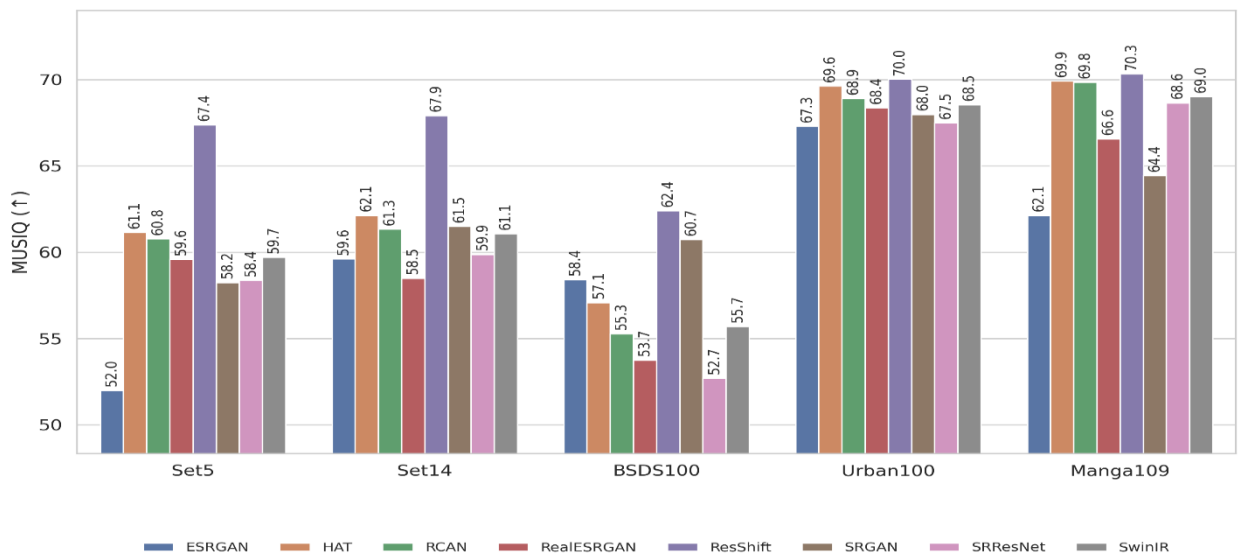


Рисунок 4.5 – Показники MUSIQ для різних архітектур та датасетів

Фізичний зміст переваги дифузійного підходу (ResShift) полягає в ітеративному процесі відновлення сигналу з шуму, що дозволяє моделі синтезувати складні, природні деталі з високою варіативністю. Алгоритми MUSIQ та CLIPQA, що базуються на сприйнятті, оцінюють такі деталі як ознаку високої якості та відсутності характерних для CNN "пластикових"

артефактів. Таким чином, експериментально підтверджено існування фундаментального компромісу (trade-off): моделі, оптимізовані під L1/L2 (HAT, SwinIR, RCAN), гарантують математичну вірність контурів, тоді як GAN та дифузійні моделі (ESRGAN, ResShift) забезпечують візуальну перевагу за рахунок синтезу нових текстур.

Окремої уваги потребує інтерпретація результатів моделі Real-ESRGAN, яка за кількісними показниками PSNR та SSIM часто поступається не лише трансформерним архітектурам, а й класичній моделі ESRGAN. Слід наголосити, що такі результати не є свідченням архітектурних недоліків, а обумовлені методологічною розбіжністю між процесом навчання моделі та умовами бенчмаркінгу.

Архітектура Real-ESRGAN була розроблена спеціально для вирішення задачі Blind SR (Super-Resolution з невідомими параметрами спотворення) і навчена на основі складної моделі деградації вищого порядку, яка імітує реальні умови зйомки: комбінації шумів сенсора, артефактів стиснення JPEG та нелінійного розмиття. У межах даного експериментального дослідження тестування проводилося на еталонних наборах даних (Set5, Set14, Urban100), де зображення низької роздільної здатності отримані шляхом "чистої" бікубічної інтерполяції без сторонніх завад.

За таких умов Real-ESRGAN намагається компенсувати складні спотворення, які фактично відсутні у вхідних даних, що призводить до надмірної генерації текстурних деталей та зниження математичної точності реконструкції відносно оригіналу. Це підтверджує висновок про те, що дана модель є високоефективним інструментом для реставрації реальних низькоякісних фотографій, проте її використання для відновлення зображень, що пройшли стандартний процес даунсемплінгу, не є оптимальним з точки зору метричної відповідності еталону.

Для наочного підтвердження результатів кількісного аналізу та перевірки ефективності роботи модуля visualizer.py у режимі compare, було

згенеровано порівняльні колажі. Вони дозволяють візуально оцінити якість реконструкції високочастотних деталей різними архітектурами.

На рисунку 4.6 наведено результати повнорозмірного збільшення зображення з набору даних Set5. Як можна помітити, базова бікубічна інтерполяція очікувано демонструє сильне розмиття та втрату деталей. Моделі, оптимізовані за метрикою PSNR (RCAN, SwinIR, SRResNet), згладжують краї крил метелика. Натомість генеративно-змагальні мережі (ESRGAN, Real-ESRGAN) та дифузійна архітектура ResShift синтезують значно чіткішу та контрастнішу текстуру, яка суб'єктивно виглядає більш природною, що підтверджує їх лідерство за метриками LPIPS та CLIPQA.

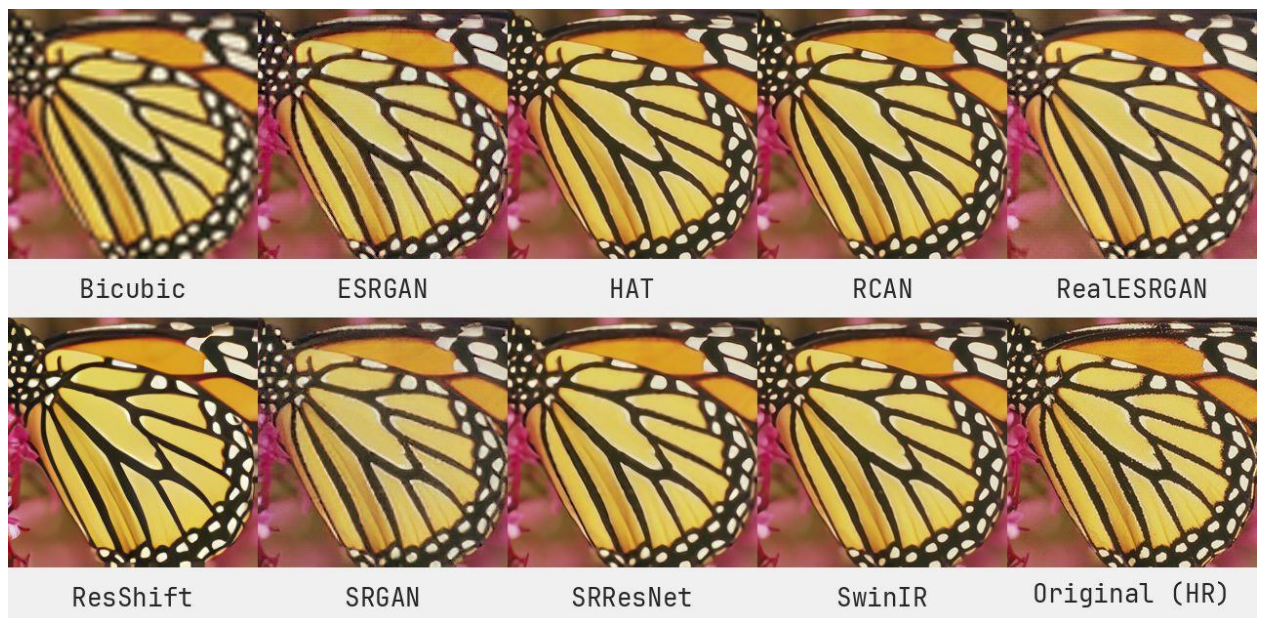


Рисунок 4.6 – Візуальне порівняння повнорозмірної генерації

З метою більш глибокого аналізу поведінки архітектур на складних геометричних паттернах, було використано функціонал генерації колажу з виділенням локальної області інтересу (Bounding Box). На рисунку 4.7 наведено реконструкцію залізничної станції з датасету Urban100.

Даний приклад яскраво ілюструє архітектурні обмеження деяких моделей. Складні періодичні структури (віконні рами та решітки) викликають ефект аліасингу (aliasing) та геометричні викривлення. Згорткові мережі, такі

як SRGAN, не здатні коректно відновити паралельність ліній, утворюючи характерні діагональні артефакти. Натомість моделі HAT та SwinIR, завдяки глобальному рецептивному полю механізму Self-Attention, найкраще справляються з геометричною узгодженістю ліній, наближаючись до еталонного зображення. Дифузійна модель ResShift, незважаючи на високу природність, у цьому конкретному випадку додає незначний шумовий патерн до однорідних ділянок скла.

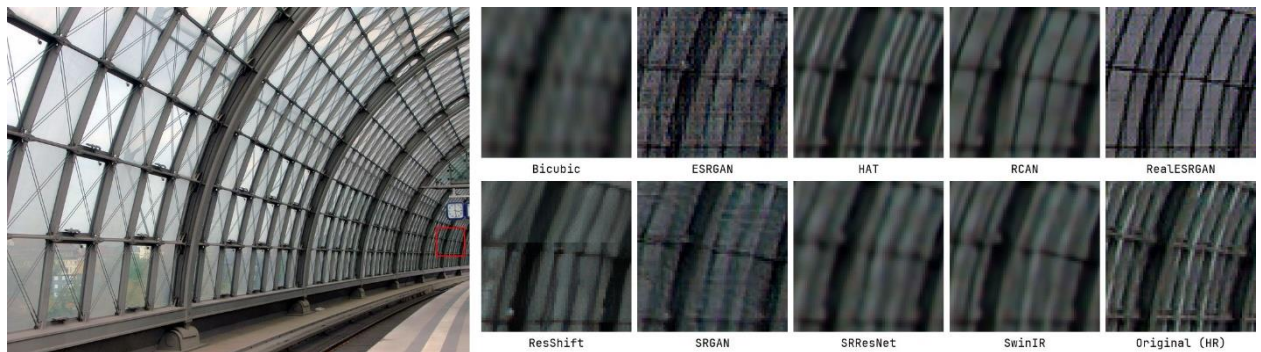


Рисунок 4.7 – Аналіз геометричних артефактів за допомогою локального фрагмента

Таким чином, інструментарій візуального порівняння, інтегрований у розроблений програмний комплекс, дозволяє не лише спиратися на "сухі" математичні цифри, а й проводити якісний експертний аналіз специфічних дефектів та особливостей кожної нейромережевої архітектури.

4.5 Дослідження апаратної ефективності та швидкодії моделей

Для визначення практичної придатності досліджуваних архітектур нейронних мереж у реальних умовах експлуатації критично важливим є аналіз їхньої апаратної ефективності. У межах даного підрозділу було проведено оцінювання моделей за двома ключовими критеріями: обсягом споживаної відеопам'яті (VRAM) та часом виконання інференсу (Inference Time). Аналіз

цих параметрів дозволяє визначити межу між теоретичною якістю реконструкції та реальними обчислювальними витратами.

Аналіз споживання відеопам'яті (рисунок 4.8) дозволяє виявити архітектурні обмеження моделей при роботі з зображеннями різної роздільної здатності. Згідно з отриманими даними, дифузійна модель ResShift виявилася найбільш ресурсомісткою серед усіх тестованих зразків – вона потребує стабільно високого обсягу пам'яті на рівні 4,1 ГБ навіть на відносно невеликих наборах даних (Set5 та Set14). Така особливість зумовлена специфікою ітеративного процесу відновлення сигналу та необхідністю зберігання проміжних станів дифузії.

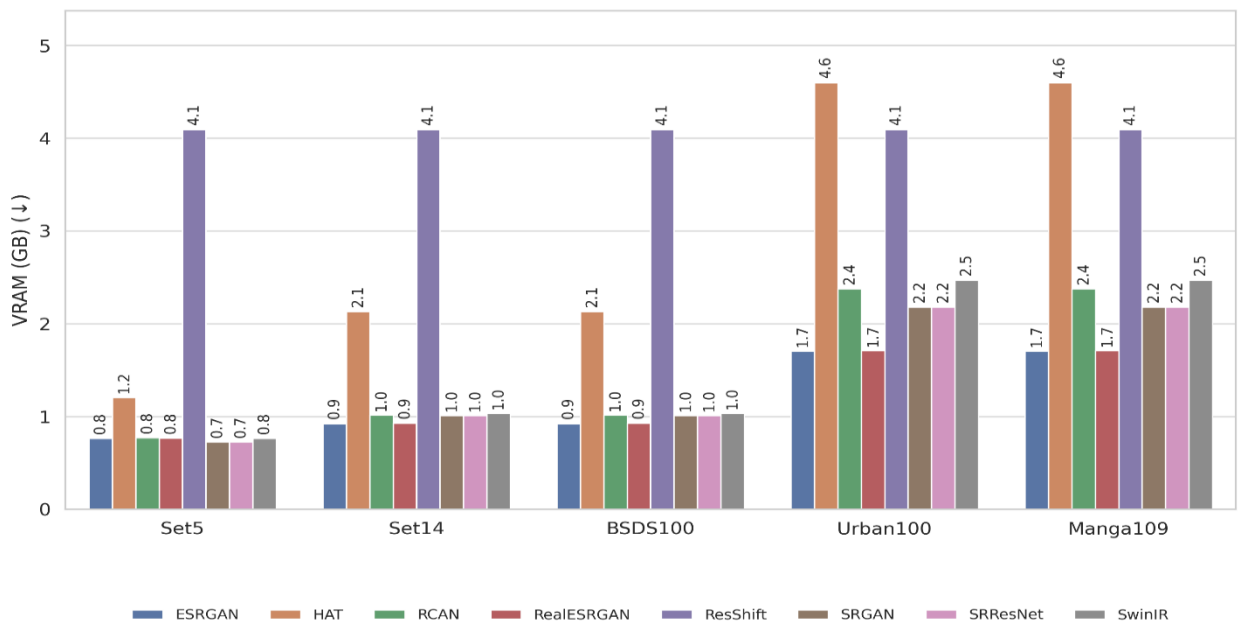


Рисунок 4.8 – Споживання відеопам'яті на тестових наборах даних

Значну динаміку споживання ресурсів продемонстрував гібридний трансформер HAT. Якщо на зображеннях низької роздільної здатності (Set5) він споживає близько 1,2 ГБ, то при переході до складніших датасетів Urban100 та Manga109 споживання пам'яті зростає до пікового значення 4,6 ГБ. Це свідчить про низьку масштабованість даної архітектури при роботі з великими вхідними даними, що пояснюється квадратичною залежністю

обчислювальної складності механізму Self-Attention від розміру вхідного тензора.

На противагу важковаговим рішенням, класичні згорткові архітектури (RCAN, SRResNet) та генеративно-змагальні мережі (ESRGAN, SRGAN, Real-ESRGAN) демонструють високий ступінь оптимізації. Більшість цих моделей на датасетах Set5 та Set14 споживають менше 1 ГБ відеопам'яті. Зокрема, моделі ESRGAN та Real-ESRGAN потребують лише 1,7 ГБ при роботі з важкими датасетами Urban100, що в три рази менше за показники НАТ, попри їхню здатність генерувати деталізовані текстури.

Дослідження швидкодії (рисунок 4.9) підтверджує значний розрив у продуктивності між різними типами архітектур. Найнижчу швидкість роботи продемонструвала модель ResShift: час інференсу для неї становить від 0,55 секунд (Set5) до 3,25 секунд (Manga109) на одне зображення. Повільність архітектури ResShift пояснюється ітеративною природою дифузійних процесів, що вимагають багаторазового проходження даних через мережу для формування фінального результату. Модель НАТ також демонструє низьку швидкодію, досягаючи показника в 1 секунду на Manga109, що знову підтверджує обчислювальну складність механізмів уваги у великих масштабах.

Найкращі результати швидкодії показали мережі CNN та GAN (RCAN, ESRGAN, SRGAN, SRResNet), що працюють неймовірно швидко. Наприклад, час інференсу для SRResNet на датасеті Set5 складає всього 0,008 секунд, а для ESRGAN – 0,048 секунд. Такі показники дозволяють інтегрувати ці архітектури у програмні комплекси, що функціонують у режимі реального часу, такі як системи відеострімінгу або онлайн редактори контенту.

Узагальнюючи результати апаратного профілювання, було зроблено висновок про високу "ціну", яку доводиться платити за генерацію високоякісних текстур у випадку трансформерів та дифузійних моделей. Хоча НАТ та ResShift забезпечують виняткову перцептивну якість, їхні вимоги до VRAM та часу обробки суттєво обмежують сферу їхнього застосування

системами з потужними обчислювальними ресурсами. Водночас оптимізовані GAN-архітектури залишаються найбільш збалансованим вибором для практичного впровадження, забезпечуючи високу швидкість при помірних апаратних витратах. Детальні результати обчислень та апаратні метрики зафіксовані у файлі `benchmark_results.xlsx`, який створюється автоматично при використанні режиму `benchmark`.

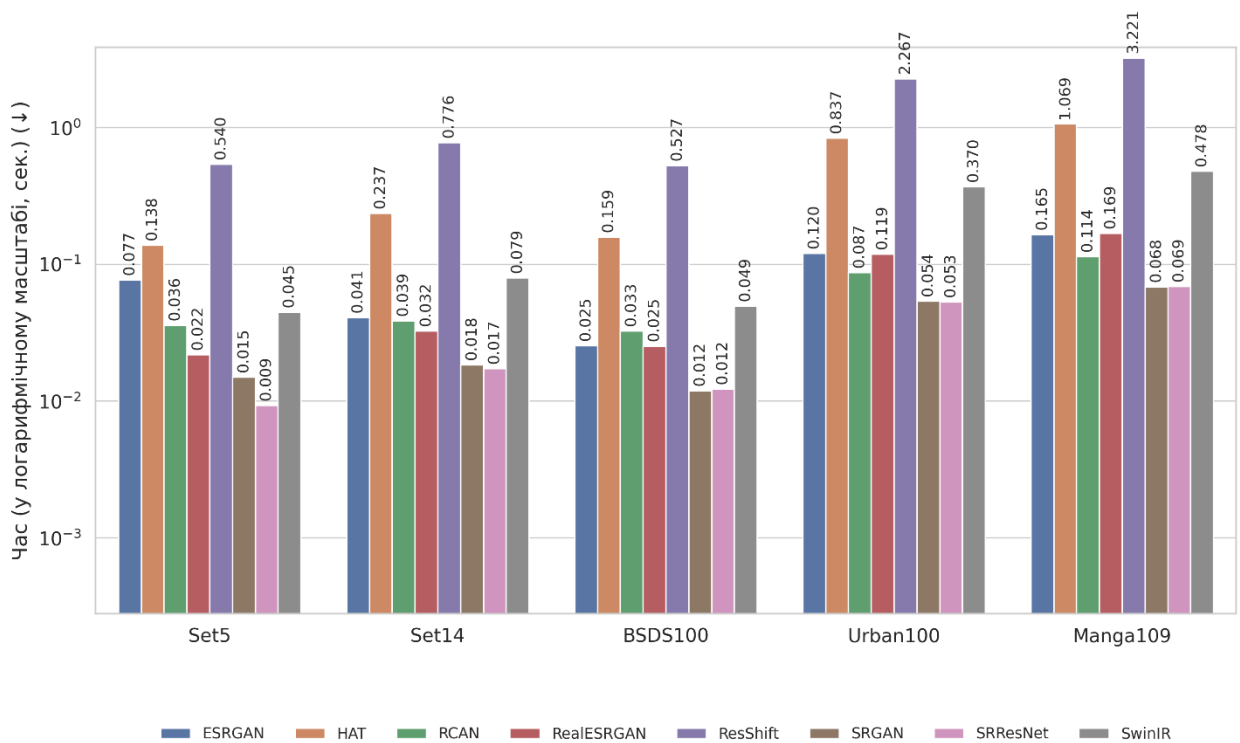


Рисунок 4.9 – Порівняльний аналіз швидкодії нейромережових архітектур

4.6 Оцінка балансу між якістю генерації та обчислювальною складністю

На основі проведених комплексних випробувань восьми архітектур нейронних мереж у межах даного підрозділу здійснено фінальний синтез результатів дослідження. Ключовим аспектом аналізу є виявлення фундаментального компромісу між якістю реконструкції та обчислювальними витратами, відомого в галузі комп'ютерного зору як компроміс між сприйняттям та спотворенням (Perception-Distortion trade-off).

Аналіз узагальненої радарної діаграми балансу характеристик моделей (рисунок 4.10) наочно демонструє відсутність універсального рішення, яке б одночасно забезпечувало екстремальні значення за всіма метриками.

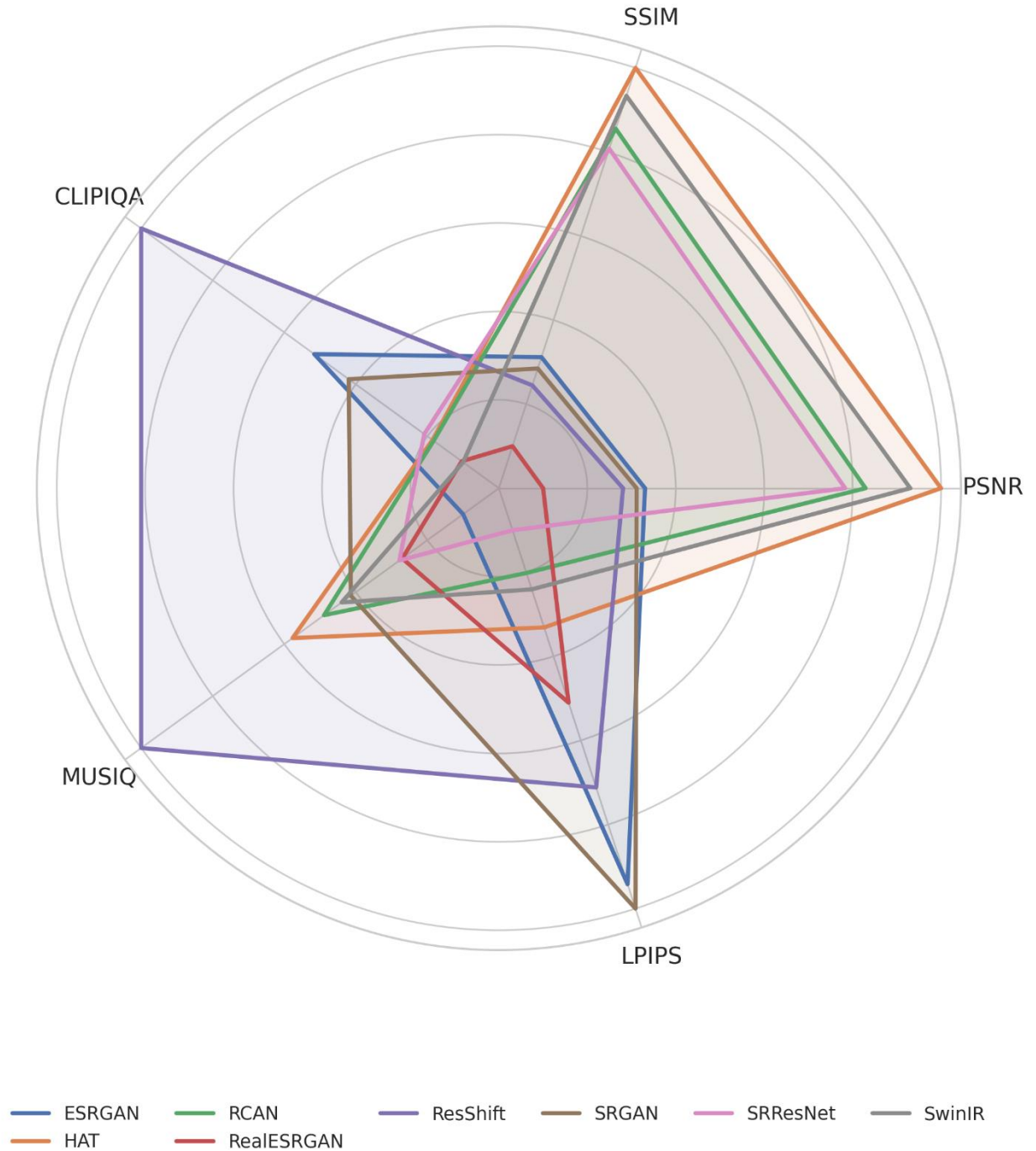


Рисунок 4.10 – Радарна діаграма балансу перцептивної якості та математичної точності моделей

Чітко виділено розподіл досліджуваних архітектур на дві основні функціональні категорії:

– напрямок мінімізації спотворень, до якого належать трансформерні архітектури HAT і SwinIR, CNN з механізмом уваги RCAN та класична залишкова мережа SRResNet; дані моделі максимально розширюють багатокутник характеристик у бік осей PSNR та SSIM, забезпечуючи високу математичну точність відновлення структури сигналу;

– напрямок перцептивної якості, до якого входять дифузійна модель ResShift та генеративно-змагальні мережі ESRGAN, Real-ESRGAN і SRGAN; ці архітектури зміщують акцент у бік метрик MUSIQ, CLIPQA та LPIPS, що свідчить про їхню здатність генерувати фотореалістичні деталі, попри вище значення попиксельної похибки.

Дослідження взаємозв'язку між швидкістю інференсу, якістю генерації та складністю архітектур відображено на бульбашкових діаграмах (рисунок 4.11 та рисунок 4.12). Розмір бульбашки на графіках прямо пропорційний кількості параметрів моделі, а положення на осях визначає баланс між часом обробки (вісь X) та якісним показником (вісь Y).

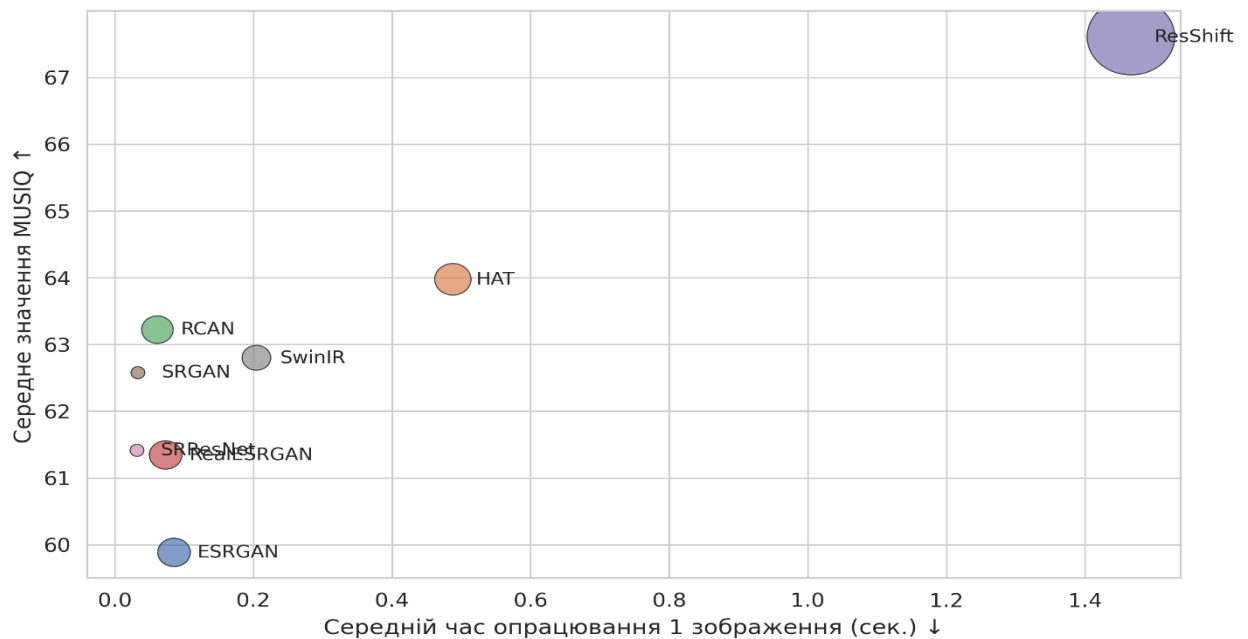


Рисунок 4.11 – Залежність перцептивної якості від часу інференсу та об'єму параметрів

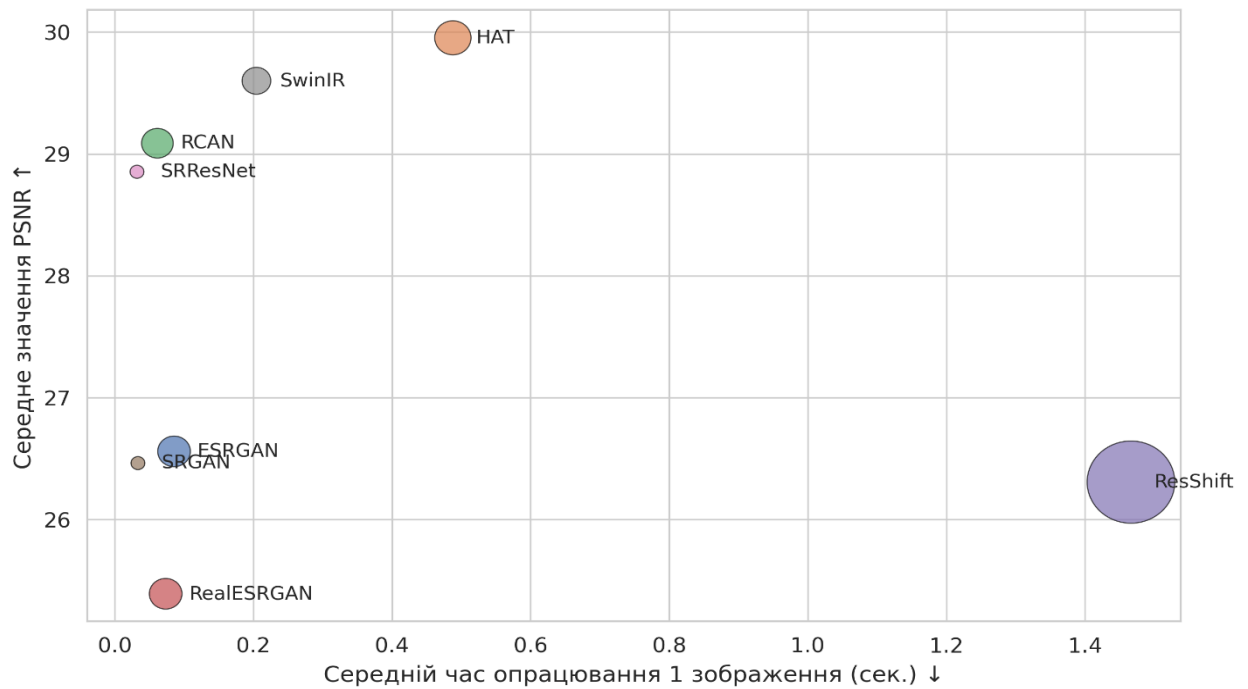


Рисунок 4.12 – Залежність математичної точності від часу інференсу та об'єму параметрів

Аналіз залежності швидкості від перцептивної якості MUSIQ (рисунок 4.11) вказує на те, що модель ResShift володіє найбільшою кількістю параметрів, що зумовлює її зміщення в область найвищої якості при значних часових витратах. Модель HAT займає проміжну позицію, пропонуючи збалансоване поєднання високої якості та помірної швидкодії.

Графік залежності від показника PSNR (рисунок 4.12) підтверджує ефективність архітектур HAT та SwinIR, які при оптимальній кількості параметрів забезпечують домінування у верхній лівій частині діаграми, що відповідає високій точності при прийнятній швидкості. Моделі сімейства GAN та класична SRResNet групуються в області найвищої швидкодії, проте демонструють нижчі показники математичної точності.

Спираючись на результати аналізу, сформовано наступні практичні рекомендації щодо вибору нейромережевої архітектури залежно від цільової галузі застосування:

- для систем реального часу та мобільних пристроїв рекомендовано використання моделей ESRGAN або SRGAN, оскільки вони забезпечують

високу якість текстур при мінімальному часі інференсу та низькому споживанні відеопам'яті;

- для наукової та медичної візуалізації або обробки супутникових знімків рекомендовано застосування RCAN або SwinIR, що гарантують найвищу структурну точність та мінімізацію небажаних візуальних артефактів;

- для студійної обробки фотографій та реставрації архівного контенту рекомендовано ResShift або HAT, де пріоритетом є абсолютна якість відновлення, а час обробки не є критичним фактором.

Підсумовуючи, слід зазначити, що розроблений програмний комплекс SRBenchmark дозволив провести глибоке багатофакторне дослідження сучасних методів збільшення роздільної здатності зображень. Виявлені закономірності та сформовані рекомендації створюють надійне підґрунтя для усвідомленого вибору архітектури нейронної мережі під конкретні технічні завдання, забезпечуючи досягнення необхідного балансу між обчислювальною складністю та якісними характеристиками результату. На основі проведеного аналізу встановлено, що вибір конкретної моделі повинен базуватися на пріоритетності точності або фотореалістичності в межах конкретного програмного рішення.

4.7 Підсумки розділу

У межах четвертого розділу було успішно проведено фінальний етап роботи – розгортання, тестування та експериментальне дослідження створеного програмного комплексу. Сформовано чітку методологію проведення бенчмаркінгу, яка дозволила отримати об'єктивні дані щодо роботи восьми архітектур нейронних мереж на п'яти стандартизованих наборах даних.

За результатами проведеного аналізу було практично підтверджено існування фундаментального компромісу між математичною точністю та

перцептивною якістю згенерованих зображень. Дослідним шляхом встановлено, що моделі на базі візуальних трансформерів забезпечують найвищу структурну точність, тоді як генеративно-змагальні та дифузійні архітектури здатні синтезувати набагато природніші та деталізовані текстури. Окрім оцінювання візуальної складової, виконано детальне профілювання, яке дозволило визначити реальні вимоги алгоритмів до обсягу відеопам'яті та швидкості обробки. Завдяки комплексному підходу до тестування вдалося сформулювати практичні рекомендації щодо доцільності впровадження конкретних архітектур у різних експлуатаційних сценаріях.

Таким чином, усі поставлені завдання експериментального етапу виконано в повному обсязі, а створений інструментарій повністю довів свою ефективність, стабільність та придатність для використання в реальних умовах. Завершивши всі передбачені стадії дослідження – від теоретичного аналізу і проєктування системи до її практичної перевірки, останнім логічним кроком залишається підбиття загальних підсумків кваліфікаційної роботи.

ВИСНОВКИ

За результатами виконання дипломної кваліфікаційної роботи розв'язано актуальне науково-практичне завдання, що полягає у зменшенні витрат часу та підвищенні точності порівняльного аналізу архітектур нейронних мереж для збільшення роздільної здатності зображень. Відповідно до поставлених завдань дослідження зроблено такі висновки:

- проаналізовано існуючі методи та архітектури нейронних мереж для збільшення роздільної здатності зображень; досліджено еволюцію підходів від базових згорткових мереж до сучасних генеративно-змагальних, трансформерних та дифузійних моделей; встановлено, що класичні метрики оцінювання не здатні повною мірою відобразити якість візуального контенту, що зумовило необхідність використання багатовимірного підходу до тестування;

- спроектовано модульну архітектуру програмного забезпечення для стандартизованого бенчмаркінгу, а застосування принципу відкритості-закритості дозволило створити гнучку систему, яка забезпечує повну ізоляцію обчислювального ядра від логіки обробки та дозволяє інтегрувати нові архітектури через стандартизовані інтерфейси і конфігураційні файли без зміни основного коду;

- розроблено програмні модулі для обробки даних, розрахунку метрик якості та створення графіків; для подолання апаратних обмежень обсягу відеопам'яті успішно реалізовано алгоритм тайлінгового інференсу; створено підсистему для обчислення класичних (PSNR, SSIM) та перцептивних (LPIPS, CLIPQA, MUSIQ) метрик, а також автоматичної генерації порівняльних колажів і аналітичних діаграм;

- здійснено тестування розробленого програмного комплексу та проведено порівняльний аналіз обраних архітектур на п'яти стандартизованих наборах даних; експериментально підтверджено існування компромісу між математичною точністю, де лідирують трансформерні моделі, та

перцептивною якістю, де перевагу мають дифузійні та генеративно-змагальні мережі; на основі апаратного профілювання сформовано практичні рекомендації щодо вибору моделей залежно від доступних ресурсів та цільового застосування.

Розроблений програмний комплекс є повністю завершеним інженерним продуктом, придатним для використання у дослідницьких та виробничих циклах для комплексного та об'єктивного оцінювання систем комп'ютерного зору.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bilinear interpolation : technical guide. URL: <https://x-engineer.org/bilinear-interpolation/> (date of access: 11.05.2026).
2. Bicubic Interpolation : image scaling article. URL: <https://image-scaler.com/blog/bicubic-interpolation/> (date of access: 11.05.2026).
3. Fardo F.A., Conforto V.H., Oliveira F.C., Rodrigues P.S. A Formal Evaluation of PSNR as Quality Measurement Parameter for Image Segmentation Algorithms. arXiv preprint arXiv:1501.00092. 2015. URL: <https://arxiv.org/abs/1605.07116> (date of access: 11.05.2026).
4. Wang Z., Bovik A. C., Sheikh H. R., Simoncelli E. P. Image Quality Assessment: From Error Visibility to Structural Similarity. IEEE Transactions on Image Processing. 2004. Vol. 13, no. 4. P. 600–612. URL: <https://arxiv.org/abs/2006.13846> (date of access: 11.05.2026).
5. Zhang R., Isola P., Efros A. A., Shechtman E., Wang O. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. arXiv preprint arXiv:1801.03924. 2018. URL: <https://arxiv.org/abs/1801.03924> (date of access: 11.05.2026).
6. Wang J., Chan K. C., Loy C. C. Exploring CLIP for Assessing the Perceptual Quality of Images. arXiv preprint arXiv:2207.12396. 2022. URL: <https://arxiv.org/abs/2207.12396> (date of access: 11.05.2026).
7. Ke J. et al. MUSIQ: Multi-scale Image Quality Transformer. Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). 2021. P. 5148–5157. URL: <https://arxiv.org/abs/2108.05997> (date of access: 11.05.2026).
8. Zhao H., Gallo O., Frosio I., Kautz J. Loss Functions for Image Restoration with Neural Networks. ArXiv. URL: <https://arxiv.org/abs/1511.08861> (date of access: 05.05.2026).

9. Dong C., Loy C. C., He K., Tang X. Image Super-Resolution Using Deep Convolutional Networks. ArXiv. URL: <https://arxiv.org/abs/1501.00092> (date of access: 05.05.2026).
10. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. ArXiv. URL: <https://arxiv.org/abs/1512.03385> (date of access: 05.05.2026).
11. He K., Zhang X., Ren S., Sun J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. ArXiv. URL: <https://arxiv.org/abs/1502.01852> (date of access: 05.05.2026).
12. Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. ArXiv. URL: <https://arxiv.org/abs/1502.03167> (date of access: 05.05.2026).
13. Zhang Y., Li K., Li K., Wang L., Zhong B., Fu Y. Image Super-Resolution Using Very Deep Residual Channel Attention Networks. ArXiv. URL: <https://arxiv.org/abs/1807.02758> (date of access: 05.05.2026).
14. Odena A., Dumoulin V., Olah C. Deconvolution and Checkerboard Artifacts. Distill. URL: <https://distill.pub/2016/deconv-checkerboard/> (date of access: 05.05.2026).
15. Ledig C., Theis L., Huszar F. та ін. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. ArXiv. URL: <https://arxiv.org/abs/1609.04802> (date of access: 05.05.2026).
16. Wang X., Yu K., Wu S. та ін. ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. ArXiv. URL: <https://arxiv.org/abs/1809.00219> (date of access: 05.05.2026).
17. Wang X., Xie L., Dong C., Shan Y. Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data. ArXiv. URL: <https://arxiv.org/abs/2107.10833> (date of access: 05.05.2026).
18. Ronneberger O., Fischer P., Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. ArXiv. URL: <https://arxiv.org/abs/1505.04597> (date of access: 05.05.2026).

19. Dosovitskiy A., Beyer L., Kolesnikov A. та ін. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. ArXiv. URL: <https://arxiv.org/abs/2010.11929> (date of access: 05.05.2026).
20. Liu Z., Lin Y., Cao Y. та ін. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. ArXiv. URL: <https://arxiv.org/abs/2103.14030> (date of access: 05.05.2026).
21. Liang J., Cao J., Sun G. та ін. SwinIR: Image Restoration Using Swin Transformer. ArXiv. URL: <https://arxiv.org/abs/2108.10257> (date of access: 05.05.2026).
22. Chen X., Wang X., Zhou J., Dong C. Activating More Pixels in Image Super-Resolution Transformer. ArXiv. URL: <https://arxiv.org/abs/2205.04437> (date of access: 05.05.2026).
23. Chen X., Wang X., Zhang W. та ін. HAT: Hybrid Attention Transformer for Image Restoration. ArXiv. URL: <https://arxiv.org/abs/2309.05239> (date of access: 05.05.2026).
24. Sohl-Dickstein J., Weiss E. A., Maheswaranathan N., Ganguli S. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. ArXiv. URL: <https://arxiv.org/abs/1503.03585> (date of access: 05.05.2026).
25. Ho J., Jain A., Abbeel P. Denoising Diffusion Probabilistic Models. ArXiv. URL: <https://arxiv.org/abs/2006.11239> (date of access: 05.05.2026).
26. Song J., Meng C., Ermon S. Denoising Diffusion Implicit Models. ArXiv. URL: <https://arxiv.org/abs/2010.02502> (date of access: 05.05.2026).
27. Esser P., Rombach R., Ommer B. Taming Transformers for High-Resolution Image Synthesis. ArXiv. URL: <https://arxiv.org/abs/2012.09841> (date of access: 05.05.2026).
28. Rombach R., Blattmann A., Lorenz D. та ін. High-Resolution Image Synthesis with Latent Diffusion Models. ArXiv. URL: <https://arxiv.org/abs/2112.10752> (date of access: 05.05.2026).

29. Yue Z., Wang J., Loy C. C. ResShift: Efficient Diffusion Model for Image Super-resolution by Residual Shifting. ArXiv. URL: <https://arxiv.org/abs/2307.12348> (date of access: 05.05.2026).
30. DIV2K Dataset : official website. URL: <https://data.vision.ee.ethz.ch/cvl/DIV2K/> (дата звернення: 11.05.2026).
31. DF2K Dataset : Kaggle repository. URL: <https://www.kaggle.com/datasets/anvul204/df2kdata> (дата звернення: 11.05.2026).
32. ImageNet : large-scale hierarchical image database. URL: <https://www.image-net.org/> (дата звернення: 11.05.2026).
33. BSD100, Set5, Set14, Urban100 Dataset : Figshare archive. URL: https://figshare.com/articles/dataset/BSD100_Set5_Set14_Urban100/21586188 (дата звернення: 11.05.2026).
34. Manga109 : dataset for manga analysis. URL: <https://manga109.github.io/manga109-project-website/en/index.html> (дата звернення: 11.05.2026).

ДОДАТОК А
Текст програми

A.1 Текст файла evaluator.py

```

import importlib
import io
import time
import warnings
from contextlib import redirect_stdout
from pathlib import Path
from typing import Literal, cast

import numpy as np
import pyiqa
import torch
from ptflops import get_model_complexity_info
from safetensors.torch import load_file
from torch import Tensor, nn
from torch.nn import functional as F
from torch.utils.data import DataLoader
from torchvision.io import ImageReadMode, read_image
from tqdm import tqdm

from core_utils import ModelConfig, calculate_psnr, calculate_ssim, imresize
from dataset import BenchmarkDataset

class Evaluator:
    def __init__(
        self,
        config_path: Path,
        device: Literal["cpu", "cuda"] = "cpu",
        tile_size: int | None = None,
        tile_overlap: int = 32,
        num_workers: int = 4,
        prefetch_factor: int = 4,
    ) -> None:
        self.config = ModelConfig.from_yaml(config_path)

        self.device = torch.device(device)
        self.scaling_factor = self.config.scaling_factor
        self.tile_size = tile_size
        self.tile_overlap = tile_overlap
        self.num_workers = num_workers
        self.prefetch_factor = prefetch_factor

        self._init_model(config_path)

```

```

self._init_perceptual_metrics()

if self.device.type == "cuda":
    if torch.cuda.is_bf16_supported():
        self.autocast_dtype = torch.bfloat16
    else:
        self.autocast_dtype = torch.float16
else:
    self.autocast_dtype = torch.bfloat16

def _init_model(self, config_path: Path) -> None:
    self.model_name = config_path.parent.name
    self.module_path = f"models.{self.model_name}.model"

    self.model_class = getattr(importlib.import_module(self.module_path),
self.model_name)
    self.model = cast(nn.Module,
self.model_class(**self.config.model_params)).to(self.device)

    self.weights_path = config_path.parent / self.config.weights_path

    if self.weights_path.suffix in [".pt", ".pth"]:
        self.model.load_state_dict(torch.load(self.weights_path,
map_location=self.device, weights_only=True))
    elif self.weights_path.suffix == ".safetensors":
        self.model.load_state_dict(load_file(self.weights_path,
device=str(self.device)))

    self.model.eval()

def _init_perceptual_metrics(self) -> None:
    warnings.filterwarnings("ignore", category=UserWarning)

    with redirect_stdout(io.StringIO()):
        self.perceptual_metrics = {
            "lpips": pyiqa.create_metric("lpips", as_loss=False, device=self.device),
            "clipiqa": pyiqa.create_metric("clipiqa", as_loss=False,
device=self.device),
            "musiq": pyiqa.create_metric("musiq", as_loss=False,
device=self.device),
        }

    warnings.filterwarnings("default", category=UserWarning)

def _run_model_tiled(self, lr_img_tensor: Tensor) -> Tensor:

```

```

assert self.tile_size is not None

num_channels, lr_img_height, lr_img_width = lr_img_tensor.shape

sr_img_height = lr_img_height * self.scaling_factor
sr_img_width = lr_img_width * self.scaling_factor
sr_img_shape = (num_channels, sr_img_height, sr_img_width)

sr_accumulated_values = torch.zeros(sr_img_shape, dtype=torch.float32,
device="cpu")
sr_weight_map = torch.zeros(sr_img_shape, dtype=torch.float32,
device="cpu")

stride = self.tile_size - self.tile_overlap
height_steps = list(range(0, lr_img_height - self.tile_size, stride)) +
[lr_img_height - self.tile_size]
width_steps = list(range(0, lr_img_width - self.tile_size, stride)) +
[lr_img_width - self.tile_size]

if lr_img_height < self.tile_size:
    height_steps = [0]

if lr_img_width < self.tile_size:
    width_steps = [0]

pbar = tqdm(total=len(height_steps) * len(width_steps), desc="Processing
tiles", leave=False)

for height_step in height_steps:
    for width_step in width_steps:
        lr_height_end = min(height_step + self.tile_size, lr_img_height)
        lr_width_end = min(width_step + self.tile_size, lr_img_width)
        lr_height_start = max(0, lr_height_end - self.tile_size)
        lr_width_start = max(0, lr_width_end - self.tile_size)

        lr_img_patch = lr_img_tensor[:, lr_height_start:lr_height_end,
lr_width_start:lr_width_end]
        lr_img_patch = lr_img_patch.to(self.device)

        pad = 16
        lr_img_patch_padded = F.pad(lr_img_patch, (pad, pad, pad, pad),
mode="replicate")

        with torch.autocast(device_type=self.device.type,
dtype=self.autocast_dtype):

```

```

        sr_img_patch_padded = sr_img_patch_padded.float()

        sr_pad = pad * self.scaling_factor
        sr_img_patch = sr_img_patch_padded[:, sr_pad:-sr_pad, sr_pad:-sr_pad]

        sr_height_end = lr_height_end * self.scaling_factor
        sr_width_end = lr_width_end * self.scaling_factor
        sr_height_start = lr_height_start * self.scaling_factor
        sr_width_start = lr_width_start * self.scaling_factor

        sr_accumulated_values[:, sr_height_start:sr_height_end,
sr_width_start:sr_width_end] += sr_img_patch
        sr_weight_map[:, sr_height_start:sr_height_end,
sr_width_start:sr_width_end] += 1.0

        pbar.update(1)
        pbar.close()

    return sr_accumulated_values.div_(sr_weight_map.clamp(min=1e-6)).to(self.device)

def _run_model(self, lr_img_tensor: Tensor) -> Tensor:
    if self.tile_size is not None and self.tile_size > 0:
        return self._run_model_tiled(lr_img_tensor)

    window_size = self.config.window_size

    _, lr_img_height, lr_img_width = lr_img_tensor.shape
    padding_right = (window_size - (lr_img_width % window_size)) % window_size
    padding_bottom = (window_size - (lr_img_height % window_size)) % window_size

    if padding_right > 0 or padding_bottom > 0:
        lr_tensor_padded = F.pad(lr_img_tensor, (0, padding_right, 0, padding_bottom), mode="replicate")
    else:
        lr_tensor_padded = lr_img_tensor

    with torch.autocast(device_type=self.device.type, dtype=self.autocast_dtype):
        sr_tensor_padded = self.model(lr_tensor_padded.unsqueeze(0)).squeeze(0)

```

```

sr_tensor_padded = sr_tensor_padded.float()

if padding_right > 0 or padding_bottom > 0:
    sr_img_tensor = sr_tensor_padded[
        :, : lr_img_height * self.scaling_factor, : lr_img_width *
self.scaling_factor
    ]
else:
    sr_img_tensor = sr_tensor_padded

return sr_img_tensor

def _preprocess_img(self, img_path: Path) -> Tensor:
    img_tensor = read_image(str(img_path), mode=ImageReadMode.RGB)
    img_tensor = img_tensor.float() / 255.0

    return img_tensor.to(self.device)

def _init_dataloader(self, dataset_path: Path) -> DataLoader:
    dataset = BenchmarkDataset(dataset_path, scaling_factor=self.scaling_factor)
    dataloader = DataLoader(
        dataset=dataset,
        batch_size=1,
        shuffle=False,
        num_workers=self.num_workers,
        pin_memory=True if self.device.type == "cuda" else False,
        prefetch_factor=self.prefetch_factor,
        persistent_workers=True if self.num_workers > 0 else False,
    )

    return dataloader

def _evaluate_one_dataset(
    self,
    dataloader: DataLoader,
    dataset_name: str,
) -> dict[str, float]:
    scores = {"psnr": [], "ssim": [], "lpips": [], "clipiq": [], "musiq": [], "time": [],
"vram": []}

    starter = torch.cuda.Event(enable_timing=True)
    ender = torch.cuda.Event(enable_timing=True)
    is_cuda = self.device.type == "cuda"

    for batch in tqdm(

```

```

        dataloader, desc=f"Evaluating {self.model_name} on {dataset_name}",
        total=len(dataloader), leave=False
    ):
        lr_img_tensor = batch["lr"][0].to(device=self.device, non_blocking=True)
        hr_img_tensor = batch["hr"][0].to(device=self.device, non_blocking=True)

        if is_cuda:
            torch.cuda.synchronize()
            torch.cuda.reset_peak_memory_stats(self.device)
            starter.record()
        else:
            start_time = time.perf_counter()

        sr_img_tensor = self._run_model(lr_img_tensor)

        if is_cuda:
            ender.record()
            torch.cuda.synchronize()
            inference_time = starter.elapsed_time(ender) / 1000.0

        scores["vram"].append(torch.cuda.max_memory_reserved(self.device) /
(1024 * 1024 * 1024))
        else:
            inference_time = time.perf_counter() - start_time
            scores["vram"].append(0.0)

        scores["time"].append(inference_time)

        sr_img_tensor = sr_img_tensor.float().clamp(0, 1)

        scores["psnr"].append(calculate_psnr(sr_img_tensor,          hr_img_tensor,
crop_border=self.scaling_factor))
        scores["ssim"].append(calculate_ssim(sr_img_tensor,          hr_img_tensor,
crop_border=self.scaling_factor))

        sr_img_batch = sr_img_tensor.unsqueeze(0)
        hr_img_batch = hr_img_tensor.unsqueeze(0)

        scores["lpips"].append(self.perceptual_metrics["lpips"](sr_img_batch,
hr_img_batch).item())

        scores["clipiq"].append(self.perceptual_metrics["clipiq"](sr_img_batch).item())

        scores["musiq"].append(self.perceptual_metrics["musiq"](sr_img_batch).item())

```

```

return {
    "PSNR": float(np.mean(scores["psnr"])),
    "SSIM": float(np.mean(scores["ssim"])),
    "LPIPS": float(np.mean(scores["lpips"])),
    "CLIPQA": float(np.mean(scores["clipqa"])),
    "MUSIQ": float(np.mean(scores["musiq"])),
    "Time": float(np.mean(scores["time"])),
    "VRAM (GB)": float(np.max(scores["vram"])),
}

@torch.inference_mode()
def evaluate(self, dataset_paths: list[Path]) -> dict[str, dict[str, float]]:
    results = {}

    for dataset_path in dataset_paths:
        dataloader = self._init_dataloader(dataset_path)
        results[dataset_path.name] = self._evaluate_one_dataset(dataloader,
dataset_path.name)

    return results

@torch.inference_mode()
def upscale(self, img_tensor: Tensor) -> Tensor:
    return self._run_model(img_tensor.to(self.device))

@torch.inference_mode()
def upscale_downscaled(self, img_tensor: Tensor) -> Tensor:
    lr_img_tensor = imresize(img_tensor, scaling_factor=1 / self.scaling_factor,
antialiasing=True).to(self.device)

    return self._run_model(lr_img_tensor.squeeze(0))

def get_model_stats(self, input_shape: tuple[int, int, int] = (3, 64, 64)) -> dict[str,
str | float]:
    param_overrides = {
        "ResShift": 118.59 * 1e6,
    }

    try:
        macs, params = get_model_complexity_info(
            model=self.model,
            input_res=input_shape,
            as_strings=False,
            print_per_layer_stat=False,
            verbose=False,

```

```

    )

    if self.model_name in param_overrides:
        params = param_overrides[self.model_name]

    return {"model_name": self.model_name, "params": params, "flops": macs
* 2} # type: ignore

except Exception:
    fallback_params = param_overrides.get(self.model_name, 0.0)
    return {"model_name": self.model_name, "params": fallback_params,
"flops": 0.0}

```

A.2 Текст файлу **benchmark.py**

```

import gc
from pathlib import Path
from typing import Literal

import torch
from torch.nn import functional as F

from core_utils import ModelConfig, imresize
from evaluator import Evaluator
from logger import logger
from visualizer import Visualizer

class Benchmark:
    def __init__(
        self,
        device: Literal["cpu", "cuda"] = "cpu",
        tile_size: int | None = None,
        tile_overlap: int = 32,
        num_workers: int = 4,
        prefetch_factor: int = 4,
    ) -> None:
        self.device = device

        self.tile_size = tile_size
        self.tile_overlap = tile_overlap

        self.num_workers = num_workers
        self.prefetch_factor = prefetch_factor

```

```

self.visualizer = Visualizer()

def _init_evaluator(self, config_path: Path) -> Evaluator:
    return Evaluator(
        config_path=config_path,
        device=self.device,
        tile_size=self.tile_size,
        tile_overlap=self.tile_overlap,
        num_workers=self.num_workers,
        prefetch_factor=self.prefetch_factor,
    )

def evaluate(
    self,
    config_paths: list[Path],
    dataset_paths: list[Path],
    output_dir: Path | None = None,
) -> None:
    results = {}

    for config_path in config_paths:
        evaluator = self._init_evaluator(config_path)
        model_name = evaluator.model_name

        logger.info(f"[{model_name}] | Starting benchmark...")

        model_results = evaluator.evaluate(dataset_paths)
        results[model_name] = model_results

        stats = evaluator.get_model_stats((3, 64, 64))

        for dataset_name in model_results:
            model_results[dataset_name]["Params (M)"] = stats["params"] / 1e6 #
type: ignore
            model_results[dataset_name]["FLOPs (G)"] = stats["flops"] / 1e9 # type:
ignore

        del evaluator
        gc.collect()
        torch.cuda.empty_cache()

    self.visualizer.print_comparative_table(results)

    if output_dir and results:
        self.visualizer.generate_plots(results, output_dir)

```

```

self.visualizer.save_benchmark_excel(results, output_dir)

def upscale(
    self,
    config_paths: list[Path],
    input_img_paths: list[Path],
    output_img_paths: list[Path],
    downscale: bool = False,
) -> None:
    for config_path in config_paths:
        evaluator = self._init_evaluator(config_path)

        for input_img_path, output_img_path in zip(input_img_paths,
output_img_paths):
            img_tensor = self.visualizer.read_image(input_img_path)

            if downscale:
                sr_img_tensor = evaluator.upscale_downscaled(img_tensor)
            else:
                sr_img_tensor = evaluator.upscale(img_tensor)

            output_img_path = output_img_path.parent /
f"{evaluator.model_name.lower()}_{output_img_path.name}.png"

            self.visualizer.save_image(sr_img_tensor, output_img_path)

        del evaluator
        gc.collect()
        torch.cuda.empty_cache()

def compare(
    self,
    config_paths: list[Path],
    hr_img_paths: list[Path],
    output_img_paths: list[Path],
    lr_img_paths: list[Path | None] | None = None,
    crop_size: int | tuple[int, int] | None = None,
) -> None:
    scaling_factors = [ModelConfig.from_yaml(path).scaling_factor for path in
config_paths]

    if len(set(scaling_factors)) > 1:
        raise ValueError(
            f"Cannot compare models with different scaling factors in a single run! "
            f"Found factors: {scaling_factors}"

```

```

    )

    scaling_factor = scaling_factors[0]

    hr_img_tensors = [self.visualizer.read_image(path) for path in hr_img_paths]
    lr_img_tensors = []

    for i, hr_img_tensor in enumerate(hr_img_tensors):
        lr_img_path = lr_img_paths[i] if lr_img_paths else None
        if lr_img_path is None:
            lr_img_tensor = imresize(hr_img_tensor, scaling_factor=1 /
scaling_factor, antialiasing=True)
            lr_img_tensors.append(lr_img_tensor)
        else:
            lr_img_tensor = self.visualizer.read_image(lr_img_path)
            lr_img_tensors.append(lr_img_tensor)

    results = []
    for hr_img_tensor, lr_img_tensor in zip(hr_img_tensors, lr_img_tensors):
        _, hr_img_height, hr_img_width = hr_img_tensor.shape
        results.append(
            {
                "Bicubic": F.interpolate(
                    lr_img_tensor.unsqueeze(0),
                    size=(hr_img_height, hr_img_width),
                    mode="bicubic",
                )
                .squeeze(0)
                .clamp(0, 1)
            }
        )

    for config_path in config_paths:
        evaluator = self._init_evaluator(config_path)
        model_name = config_path.parent.name

        for i, lr_img_tensor in enumerate(lr_img_tensors):
            results[i][model_name] = evaluator.upscale(lr_img_tensor)

        del evaluator
        gc.collect()
        torch.cuda.empty_cache()

    for i, hr_img_tensor in enumerate(hr_img_tensors):
        results[i]["Original (HR)"] = hr_img_tensor

```

```

        if crop_size:
            crop_center_coords =
self.visualizer.get_crop_center(img_tensor=hr_img_tensor, crop_size=crop_size)

        self.visualizer.create_collage_with_crop(
            hr_img_tensor=hr_img_tensor,
            sr_img_tensors_dict=results[i],
            crop_center=crop_center_coords,
            crop_size=crop_size,
            output_img_path=output_img_paths[i],
        )
    else:
        self.visualizer.create_collage(
            sr_img_tensors_dict=results[i],
            output_img_path=output_img_paths[i],
        )

```

ДОДАТОК Б
Слайди презентації

Дипломна кваліфікаційна робота бакалавра
Тема: “Дослідження та програмна реалізація
нейронних мереж для збільшення роздільної
здатності зображень”

Виконав ст. КНТ-113сп
Керівник зав. каф., д.т.н., проф.

Андрій ШОСТАК
Сергій СУББОТІН

2026

Рисунок Б.1 – Слайд 1

Мета, об'єкт, предмет дослідження та завдання на роботу

Об'єкт дослідження: процес інженерії програмного забезпечення для інтелектуального опрацювання цифрових зображень.

Предмет дослідження: методи, алгоритми та програмні засоби архітектур нейронних мереж для збільшення роздільної здатності зображень.

Мета роботи: зменшення витрат часу та підвищення точності проведення порівняльного аналізу архітектур нейронних мереж шляхом розроблення програмного комплексу для автоматизованого тестування та візуалізації результатів збільшення роздільної здатності зображень.

Завдання на роботу:

- проаналізувати існуючі методи та архітектури нейронних мереж для збільшення роздільної здатності зображень;
- спроектувати модульну архітектуру програмного забезпечення для стандартизованого бенчмаркінгу;
- розробити програмні модулі для обробки даних, розрахунку метрик якості та створення графіків;
- здійснити тестування розробленого програмного комплексу та провести порівняльний аналіз обраних архітектур.

2

Рисунок Б.2 – Слайд 2

Візуальне зображення методів збільшення роздільної здатності



Рисунок 3.1 — Розтягнене оригінальне зображення



Рисунок 3.2 — Збільшення за допомогою бікубічної інтерполяції



Рисунок 3.3 — Результат роботи нейронної мережі NAT

3

Рисунок Б.3 – Слайд 3

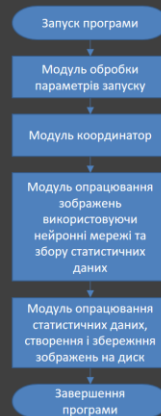
Огляд архітектур нейронних мереж для обробки зображень

Характеристика	CNN	GAN	Transformers	Diffusers
Математична точність (PSNR)	Висока	Середня	Висока	Низька
Перцептивна якість (реалізм)	Низька	Висока	Середня	Висока
Швидкість роботи (Inference)	Висока	Висока	Середня	Низька
Вимоги до пам'яті (VRAM)	Низькі	Середні	Високі	Високі
Ризик появи артефактів	Низький	Високий	Низький	Середній

4

Рисунок Б.4 – Слайд 4

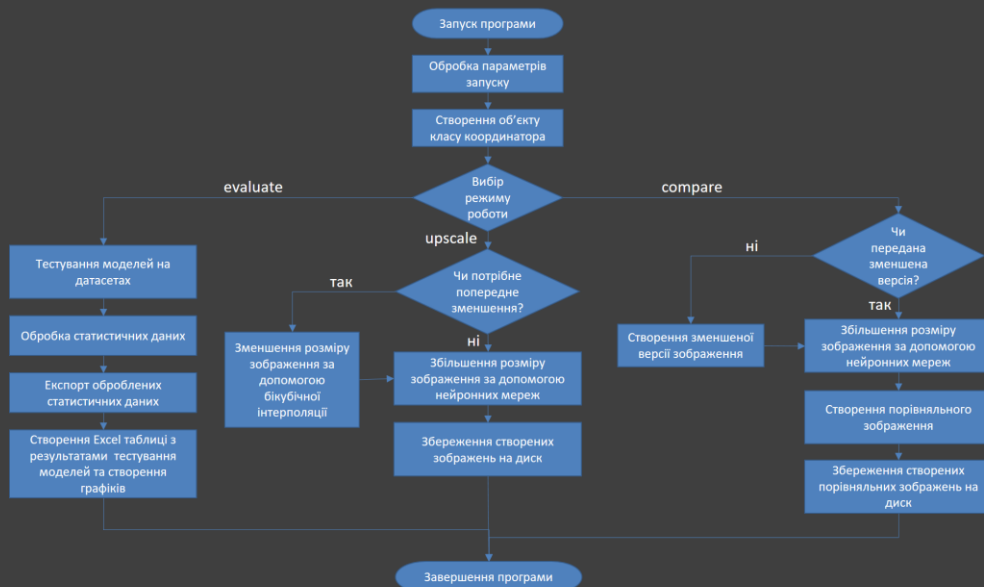
Схема проектування програмного забезпечення



5

Рисунок Б.5 – Слайд 5

Схема функціонування програмного забезпечення



6

Рисунок Б.6 – Слайд 6

Вибір мови програмування

Критерій порівняння	Python	R	MATLAB
Наявність сучасних фреймворків для створення нейронних мереж	Так	Ні	Ні
Рівень підтримки апаратного прискорення GPU (CUDA)	Високий	Низький	Середній
Наявність готових SOTA архітектур	Так	Ні	Ні
Ефективність інструментів обробки зображень	Висока	Низька	Висока
Відкритий вихідний код	Так	Так	Ні

7

Рисунок Б.7 – Слайд 7

Приклади зображень тестових наборів даних



Рисунок 8.1 — butterfly (Set5)



Рисунок 8.2 — 108082 (BSDS100)



Рисунок 8.3 — baboon (Set14)



Рисунок 8.4 — img_008 (Urban100)



Рисунок 8.5 — Donburakokko (Manga109)

8

Рисунок Б.8 – Слайд 8

Порівняльний аналіз об'єктивної точності та візуальної якості моделей

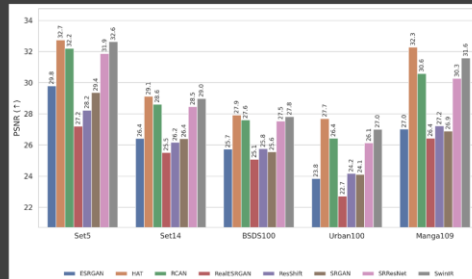


Рисунок 9.1 — Показники PSNR для різних архітектур та датасетів

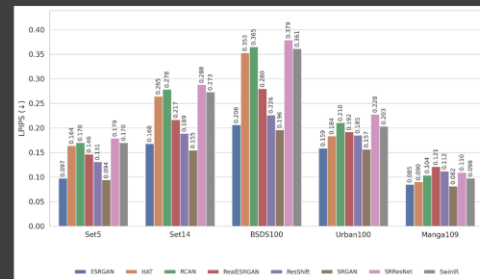


Рисунок 9.2 — Показники LPIPS для різних архітектур та датасетів

9

Рисунок Б.9 – Слайд 9

Порівняльний аналіз об'єктивної точності та візуальної якості моделей

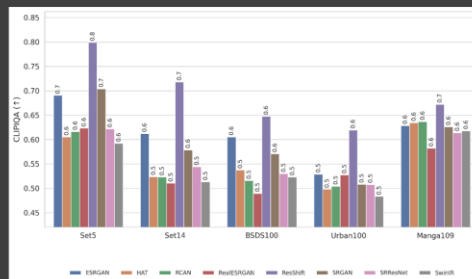


Рисунок 10.1 — Показники CLIPQA для різних архітектур та датасетів

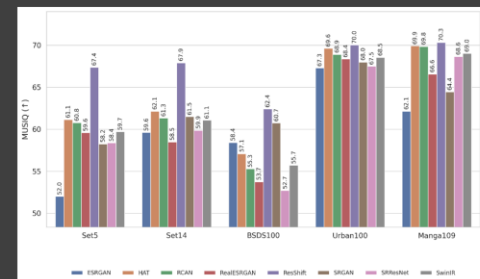


Рисунок 10.2 — Показники MUSIQ для різних архітектур та датасетів

10

Рисунок Б.10 – Слайд 10

Аналіз обчислювальної ефективності та ресурсних витрат алгоритмів

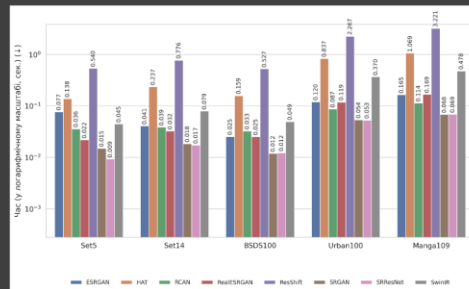


Рисунок 11.1 — Порівняльний аналіз швидкодії нейромережових архітектур

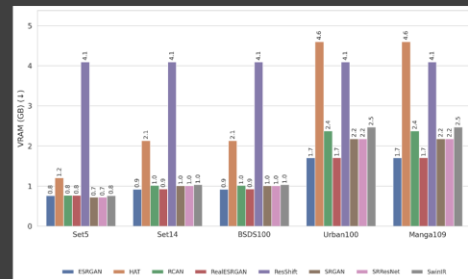


Рисунок 11.2 — Споживання відеопам'яті на тестових наборах даних

11

Рисунок Б.11 – Слайд 11

Аналіз компромісу між якістю відновлення та часом обробки

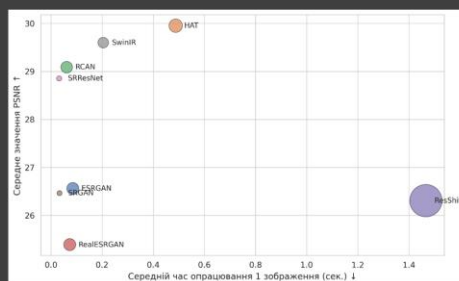


Рисунок 12.1 — Залежність математичної точності від часу інференсу та об'єму параметрів

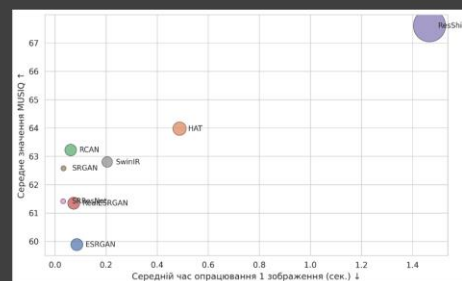


Рисунок 12.2 — Залежність перцептивної якості від часу інференсу та об'єму параметрів

12

Рисунок Б.12 – Слайд 12

Комплексна оцінка балансу характеристик моделей

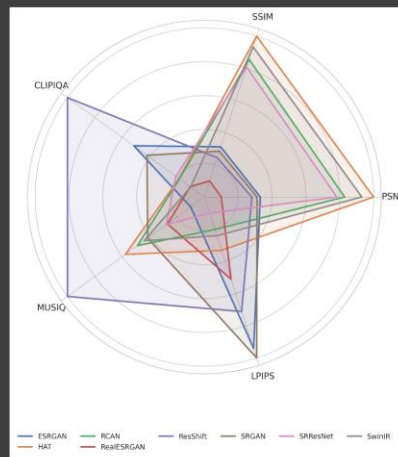


Рисунок 13.1 — Радарна діаграма балансу перцептивної якості та математичної точності моделей

13

Рисунок Б.13 – Слайд 13

Результати роботи моделей



Рисунок 14.1 — Візуальне порівняння повнорозмірної генерації

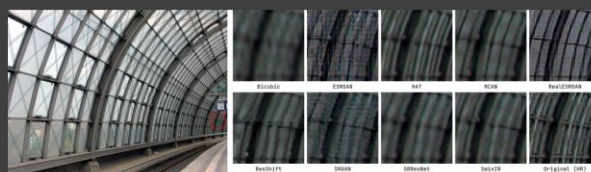


Рисунок 14.2 — Аналіз геометричних артефактів за допомогою локального фрагмента

14

Рисунок Б.14 – Слайд 14

Висновки

- проаналізовано існуючі методи та архітектури нейронних мереж для збільшення роздільної здатності зображень з обґрунтуванням необхідності використання комплексного багатовимірного підходу до їх оцінювання;
- спроектовано модульну архітектуру програмного забезпечення для стандартизованого бенчмаркінгу, яка дозволяє легко інтегрувати нові моделі без зміни основного програмного коду;
- розроблено програмні модулі для обробки даних, розрахунку метрик якості та створення графіків, що додатково оптимізовані алгоритмом тайлінгу для подолання обмежень відеопам'яті;
- здійснено тестування розробленого програмного комплексу та проведено порівняльний аналіз обраних архітектур, який довів наявність компромісу між точністю і фотореалізмом та дозволив сформулювати практичні рекомендації.

15

Рисунок Б.15 – Слайд 15