Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Рекомендовано до захисту:
протокол засідання кафедри
№ _____ від _____._____.2025 р.
Завідувач кафедри ММАД
_____ Струков В.М.
(підпис) (прізвище та ініціали)

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему Розробка моделі селективного кодування для захисту відеоінформаційних ресурсів в інфокомунікаційних системах

Захищено на засіданні ЕК № _____
протокол № _____ від _____._____.2025 р.
Оцінка _____ / _____
Голова ЕК
_____ _____
(підпис) (прізвище та ініціали)

Виконав:
студент 4 курсу, групи КС-42
Спеціальності:
122 Комп'ютерні науки
(шифр і назва спеціальності підготовки)
Малінін Ігор Геннадійович
(прізвище, ім'я та по батькові, підпис)
Керівник д.т.н професор Бараннік Володимир Вікторович
(ступень, звання, прізвище та ініціали, підпис)
Рецензент _____
(ступень, звання, прізвище та ініціали, підпис)

Харків – 2025

БЛАНК ЗАВДАННЯ НА ДИПЛОМНУ РОБОТУ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н. Каразіна

Факультет Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра Математичного моделювання та аналізу даних
Рівень вищої освіти (освітньо-кваліфікаційний рівень) бакалавр
Напрям підготовки Інформаційні технології
Спеціальність 122 – комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ __Струков В. М.__
підпис ініціали, прізвище

"____" _____20___ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____Малініна Ігоря Геннадійовича_____
(прізвище, ім'я по батькові студента)

1. Тема роботи Розробка моделі селективного кодування для захисту_____
відеоінформаційних ресурсів в інфокомунікаційних системах_____
_____

керівник роботи_д.т.н. професор Бараннік Володимир Вікторович_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "___"_____20__ року №_____

2. Строк подання студентом роботи__2 травня 2025 року_____

3. Перелік питань, які потрібно розробити
_Дослідити використання візуальної і відеоінформації. Проаналізувати _методи повного та селективного шифрування, їх переваги та недоліки. Розробити експериментальний метод селекції та кодування відеоінформації_____
_____
_____

4. План роботи

№ з/п	Назви етапів роботи
1	Дослідження теоретичних основ про візуальну та відеоінформацію
2	Аналіз використання візуальної інформації та обсягів її використання
3	Аналіз робіт про повне та селективне шифрування
4	Пошук даних для навчання моделі
5	Навчання моделі для семантичної сегментації
6	Тестування готової нейронної мережі
7	Розробка методу кодування зображень
8	Оцінка якості фінального продукту

5. Дата видачі завдання_____1 лютого 2025_____

Студент _____ ___І.Г. Малінін___
підпис інціали, прізвище

Керівник роботи _____ ____В.В. Бараннік_
підпис інціали, прізвище

АНОТАЦІЯ

Дипломна робота присвячена дослідженню методів селективного шифрування відеоінформації.

Дипломна робота складається зі вступу, 3 розділів, висновків, списку використаних джерел, що складається з 25 джерел і 2 додатки. Роботу викладено на 96 аркушах друкованого тексту, містить 9 формул та 35 рисунків.

Метою є дослідження підходів до селективного шифрування зображень та розробка програмного інструменту, що реалізує ці підходи.

В рамках даної роботи розроблено комплексну систему селективного захисту зображень. Для автоматичної ідентифікації областей інтересу було навчено нейромережеву модель архітектури U-Net на наборі даних Stanford Background Dataset з використанням фреймворку PyTorch, оптимізатора Adam та функції втрат на основі перехресної ентропії, застосовуючи аугментацію даних та стратегію донавчання. Для криптографічного захисту виділених ROI було розроблено та реалізовано новий 9-раундовий симетричний алгоритм шифрування, який використовує функцію деривації ключа PBKDF2, генерацію ключових потоків на основі зв'язаних хаотичних наметових відображень, глобальну перестановку пікселів, чергування операцій заміни та HMAC-SHA256 для автентифікації. Реалізація виконана на Python з використанням бібліотек hashlib, hmac та NumPy.

Ключові слова: Селективне шифрування, захист зображень, семантична сегментація, U-Net, область інтересу, багато-раундове шифрування.

ABSTRACT

The thesis is devoted to the study of methods for selective encryption of video information.

The thesis consists of an introduction, 3 sections, conclusions, a list of sources used, consisting of 25 sources and 2 appendices. The work is presented on 96 pages of printed text, contains 9 formulas and 35 figures.

The goal is to study approaches to selective encryption of images based on neural networks and develop a software tool that implements these approaches

As part of this work, a comprehensive system for selective image protection was developed. For automatic identification of areas of interest, a neural network model of the U-Net architecture was trained on the Stanford Background Dataset dataset using the PyTorch framework, the Adam optimizer and a cross-entropy loss function, using data augmentation and a training strategy. For cryptographic protection of the selected ROIs, a new 9-round symmetric encryption algorithm was developed and implemented, which uses the PBKDF2 key derivation function, key stream generation based on coupled chaotic tent mappings, global pixel permutation, interleaving of replacement operations, and HMAC-SHA256 for authentication. The implementation is done in Python using the hashlib, hmac, and NumPy libraries.

Keywords: Selective encryption, image protection, semantic segmentation, U-Net, region of interest, multi-round encryption.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ГПЧ — Генератор псевдовипадкових чисел

ДКП — Дискретне косинусне перетворення

ДНК — Дезоксирибонуклеїнова кислота

ШІ — Штучний інтелект

2D-LFHCM — Вдосконалене двовимірне логістично-дробове гібридне хаотичне відображення

2D-LM — Двовимірна логістична карта

2D-LMM — Двовимірна логістична модифікована карта

4K — Роздільна здатність екрана або відео, що приблизно відповідає 4000 пікселям по горизонталі

5G — П'яте покоління технологій мобільного зв'язку

8K — Роздільна здатність екрана або відео, що приблизно відповідає 8000 пікселям по горизонталі

AC — Alternating Current

AES — Advanced Encryption Standard

AT-ResNet-CM — Метод шифрування медичних зображень, що поєднує механізм уваги, ResNet та хаотичне відображення

AV1 — AOMedia Video 1

AVC — Advanced Video Coding

AVIF — AV1 Image File Format

CABAC — Context-Adaptive Binary Arithmetic Coding

CBC — Cipher Block Chaining

CMYK — Cyan, Magenta, Yellow, Key

CTR — Counter Mode

D — Discriminator

DC — Direct Current

DES — Data Encryption Standard

ECB — Electronic Codebook

EKM — Extended Key Material

EXIF — Exchangeable Image File Format

G — Generator

GAN — Generative Adversarial Network

GIF — Graphics Interchange Format

GPS — Global Positioning System

H.264 — Стандарт кодування відео, також відомий як MPEG-4 Part 10 або Advanced Video Coding

H.265 — Стандарт кодування відео, також відомий як High Efficiency Video Coding

HD — High Definition

HDR — High Dynamic Range

HEVC — High Efficiency Video Coding

HMAC — Hash-based Message Authentication Code

HMAC-SHA256 —HMAC на основі хеш-функції SHA-256

IoU — Intersection over Union

IoVT — Internet of Video Things

IPMs — Intra Prediction Modes

JPEG — Joint Photographic Experts Group

LCG — Linear Congruential Generator

LFHCM — Логістично-дробове гібридне хаотичне відображення

mIoU — mean Intersection over Union

MKV — Matroska Multimedia Container

MOV — QuickTime File Format

MP4 — MPEG-4 Part 14

MSRC — Microsoft Research Cambridge

MVDs — Motion Vector Differences, Різниці векторів руху

NPCR — Number of Pixels Change Rate

PASCAL VOC — Pattern Analysis, Statistical Modelling and Computational Learning Visual Object Classes

PBKDF2 — Password-Based Key Derivation Function 2

PCG64 — Пермутований конгруентний генератор, 64-бітна версія

PNG — Portable Network Graphics

PSD — Photoshop Document

ReLU — Rectified Linear Unit

ResNet — Residual Network

RGB — Red, Green, Blue

RMSProp — Root Mean Square Propagation, Адаптивний алгоритм оптимізації

ROI — Region of Interest, Регіон інтересу

TIFF — Tagged Image File Format, Теговий формат файлів зображень

UACI — Unified Average Changed Intensity, Уніфікована середня зміна інтенсивності

U-Net — Архітектура згорткової нейронної мережі для семантичної сегментації

VP9 — Відеокодек, розроблений Google

VVC — Versatile Video Coding, Універсальне кодування відео, також H.266

WebP — Web Picture format, Формат зображень для Web, розроблений Google

XMP — Extensible Metadata Platform, Розширювана платформа метаданих

XOR — Exclusive OR, Виключне АБО

DPI — Dots Per Inch, Точок на дюйм

FPS — Frames Per Second. Кадрів на секунду

PPI — Pixels Per Inch, Пікселів на дюйм

Гбіт/с — Гігабіт за секунду

Мбіт/с — Мегабіт за секунду

ВСТУП

У сучасному світі обсяги відеоінформації, що генерується щодня, стрімко зростають. Камери відеоспостереження, відеореєстратори, смартфони, дрони, онлайн-платформи на зразок YouTube і TikTok — усі ці джерела створюють величезні масиви візуальних даних. Відеозапис дедалі активніше використовується в охоронних системах, контролі доступу, транспорті, розумному місті, медицині, освіті, розвагах, криміналістиці. Це зумовлює потребу не лише в ефективному зберіганні та обробці відео, але й у його захисті від несанкціонованого доступу, витоку чи спотворення.

Відео часто містить конфіденційні відомості: обличчя людей, номерні знаки, логотипи, елементи приватної території, які можуть бути використані зі шкідливою метою. Захист відеоінформації стає критично важливим — як з позицій інформаційної безпеки, так і в контексті захисту права на приватність. Повне шифрування відеопотоку залишається ефективним, однак має низку недоліків: воно вимагає значних обчислювальних ресурсів, збільшує затримки під час передачі, ускладнює перегляд і аналіз у реальному часі. До того ж, шифрування всього відео часто є надмірним — зазвичай достатньо захистити лише ті фрагменти, що містять чутливу інформацію.

Тому дедалі більшої популярності набувають методи вибіркового захисту відео. Їхня суть полягає в тому, що шифрування або інший захисний механізм застосовується лише до визначених зон зображення чи кадру, попередньо ідентифікованих як критично важливі.

Одним із головних викликів у вибірковому захисті є автоматичне виявлення зон, що підлягають захисту. Ручне маркування неможливе через обсяги відео та потребу в обробці в реальному часі. Тут на допомогу приходить штучний інтелект, зокрема методи комп’ютерного зору. Сучасні нейромережі демонструють високу точність у розпізнаванні облич, людей, транспортних засобів, тексту та інших об’єктів, що робить їх надзвичайно ефективними для завдань селективного захисту.

Завдяки стрімкому розвитку глибинного навчання, механізмів уваги і трансформерів, стало можливим створення точних детекторів навіть у складних умовах — при завадах, вночі, зі зміною ракурсу або частковим перекриттям. Поєднання таких технологій з методами вибіркового шифрування відкриває нові горизонти в інформаційній безпеці, дозволяючи створювати гнучкі, ефективні й масштабовані системи відеозахисту.

Метою цієї роботи є дослідження сучасних підходів до вибіркового захисту відеоінформації, аналіз їхніх переваг і обмежень, а також оцінка можливостей використання штучного інтелекту для автоматичного визначення зон, які потребують захисту.

1 ПРОБЛЕМАТИКА ЗАХИСТУ ВІЗУАЛЬНОЇ ІНФОРМАЦІЇ

1.1 Візуальна інформація та її характеристики

Візуальна інформація — це форма цифрових даних, що репрезентує візуальні образи у вигляді статичних зображень або послідовностей кадрів та використовується для збереження, передавання, обробки й аналізу за допомогою інформаційно-комунікаційних технологій.

У нашому контексті візуальна інформація розглядається як специфічний тип даних, що має ряд характерних властивостей, пов'язаних із якістю, обсягом, структурою та методами обробки. Ці властивості визначають характеристики зображень та відео. Розуміння технічних характеристик цифрових зображень та відео є ключовим для оцінки викликів, пов'язаних з їх обробкою, передачею та захистом.

Розуміння технічних характеристик статичних візуальних об'єктів є важливою передумовою для нашого дослідження.

Залежно від способу представлення візуальної інформації, зображення поділяються на растрові, векторні та фрактальні. Найбільш поширеним є растровий (або бітмапний) підхід, де зображення представлене у вигляді двовимірної сітки пікселів – окремих точок, кожна з яких має визначений колір та яскравість. Цей метод є оптимальним для передачі фотореалістичних сцен, складних градієнтів та текстур завдяки можливості детального опису кожного елемента зображення. Однак, растрові зображення мають суттєве обмеження: вони чутливі до масштабування.

Альтернативний підхід – векторна графіка. Тут зображення описується не пікселями, а за допомогою математичних об'єктів: точок, ліній, кривих та багатокутників. Ці об'єкти визначаються координатами, параметрами напрямку, товщини ліній, кольором заливки та обведення. Головною перевагою векторного формату є його незалежність від роздільної здатності. Зображення можна масштабувати до будь-яких розмірів без втрати якості, оскільки система просто перераховує математичні формули для нового

розміру. Це робить векторну графіку ідеальним вибором для створення логотипів, іконок, технічних ілюстрацій, шрифтів та будь-яких зображень, що потребують чітких контурів та можливості масштабування. Крім того, окремі елементи векторного зображення легко редагувати. Проте, векторний підхід погано пристосований для передачі фотореалістичних зображень зі складними тональними переходами.

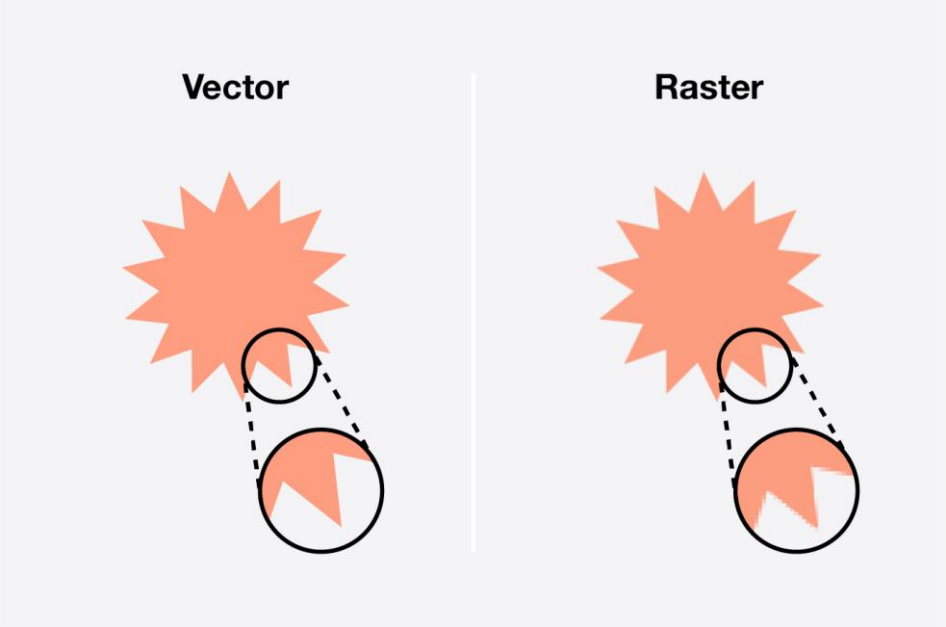

Рисунок 1.1 – порівняння векторної і растрової графіки

Менш поширеним є фрактальний метод генерації зображень. Він базується на рекурсивних математичних алгоритмах, що створюють самоподібні структури. Це означає, що фрагменти зображення при збільшенні подібні до цілого. Фрактальні зображення теоретично мають нескінченний рівень деталізації та дозволяють генерувати надзвичайно складні візуальні структури з відносно простих формул, що забезпечує високий ступінь стиснення інформації. Однак, через складність контролю над кінцевим результатом та специфіку застосування цей метод залишається нішевим.

Мабуть найважливішою характеристикою зображень, за винятком способу представлення, є формат файлу, який і визначає спосіб їх зберігання, ефективність та застосування.

Одним з найбільш важливих аспектів формату є тип стиснення даних. Стиснення з втратами дозволяє суттєво зменшити розмір файлу шляхом вибіркового видалення інформації, яка є найменш помітною для людини. Це робить такі формати, як JPEG, популярними для веб-сторінок, однак надмірне стиснення може призвести до помітної втрати якості. На противагу, стиснення без втрат, що використовується у форматах типу PNG або TIFF, зберігає всю вихідну інформацію, гарантуючи повне відновлення оригінальної якості зображення, але, зазвичай, ціною більшого розміру файлу.

Сам розмір файлу залежить не тільки від методу стиснення, але й від характеристик самого растрового зображення, таких як роздільна здатність та глибина кольору. Роздільна здатність, що вимірюється загальною кількістю пікселів по ширині та висоті, безпосередньо визначає рівень деталізації – більше пікселів дозволяють передати дрібніші деталі. Проте абсолютну кількість пікселів варто відрізняти від піксельної щільності (PPI/DPI), яка вказує на кількість пікселів на одиницю фізичної довжини (зазвичай дюйм) і є критично важливою для якісного відображення на екранах та при друці. Поряд із кількістю пікселів, глибина кольору визначає багатство палітри зображення, вимірюючись кількістю бітів на піксель. Чим вона більша, тим більше унікальних кольорів та відтінків може бути представлено, що покращує реалістичність передачі кольорів.

Для правильної інтерпретації цих кольорів використовується концепція колірного простору, яка визначає конкретний діапазон кольорів, що можуть бути відтворені. Підтримка різних колірних просторів може варіюватися між форматами. Моделі на основі RGB є стандартом для екранних зображень, тоді як модель CMYK використовується переважно для підготовки матеріалів до поліграфічного друку.

Крім основних даних, різні формати можуть підтримувати додаткові можливості. Прозорість дозволяє створювати зображення з невидимими ділянками, що є важливим для веб-графіки, логотипів та іконок. Деякі формати, як-от GIF або WebP, підтримують анімацію, дозволяючи створювати

послідовності кадрів, що використовуються для реклами та неформального спілкування в мережі. Формати, орієнтовані на професійне редагування, наприклад TIFF та PSD, часто підтримують багатошаровість, що дозволяє зберігати та редагувати різні елементи зображення незалежно.

Також файли зображень можуть містити метадані – вбудовану інформацію про саме зображення чи умови його створення. Найбільш поширеним є стандарт EXIF, який зазвичай зберігає технічні параметри зйомки : камера, налаштування, дата, час, GPS-координати. Інші стандарти додають описову інформацію: автор, ключові слова, а XMP слугує гнучким форматом для різноманітних даних.

При виборі формату необхідно враховувати і його сумісність із цільовими пристроями, програмами та платформами, оскільки не всі формати підтримуються однаково універсально. Нарешті, варто згадати, що більш сучасні формати, наприклад AVIF, розширюють можливості, підтримуючи не тільки високі роздільні здатності, але й розширений динамічний діапазон (HDR), що забезпечує значно кращу передачу яскравості та контрасту на сумісних екранах.

1.2 Характеристики відео

Відео, по суті, є послідовністю статичних зображень, або кадрів, швидка зміна яких створює ілюзію безперервного руху.

Основними параметрами, що визначають вихідну якість та обсяг відеопотоку, є роздільна здатність, частота кадрів та бітрейт. Роздільна здатність, вимірювана кількістю пікселів у кожному кадрі (від стандартів високої чіткості HD до надвисокої 8K), визначає рівень деталізації зображення. Вища роздільна здатність забезпечує більшу чіткість, що може бути критично важливим для аналітичних завдань, але призводить до зростання обсягу необроблених даних.

Частота кадрів, він же FPS, тобто кількість кадрів, що відображаються за секунду, впливає на плавність відтворення руху. Більші значення частоти

кадрів забезпечують більш плавне зображення, проте прямо пропорційно збільшують обсяг даних.

Бітрейт, визначає кількість даних, яка використовується для кодування однієї секунди відео після стиснення. Цей параметр безпосередньо пов'язаний з якістю зображення та розміром кінцевого файлу: висока якість при високій роздільній здатності потребує значного бітрейту, наприклад, 4K відео може вимагати 35–68 Мбіт/с, а 8K – від 80 до 240 Мбіт/с і більше. Недостатній бітрейт для обраної роздільної здатності та частоти кадрів призводить до появи видимих артефактів стиснення, блочності або розмиття деталей.

Величезні обсяги необроблених відеоданих роблять їх стиснення необхідним етапом для зберігання та передачі. Цю функцію виконують стандарти стиснення, або кодеки (алгоритми кодування/декодування). Історично домінуючим був H.264 (AVC), який забезпечував хороший баланс між ступенем стиснення та якістю і здобув широку апаратну та програмну підтримку, проте його ефективність є недостатньою для сучасних форматів 4K та 8K. Наступним кроком став H.265 (HEVC), що приблизно на 50% ефективніший за H.264 і краще підходить для відео високої роздільної здатності та з розширеним динамічним діапазоном. Однак він вимагає більшої обчислювальної потужності для кодування/декодування. Як безкоштовна альтернатива був розроблений VP9 від Google, що ефективніший за H.264, широко використовується платформами на кшталт YouTube і має добру апаратну підтримку. Найновішим та найефективнішим на сьогодні є відкритий та безкоштовний кодек AV1, який демонструє приблизно на 30% кращу ефективність стиснення порівняно з H.265. Це робить його оптимальним для стрімінгу відео надвисокої чіткості 4K/8K в умовах обмеженої пропускної здатності мережі. Проте AV1 вимагає найбільших обчислювальних ресурсів, а його апаратна підтримка все ще знаходиться на етапі активого впровадження.

Стиснені відео- та аудіодоріжки, разом із субтитрами та метаданими, зберігаються у файлах-контейнерах, таких як MP4, MKV або MOV. Вибір

контейнера впливає на сумісність та можливості зберігання різних типів потоків.

Перехід до форматів надвисокої роздільної здатності 4K/8K та технології HDR значно ускладнює роботу з відео. Збільшення кількості пікселів та необхідність передачі розширеного діапазону яскравості й кольору призводять до різкого зростання обсягів даних. Це створює значне навантаження на системи зберігання, обробки та передачі даних, стимулюючи розробку та впровадження ефективніших кодеків.

Наразі існує помітний технологічний розрив між можливостями сучасних камер захоплювати відео надвисокої чіткості та ефективністю і доступністю засобів для його подальшої обробки, стиснення та передачі. Стиснення за допомогою найефективніших кодеків є ресурсомістким процесом і вимагає сучасної апаратної підтримки, яка впроваджується поступово. Це частково пояснює, чому менш ефективний, але широко сумісний H.264 все ще залишається поширеним, а H.265 часто виступає як проміжний крок.

1.3 Використання візуальної інформації

Візуальні дані, охоплюючи як статичні зображення, так і динамічні відеопотоки, проникли у безліч сучасних технологічних систем та прикладних областей, ставши важливим фактором їхньої функціональності та розвитку. Одним з найяскравіших прикладів їх інтеграції є концепція "Розумних Міст". Узагальнена архітектура таких міст демонструє, як різноманітні підсистеми – розумна спільнота, системи управління будинком, підключені транспортні магістралі, розумна промисловість, електронна охорона здоров'я – інтегруються за допомогою центральної хмарної платформи та дата-центрів, звідки дані можуть надходити до муніципальних командних центрів. Функціонування Розумних Міст значною мірою спирається на інформацію, що безперервно збирається, зокрема, з розгалужених мереж міських камер. Завдяки застосуванню технологій відеоаналітики, ці дані перетворюються на

інформацію для інтелектуального моніторингу стану ключових об'єктів міської інфраструктури, таких як дороги, мости та комунікації. Відеоаналітика також дозволяє здійснювати управління транспортними потоками, оптимізувати роботу світлофорних систем, виявляти та реагувати на дорожньо-транспортні пригоди. Крім того, вона відіграє незамінну роль у системах забезпечення громадської безпеки, уможливлюючи моніторинг публічних просторів, знаходження потенційно небезпечних ситуацій чи підозрілої активності, а також покращуючи координацію дій правоохоронних органів та екстрених служб.

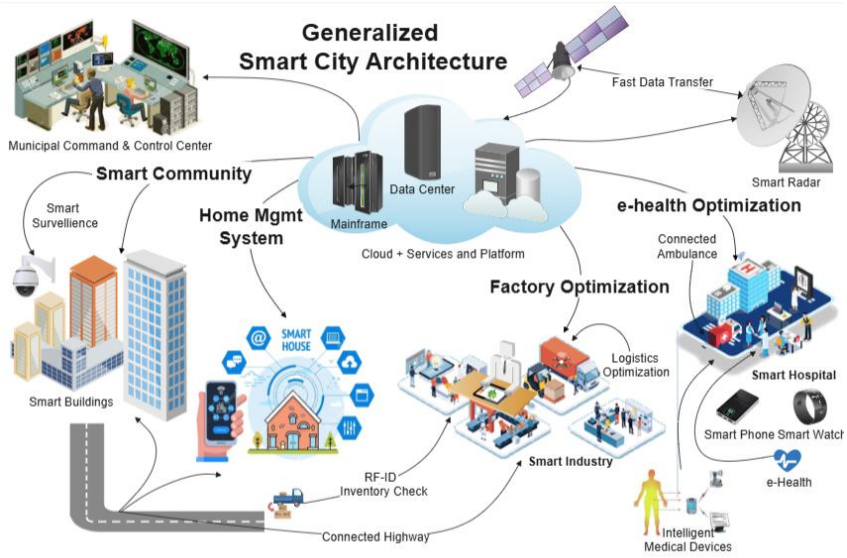

Рисунок 1.2 – загальна концепція архітектури розумного міста

У сфері програмування та розробки програмного забезпечення візуальні дані також демонструють свою багатогранність. Вони активно використовуються для візуалізації складних масивів даних, полегшуючи їх аналіз та інтерпретацію. У процесі розробки користувацьких інтерфейсів візуальні елементи є основою для створення інтерактивних прототипів, а аналіз відеозаписів користувацьких сесій дозволяє виявляти проблеми зручності використання. Відеоаналіз взаємодії користувача з програмою також застосовується під час тестування програмного забезпечення. Особливо

вагомим є внесок візуальних даних у стрімкий розвиток комп'ютерного зору. Тут, за допомогою спеціалізованих бібліотек та сучасних фреймворків штучного інтелекту, здійснюється автоматизований аналіз зображень та відео для вирішення широкого кола завдань – від розпізнавання об'єктів до класифікації зображень, сегментації та аналізу руху.

Застосування візуальних даних простягається і на численні суміжні сфери. У світі зображення та відео є повністю домінуючим форматом контенту, що використовується в маркетингових комунікаціях, освітніх платформах, індустрії розваг та журналістиці. Вони забезпечують функціонування таких повсякденних сервісів, як відеоконференцзв'язок та потокове інтернет-телебачення.

Однак настільки широке та глибоке проникнення візуальних даних у наше життя, особливо в таких критично важливих та чутливих сферах, як медицина, фінанси чи громадська безпека, неминуче породжує серйозні етичні дилеми та створює значні виклики у сфері захисту приватності. Ефективність та точність багатьох інтелектуальних систем прямо залежать від обсягу та репрезентативності даних, на яких вони навчаються та функціонують. Проте ці дані часто містять особисту, конфіденційну інформацію або фіксують аспекти приватного життя людей. Це створює конфлікт між потребою у зборі даних для вдосконалення технологій та забезпечення суспільної користі і правом людини на приватність та захист персональних даних. Вирішення цієї складної проблеми вимагає комплексного підходу, який би поєднував у собі впровадження надійних технічних засобів захисту інформації, розробку чіткого правового регулювання, що відповідало б швидкості технологічного розвитку, а також формування та дотримання етичних норм і стандартів усіма людьми, що працюють із візуальними даними.

1.4 Проблематика

Однією з найгостріших технологічних проблем, що виникають при впровадженні та експлуатації систем які працюють з візуальними даними, є

передача величезних обсягів інформації, що генеруються. Сучасні камери здатні створювати відеопотоки у форматах 4K, 8K та з розширеним динамічним діапазоном, які характеризуються надзвичайно високим бітрейтом. Наприклад, для 4K відео він може сягати 68 Мбіт/с і вище, а для 8K – навіть до 300 Мбіт/с. Таке зростання обсягів даних створює значне навантаження на мережеву інфраструктуру. Аналіз реального інтернет-трафіку, подібний до представленого на діаграмі розподілу трафіку за типами, підтверджує, що саме відео є домінуючим компонентом у напрямку до користувача, часто складаючи 50-65% від усього обсягу переданої інформації, що й створює основне навантаження на канали доставки контенту.

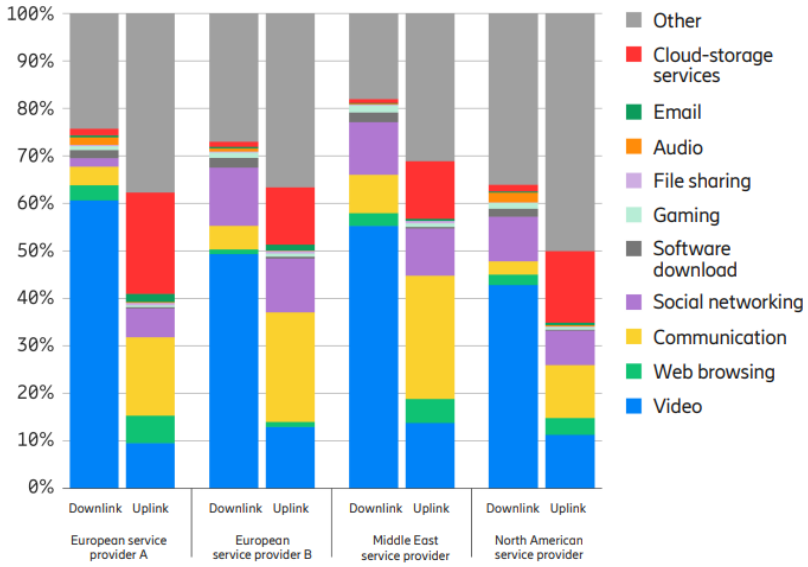

Рисунок 1.3 - частка обсягу трафіку в низхідному та висхідному каналах за категоріями застосунків

В основі цієї проблеми лежить постійно спостережувана тенденція: технології генерації даних розвиваються значно швидшими темпами, ніж відбувається модернізація та розгортання відповідної інфраструктури передачі даних. Загальний обсяг даних, що генеруються людством, зростає експоненційно, що наочно ілюструє діаграма глобального зростання генерації даних, де показники щорічно створених даних вимірюються десятками зетабайт вже станом на 2020 рік. Це відбувається з кількох причин. По-перше,

цикли розробки та впровадження нових сенсорів, камер та алгоритмів обробки зображень часто є коротшими та потребують менших капіталовкладень порівняно з масштабними інфраструктурними проектами. По-друге, ринковий попит на вищу якість зображення та нові візуальні можливості стимулює швидкі інновації у кінцевих пристроях. По-третє, модернізація мереж є складним, довготривалим процесом, що часто залежить від регуляторних дозволів, значних інвестицій та фізичних обмежень. Як наслідок, можливості зі створення даних постійно випереджають здатність існуючих мереж їх ефективно та своєчасно передавати.

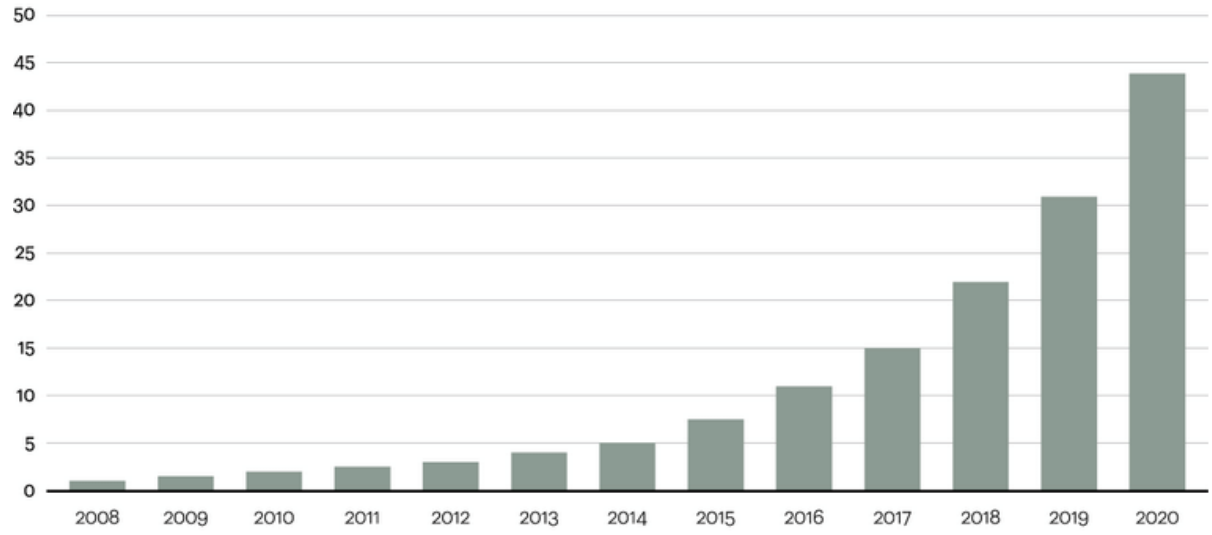

Рисунок 1.4 - зростання обсягу даних за роками в зеттабайтах

Існуючі мережі передачі даних, особливо бездротові сегменти та висхідні канали зв'язку, часто виявляються неспроможними обробити настільки інтенсивні потоки даних у режимі реального часу. Причому характер навантаження відрізняється: якщо канали завантаження перевантажуються переважно через відео, то висхідні канали зазнають навантаження від інших джерел, таких як завантаження у хмарні сховища, комунікаційні сервіси чи соціальні мережі, залежно від регіону та провайдера. Це призводить до виникнення так званого "вузького місця" – обмеження

пропускної здатності, яке або змушує знижувати якість відео, або викликає значні затримки при передачі.

Ця проблема загострюється вимогою низької затримки, яка є важливою для багатьох сучасних застосувань візуальних технологій. Традиційні методи та протоколи передачі даних часто не можуть забезпечити необхідний рівень затримки при роботі з такими великими обсягами даних.

Для подолання цих інфраструктурних обмежень активно розробляються та впроваджуються нові мережеві технології. Стандарт мобільного зв'язку 5G обіцяє значно вищі швидкості передачі даних, теоретично до 10 Гбіт/с і більше, та суттєво нижчу затримку, до 1 мілісекунди. Однак реальна продуктивність мереж 5G сильно залежить від щільності покриття, поточного завантаження мережі та характеристик кінцевого обладнання. Водночас оптоволоконні мережі залишаються основою для високошвидкісного стаціонарного доступу та магістральних каналів зв'язку, забезпечуючи стабільну пропускну здатність понад 10 Гбіт/с та потенціал для подальшого зростання. Проте, навіть ці технології постійно наздоганяють зростаючі потреби, зумовлені вибуховим зростанням генерації даних та домінуванням відеоконтенту.

1.5 Виклик до захисту візуальних даних

Важливою задачею є надійний захист великих обсягів часто чутливих візуальних даних. Ця потреба визначається вимогами до конфіденційності, цілісності, доступності та автентичності такої інформації.

Однак застосування засобів захисту для таких значних обсягів даних обмежується наявними ресурсами. Великі потоки даних створюють суттєве навантаження на мережі передачі та вимагають значних обчислювальних потужностей для обробки. Тому головним завданням є розробка нових методів кодування та захисту візуальної інформації. Ключова вимога до цих методів – поєднання високого рівня безпеки з економним використанням

обчислювальних та мережевих ресурсів. Необхідно створювати більш гнучкі, адаптивні та ресурсоефективні рішення.

Пошук оптимального балансу між необхідним рівнем захисту візуальних даних та пов’язаними з цим витратами ресурсів залишається важливим напрямком досліджень.

2 ОГЛЯД СУЧАСНИХ МЕТОДІВ КОДУВАННЯ ЗОБРАЖЕНЬ ТА ВІДЕО

2.1 Два основні підходи

Існують два основні підходи до шифрування візуальної інформації: повне та селективне шифрування.

Повне шифрування ґрунтується на ідеї застосування криптографічних алгоритмів до всіх даних в зображенні чи відеофайлі. Основна мета цього підходу полягають у досягненні максимально можливою конфіденційності даних. Змінюючи кожен байт, повне шифрування спрямоване на переробку всієї інформації повністю незрозумілою та зробити неможливим її використання без належного ключа, що теоретично забезпечує найвищий можливий рівень захисту контенту.

Селективне шифрування використовуе іншу стратегію: замість обробки всіх даних воно сфокусоване на шифруванні лише певно частини важливо інформації. Основна ідея полягає в досягненні оптимального балансу між безпекою та ефективністю.

Обидва методи мають свої області застосування. Вибір між повноцінним та селективним шифруванням залежить від конкретних потреб у безпеці, наявності ресурсів і характеристик каналу зв'язку та особливостей самої задачі захисту даних.

2.2 Традиційні підходи

Традиційні підходи до шифрування зображень часто спираються на застосування стандартних симетричних блокових шифрів. Серед них DES, Data Encryption Standard, є старим алгоритмом, який працював з 64-бітними блоками та 56-бітним ключем, але зараз вважається застарілим через недостатню стійкість. На зміну йому прийшов сучасний стандарт AES, Advanced Encryption Standard, який оперує 128-бітними блоками даних та

використовує ключі довжиною 128, 192 або 256 біт, забезпечуючи значно вищий рівень безпеки.

Оскільки зображення складаються з великої кількості даних, що перевищує розмір одного блоку шифру, застосування таких алгоритмів потребує використання спеціальних режимів роботи. Найпростішим режимом є ECB, Electronic Codebook, де кожен блок зображення шифрується окремо за допомогою того самого ключа. Однак для зображень цей режим є вкрай невдалим вибором: однакові блоки пікселів (наприклад, ділянки суцільного кольору) перетворюються на однакові зашифровані блоки, що дозволяє легко відновити контури та структуру вихідного зображення. Тому ECB не забезпечує належної конфіденційності для візуальних даних.

Щоб подолати недоліки ECB, використовують складніші режими. Наприклад, режим CBC, Cipher Block Chaining, перед шифруванням поточного блоку даних виконує операцію XOR із попереднім вже зашифрованим блоком. Це створює ланцюгову залежність між блоками, завдяки чому однакові блоки відкритого тексту шифруються по-різному, приховуючи патерни зображення. Іншим поширеним режимом є CTR, Counter Mode, який фактично перетворює блоковий шифр на потоковий. Він генерує унікальний ключовий потік для кожного блоку шляхом шифрування послідовних значень лічильника, а потім цей потік XOR-иться з відповідним блоком даних зображення. Цей режим також добре приховує патерни та додатково дозволяє розпаралелювати процеси шифрування і дешифрування.

Хоча такі режими, як CBC та CTR, значно надійніші за ECB для шифрування зображень за допомогою стандартних алгоритмів на кшталт AES, специфічні властивості зображень стимулювали пошук та розробку нових криптографічних технік, адаптованих саме для візуальних даних. Ці новітні підходи часто прагнуть не лише забезпечити конфіденційність, але й врахувати ці особливості для підвищення ефективності та стійкості шифрування.

Саме тому далі перейдемо до аналізу низки сучасних методів шифрування зображень, які використовують інноваційні підходи для забезпечення надійного захисту цифрових зображень.

2.3 Аналіз новітніх методів повного шифрування зображень

2.3.1 Покращене двовимірне логістичне картування та обчислення ДНК

В своїй статті «A novel image encryption method based on improved two-dimensional logistic mapping and DNA computing» Юанлін Чен та співавтори аналізують різні хаотичні карти для шифрування зображень та пропонують свою, покращену версію на основі старих. Їх метод базується на гібридному підході, об'єднує два компоненти: вдосконалене двовимірне логістично-дробове гібридне хаотичне відображення - 2D-LFHCM, та операції, засновані на принципах ДНК-обчислень.

Центральним елементом, відповідальним за конфузію є 2D-LFHCM. Це хаотичне відображення, що описується рівняннями:

$$\begin{cases} x_{n+1} = \lambda x_n(1 - x_n^2) \times \frac{1}{x_n^2+1} \\ y_{n+1} = \mu x_n(1 - y_n^2) \times \frac{1}{x_n^2+1} \end{cases}, \quad (2.1)$$

де λ та μ - слугують керуючими параметрами, а x_n та y_n позначають змінні стану.

Хаотична карта генерує псевдовипадкові послідовності s_1, s_2, s_3 на основі секретного ключа. Згідно з дослідженням Чена, ця система демонструє покращені хаотичні характеристики порівняно з аналогічними 2D-картами 2D-LM, 2D-LMM, та LFHCM.

Згенерована послідовність s1 використовується для визначення нового розташування пікселів у ДНК-форматі, руйнуючи кореляцію.

За дифузію (зміну значень пікселів) відповідають операції ДНК-обчислень. Для цього методу було встановлено шість алгебраїчних правил, що

поєднують циклічний зсув (лівий/правий) з операціями додавання, віднімання та XOR над ДНК-блоками. Процес включає перекодування: 8-бітні пікселі трансформуються в четвертинне представлення, а потім – у послідовності ДНК (A, C, G, T). До ДНК-представлення застосовуються ці операції, причому конкретна операція для кожного блоку динамічно обирається хаотичною послідовністю s_2, а другий операнд надається ключовою ДНК-послідовністю s_3. Це модифікує значення пікселів, забезпечуючи дифузію та високу ентропію шифротексту.

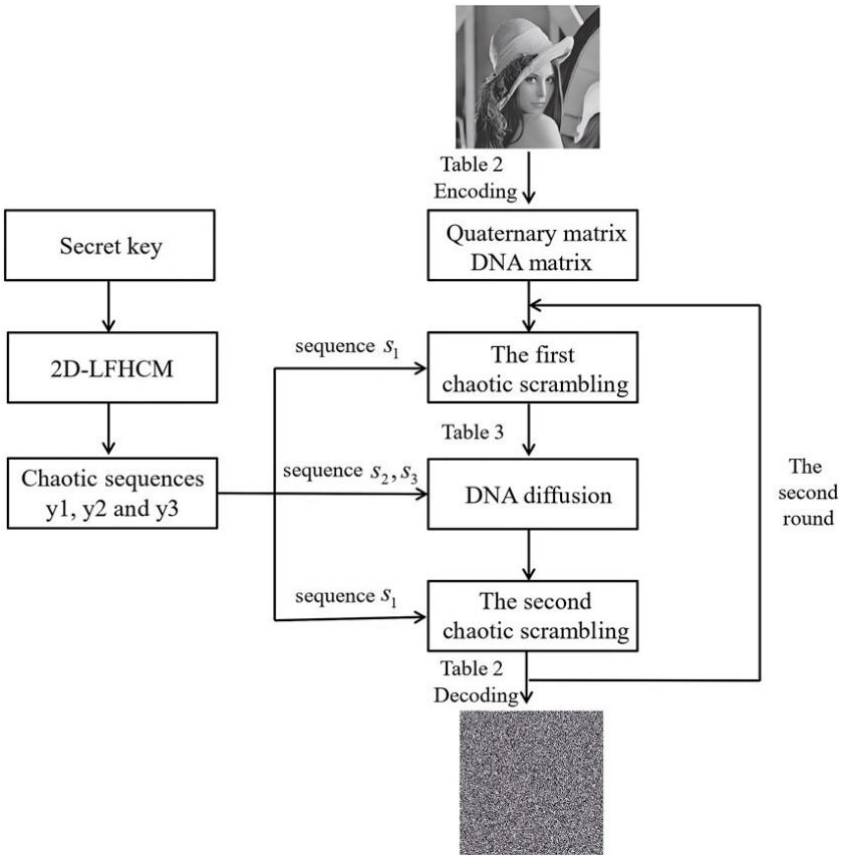

Рисунок 2.1 – схема шифрування за допомогою 2D-LFHCM да ДНК обчислень

Процедура шифрування відбувається поетапно. Спочатку, за допомогою ітерацій 2D-LFHCM із секретним ключем, генеруються ключові потоки s_1, s_2, s_3. Зображення кодується в ДНК-формат. Далі виконується перше хаотичне перемішування позицій нуклеотидів за допомогою s1. Після цього, з використанням s_2 та s_3, застосовуються динамічно обрані алгебраїчні ДНК-

операції для дифузії. Потім виконується друге хаотичне перемішування за допомогою першої послідовності s_1. На заключному етапі модифіковані ДНК-послідовності декодуються назад у 8-бітні значення, формуючи фінальне зашифроване зображення. Якщо вірити дослідженням Юанлін чена, цей комбінований підхід забезпечує стійкість до статистичних атак та високу швидкість шифрування.

2.3.2 Методика CryptoGAN

Більш нестандартним є метод, представлений Ранджітом Бхатом та Рагху Нанджундеговдою у статті «CryptoGAN: a new frontier in generative adversarial network-driven image encryption». Він базується на використанні генеративно-змагальних мереж GAN, архітектура котрих складається з Генератора (G) та Дискримінатора (D).

Процес шифрування зображення за допомогою CryptoGAN можна представити у вигляді кількох етапів:

Початкове традиційне шифрування: На самому початку вихідне зображення піддається шифруванню за допомогою одного з традиційних криптографічних алгоритмів, за приклад в статті наведений AES. Результатом цього етапу є зображення, що належить до так званого "цільового домену" – тобто, воно має характеристики, притаманні надійно зашифрованим даним.

Навчання GAN: На цьому етапі в гру вступає генеративно-змагальна мережа. Генератор навчається перетворювати вихідне незашифроване зображення на зображення, яке було б максимально схожим на представників "цільового домену" (тобто на зображення, зашифровані за допомогою AES). Одночасно дискримінатор навчається відрізняти зображення, згенеровані генератором, від "справжніх" AES-зашифрованих зображень. Це змагальний процес, де генератор прагне "обдурити" дискримінатор, а дискримінатор – якомога точніше виявляти "підробки".

Генерація шифрограми CryptoGAN: Після завершення процесу навчання, навчений генератор готовий до роботи. Він приймає на вхід вихідне

незашифроване зображення і генерує остаточну шифрограму. Важливою особливістю, згаданою в статті, є можливість додаткового посилення безпеки цієї шифрограми шляхом повторної подачі її на вхід того самого або іншого навченого генератора.

Також надається інформація про те, що дешифрування використовує схожі з генератором блоки підвищеної згортки і що існує окрема функція відображення F для цього, але стаття не пропонує окремого, настільки ж детального покрокового опису архітектури саме дешифратора, як це зроблено для генератора. Опис значною мірою спирається на аналогію з процесом шифрування.

Автори стверджують, що запропонована система CryptoGAN здатна забезпечити надійні гарантії безпеки для великої кількості зображень. Це підтверджується наданими в графіках і таблицях тестами ефективності, аналізом ентропії, гістограмним аналізом, а також успішною перевіркою стійкості до диференціальних атак, атак обрізанням та шумом. Проте, якщо аналізувати статті, що аналізують CryptoGAN, можна дізнатись що сама модель дуже велика, через що можна здогадатись про великі нагрузки на обчислювальні ресурси. При великих об’ємах даних це може стати проблемою.

2.3.3 Методика AT-ResNet-CM для обробки медичних зображень

Тепер перейдемо до більш спеціалізованого методу поєднання хаотичної карти та використання штучного інтелекту. У статті «Chaotic medical image encryption method using attention mechanism fusion ResNet model» Сяоу Лі та Хуейлін Пен аналізують недоліки традиційних підходів до захисту медичних зображень та пропонують метод AT-ResNet-CM. Їхній підхід, як і минулі, є гібридним, інтегруючи три ключові технології: глибоку нейронну мережу ResNet, механізм уваги та хаотичне шифрування на основі логістичного відображення.

Модель ResNet виступає як компонент, відповідальний за адаптивне вилучення високовимірних ознак з вихідних візуальних даних. Застосування моделі дозволяє системі не просто обробляти піксельні значення, а здійснювати глибокий аналіз зображення, ідентифікуючи складні структурні та семантичні характеристики, які є важливими для подальшого шифрування. Ключовою перевагою архітектури ResNet є використання залишкових блоків з так званими пропускними з'єднаннями. Ці з'єднання забезпечують альтернативний шлях для поширення градієнтів під час навчання мережі, що вирішує проблему їх згасання, особливо актуальну для дуже глибоких нейронних мереж. Таким чином, ResNet здатна навчатися на значно більшій глибині, що дозволяє їй вилучати більш абстрактні та інформаційно насичені представлення медичних зображень, які потім слугують основою для хаотичного шифрування.

Для підвищення ефективності та безпеки шифрування, а також для оптимізації швидкості, в модель інтегровано окремий механізм уваги. Цей механізм дозволяє системі фокусуватися на найбільш важливих областях зображення, присвоюючи їм вищі ваги. Цей елемент забезпечує більш цілеспрямоване та надійне шифрування критично важливих даних, надаючи їм більший пріоритет. Можна було б сказати що це елемент селективного шифрування, але всеж таки в результаті методу Лі шифрується вся інформація.

За безпосереднє шифрування відповідає логістична хаотична система. Вихідні дані, оброблені ResNet та зважені механізмом уваги, шифруються за допомогою логістичного відображення. Ця хаотична система генерує псевдовипадкову послідовність, яка вносить необхідну випадковість та складність у процес шифрування, як це приблизно вже було в першому розглянутому нами методі. Фінальне зашифроване зображення отримується шляхом застосування операції XOR між обробленими даними зображення та згенерованою хаотичною послідовністю.

Як і в минулий раз, дешифрування проходить по аналогії з шифруванням.

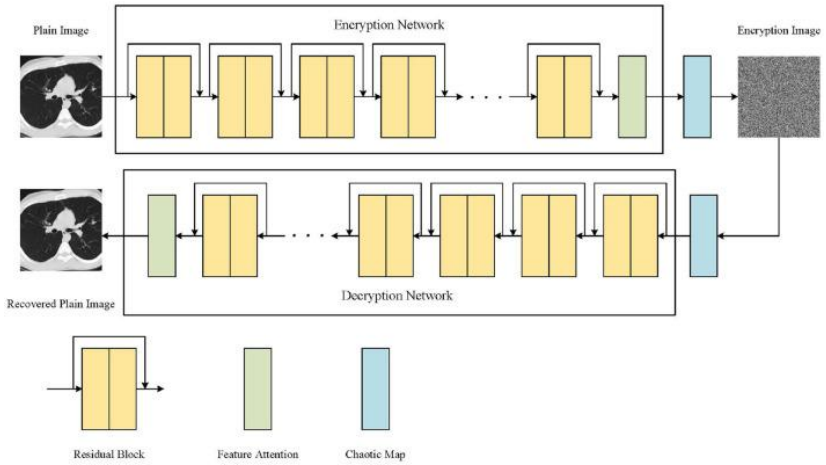

Рисунок 2.2 – загальна схема роботи методу AT-ResNet-CM

Процедура шифрування, таким чином, включає вилучення ознак за допомогою ResNet, фокусування на ROI за допомогою механізму уваги, генерацію хаотичної послідовності логістичним відображенням та застосування операції XOR. Згідно з дослідженням Лі та Пен, метод AT-ResNet-CM демонструє значні переваги: коефіцієнт горизонтальної кореляції досягає 0.0010, а інформаційна ентропія — 7.9965, що близько до ідеальних значень (0 для кореляції та 8 для 8-бітної ентропії). Це свідчить про високий рівень безпеки та руйнування статистичних залежностей в зображенні. Окрім того, автори відзначають суттєве покращення швидкості роботи системи завдяки механізму уваги, тобто елементу селекції в цьому методі.

2.3.4 Підсумки по розглянутим методам

Аналіз представлених методів шифрування зображень виявляє кілька тенденцій, що відрізняють їх від більш традиційних підходів. По-перше, спостерігається явний рух до гібридизації, де сильні сторони різних технологій поєднуються для досягнення кращої безпеки та ефективності. Замість використання одного алгоритму, нові методи інтегрують класичні і не класичні методи, просто декілька не класичних методів і так далі, все для

створення більш ефективного і безпечного варіанту захисту візуальної інформації.

По-друге, бум штучного інтелекту не обійшов стороною і шифрування з дешифруванням, все частіше з’являючись в нових роботах та демонструючи різні можливості. Методики типу CryptoGAN та AT-ResNet-CM показують, як ШІ може бути використаний не просто як інструмент аналізу, а як фундаментальний компонент процесу шифрування. Дивлячись на сучасні тенденції розвитку штучного інтелекту в цілому, можна сміливо казати що роль штучного інтелекту і в цій галузі буде все збільшуватись і збільшуватись.

Проте, не дивлячись на якісні нові приклади шифрування, вони все ще не є ідеальними. Гібридизація і активне використання ШІ ускладнює моделі, потребуючи великих обчислювальних ресурсів при великому наборі данних. Часто компроміс між швидкодією та ефективністю захисту розробники не знаходять, ну або ще не встигли знайти, тому якщо використовувати більш сладні моделі доречно було б поговорити про селективні підходи.

2.4 Селективні методи захисту

Сутність селективного шифрування полягає в ідентифікації та зашифруванні так званих регіонів інтересу (Region of Interest, ROI) — окремих ділянок зображення, кадру, або ж окремих бітів зображення/кадру, що містять важливу або конфіденційну інформацію. Решта зображення, яка не несе особливого смислового навантаження, може залишатися у відкритому вигляді. Залежно від завдання, ROI можуть бути визначені вручну, автоматично або за допомогою алгоритмів машинного зору.

У загальному випадку схема селективного шифрування виглядає наступним чином:

1. Попередній аналіз або сегментація зображення з метою виявлення ROI.
2. Застосування криптографічного алгоритму лише до вибраних даних.

3. З’єднання зашифрованих та відкритих частин в єдиний форматований вихід.

Переваги селективного шифрування полягають насамперед у значному зменшенні обсягу оброблюваних даних. Це забезпечує високу продуктивність, дозволяє використовувати алгоритми на пристроях з низькою обчислювальною потужністю, скорочує затримки при передачі відеопотоку та знижує енергоспоживання. Крім того, гнучкість цього підходу дає змогу адаптувати рівень захисту до конкретних вимог користувача або системи, що важко реалізувати при повному шифруванні.

Водночас, недоліки селективного шифрування полягають насамперед у зниженні загального рівня безпеки через частковий захист даних. Це залишає незашифровані або слабко захищені ділянки вразливими до несанкціонованого доступу та аналізу, створює ризики витоку конфіденційної інформації з цих областей та робить систему чутливою до помилок чи атак на сам механізм ідентифікації регіонів інтересу

Щож, можна перейти до конкретних прикладів реалізації вибіркового шифрування.

2.5 Конкретні приклади алгоритмів селективного захисту

2.5.1 Метод Танга

Метод Танга – це ранній підхід до захисту даних у форматах, що використовують дискретне косинусне перетворення, як-от JPEG. Він поєднує перестановку коефіцієнтів із шифруванням ключових елементів зображення.

Головним методом є зміна способу зчитування квантованих ДКП-коефіцієнтів з блоків 8x8. Стандартне «зигзаг»-сканування, оптимізоване для ентропійного кодування, яке групує важливі низькочастотні коефіцієнти на початку, а менш значущі високочастотні та нульові – в кінці, замінюється на псевдовипадкову перестановку цих коефіцієнтів. Сама послідовність цієї перестановки діє як секретний ключ, без якого коректне відновлення зображення стає неможливим, таким чином шифруючи саму структуру даних.

Для додаткового захисту найважливіших DC-коефіцієнтів, що представляють усереднену яскравість чи колір блоку, застосовується спеціальна обробка. DC-коефіцієнти з групи блоків, в випадку Танга це блоки 8x8, спочатку об'єднуються, а потім ця агрегована інформація шифрується симетричним блочним алгоритмом DES. Після цього кожен 8-бітовий DC-коефіцієнт розділяється: його старші біти залишаються як значення DC, тоді як молодші біти вбудовуються у позицію одного з високочастотних AC-коефіцієнтів того ж блоку. Такий підхід поєднує криптографічне шифрування з елементами стеганографії. На рисунку показана загальна схема методу Танга.

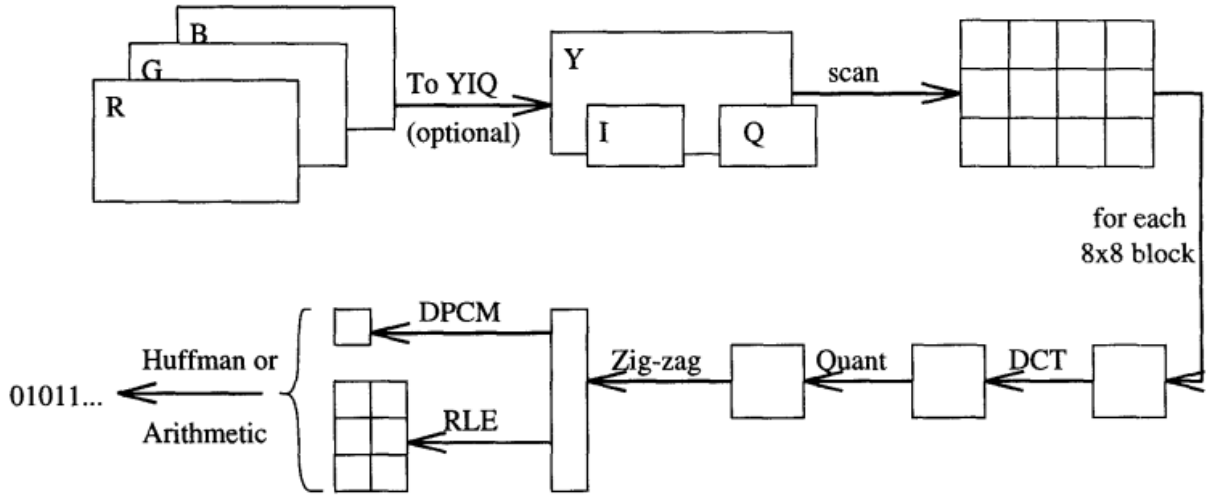

Рисунок 2.3 – схема методу шифрування Танга

Головний недолік цього методу полягає у значному погіршенні ефективності стиснення. Псевдовипадковий порядок зчитування коефіцієнтів руйнує їхні статистичні властивості, на які спираються алгоритми кодування. Це стосується як кодування довгих серій нульових коефіцієнтів, так і оптимального представлення розподілу амплітуд. Як наслідок, обсяг файлу, захищеного таким чином, може істотно зрости, тобто безпека даних досягається ціною зниження коефіцієнта стиснення.

Метод шифрування запропоновані Л. Тангом, являв собою компроміс, орієнтований на досягнення "значного рівня безпеки" для комерційних відеододатків при мінімальних обчислювальних витратах, що дозволяло інтегрувати їх у процес кодування практично без сповільнення. На сьогодні, основні криптографічні елементи цього підходу, зокрема DES з його 56-бітним ключем, є застарілими та вразливими, а значно вищі обчислювальні потужності сучасних систем дозволяють ефективно застосовувати надійніші методи шифрування потоку, роблячи специфічну реалізацію Танга менш актуальною для більшості завдань, що вимагають захисту. Незважаючи на це, робота стимулювала розвиток селективного шифрування та методів захисту, інтегрованих у процеси стиснення, де його ідеї використання особливостей стандартів компресії для криптографії продовжили розвивати інші дослідники.

2.5.2 Вибіркове шифрування відеопотоків, закодованих у VVC

Якщо ж говорити про сучасні підходи, то можна проаналізувати статтю, яка надає метод шифрування для формату VVC, що вперше був представлений в готому виді в 2020-ому році. Амір Фотовват та Хан А. Вахід у своїй статті «Selective Encryption of VVC Encoded Video Streams for the Internet of Video Things» пропонують метод селективного шифрування для стандарту H.266/VVC. Мета – безпека відео в Інтернеті Відеоречей (IoVT) через баланс візуального захисту, низьких обчислювальних витрат, мінімального впливу на бітрейт та сумісності формату.

Основою методу є селективне шифрування інформаційно насичених синтаксичних елементів бітового потоку VVC. Обрано три типи елементів, шифрування яких порушує декодування та візуальне сприйняття.

Шифруються режими внутрішньокадрового передбачення яскравості Luma IPMs. В VVC вони визначають просторове передбачення блоків. Шифрування IPMs компоненти яскравості змушує декодер використовувати

невірні напрямки передбачення, що веде до руйнування контурів, текстур та загальної нерозбірливості.

Також шифруються різниці векторів руху MVDs, що уточнюють вектори руху для між-кадрового передбачення. Їх шифрування призводить до невірної реконструкції руху декодером, спричиняючи хаотичний рух об'єктів та часові розриви.

Шифруються і знаки залишкових коефіцієнтів, помилка передбачення після ДКП та квантування. Їх шифрування змінює полярність помилки для декодера, що веде до значних похибок у значеннях пікселів, спотворюючи текстури та кольори.

Рисунок 2.4 - візуальні результати експериментальних тестів

Перший стовпець на Рисунку 2.4 показує вихідні кадри, другий стовпець – коли зашифровані режими кольоровості, третій стовпець – коли зашифровані режими яскравості, четвертий стовпець – коли зашифровані значення MVD та знаки, а п'ятий стовпець – коли всі попередні елементи зашифровані разом.

Шифрування інтегроване в кодер VVC і відбувається перед ентропійним кодуванням, наприклад, CABAC. Конкретний криптоалгоритм не вказаний, але ключовим є збереження сумісності формату: зашифровані елементи

повинні формувати валідний бітовий потік, який стандартний декодер може парсити, хоч візуальний вміст і буде захищений.

Процедура шифрування вбудована в стандартний процес кодування VVC – на етапах обробки IPMs, MVD та знаків залишків, ці дані шифруються секретним ключем перед їх фінальним записом у бітовий потік.

Автори стверджують, що їхній підхід забезпечує вищу візуальну безпеку, ніж аналоги для HEVC, при збільшенні бітрейту лише на 2-3%. Зберігається повна сумісність формату VVC, що важливо для IoVT та обробки на недовірених вузлах. Згідно із їх тексту, їхній метод вибіркового шифрування розроблений з акцентом на низькі обчислювальні накладні витрати, що є важливим для пристроїв з обмеженими ресурсами. Оскільки шифрується лише невелика, але інформаційно значуща частина відеопотоку, а не весь потік, це логічно призводить до менших затрат часу на обробку порівняно з повним шифруванням. Однак сам стандарт VVC є більш складним, ніж його попередники. Навіть вибіркове шифрування в рамках VVC може бути більш ресурсомістким, ніж аналогічні підходи для HEVC або AVC, через більшу кількість залежностей у потоці даних.

2.6 Висновки по порівнянні підходів і методів шифрування візуальної інформації

Перегляд традиційних методів повного шифрування, таких як AES, та новітніх підходів, що використовують хаотичні карти, ДНК-обчислення та штучний інтелект, підтвердив їхню здатність забезпечувати високий рівень теоретичної безпеки. Водночас зрозуміло, що суттєвим обмежувальним фактором для них, особливо для складних сучасних моделей, залишається досить висока обчислювальна ресурсоємність, особливо при застосуванні для обробки і передачі великих масивів візуальних даних.

Розгляд принципів та прикладів селективного шифрування, показав його потенціал як ефективнішої альтернативи, при умові великої кількості «зайвої», не критично важливої інформації. Зрозуміло, що вибіркове шифрування даних

дозволяє суттєво знизити навантаження на систему та прискорити обробку. Разом з тим, виявились і певні недоліки та проблеми в існуючих реалізаціях селективного шифрування.

Таким чином, можна зробити висновок, що для більш масового використання скоріше саме розвиток селективного шифрування є більш актуальним. Проте виявлені недоліки вказують на актуальність та обґрунтовану необхідність розробки нових, більш досконалих алгоритмів селективного захисту. Такі алгоритми мають бути спрямовані на подолання поточних обмежень, пропонуючи оптимальний баланс між рівнем безпеки, обчислювальною ефективністю та сумісністю з сучасними технологіями обробки візуальної інформації.

3 РОЗРОБКА СИСТЕМИ СЕЛЕКТИВНОГО ШИФРУВАННЯ ЗОБРАЖЕНЬ

3.1 Концепція

Що ж, можна перейти до саме нашої розробки. Загальна концепція, яку ми прагнули реалізувати, — це створення програмного інструменту для селективного захисту відеоінформації, керованого інтелектуальним аналізом вмісту. На відміну від традиційних підходів, що часто передбачають або надлишкове повне шифрування, або його повну відсутність, наша ідея полягає в більш гнучкому та ефективному рішенні.

Суть задумки в тому, щоб система, використовуючи технології машинного навчання, могла самостійно «розуміти» візуальний вміст кадру. Це «розуміння» означає здатність автоматично виявляти наперед визначені категорії об'єктів чи текстурні області. Після такої ідентифікації криптографічні перетворення застосовуються виключно до цих виділених, чутливих фрагментів, залишаючи решту інформації доступною.

Такий селективний підхід, на мою думку, має свої переваги. По-перше, він забезпечує цільову конфіденційність саме для критичної інформації, не обмежуючи доступ до нечутливих частин відео. Це актуально, наприклад, у відеоспостереженні для збереження загальної обстановки при анонімізації облич, або в медичних відео для захисту даних пацієнта без приховування дій лікарів.

По-друге, вибіркове шифрування сприяє оптимізації використання ресурсів інфокомунікаційної системи. Зменшення обсягу даних для складних криптографічних операцій знижує навантаження на процесор, а також вимоги до пропускної здатності каналів передачі та об'ємів сховищ. Це робить підхід перспективним для систем з обмеженими ресурсами або для додатків реального часу.

Хоча реалізація такого інтелектуального захисту має певні виклики, його переваги у сфері захисту відеоінформаційних ресурсів роблять

дослідження та розробку в цьому напрямку досить актуальними. Деталі практичної реалізації програмного забезпечення, що втілює описану концепцію, включаючи його архітектуру, використані алгоритми сегментації та шифрування, а також результати експериментальної перевірки, будуть представлені в наступних підрозділах роботи.

3.2 Розробка нейронної мережі для семантичної сегментації зображень

Наше завдання нейронної мережі в галузі комп'ютерного зору відоме як семантична сегментація. Розвиток методів для її вирішення, як і всієї галузі штучного інтелекту загалом, значно прискорився з початком нейромережевого буму приблизно в 2020-му. Для ефективного навчання будь-якої моделі, особливо глибокої нейронної мережі, першочерговим кроком є підбір та підготовка доцільного набору даних, або датасету, що відповідає поставленому завданню.

3.2.1 Вибір набору даних

Для навчання та оцінки розробленої нейромережевої моделі семантичної сегментації було обрано публічний набір даних Stanford Background Dataset. Цей датасет, хоча і не належить до найпопулярніших, є добре відомим у наукових колах та нерідко використовується для завдань, пов'язаних з аналізом сцен, розпізнаванням фону та сегментацією.

Stanford Background Dataset був представлений дослідницькою групою зі Стенфордського університету, що випливає з його назви. Вперше його описали Стівен Гулд, Річард Фултон та Дафна Коллер у своїй роботі «Decomposing a Scene into Geometric and Semantically Consistent Regions» 2009 року. Основною метою створення цього набору даних було надання дослідникам набору зображень з реального світу, що містять різноманітні фонові елементи та об'єкти переднього плану, разом з детальною піксельною розміткою.

До Stanford Background Dataset належить 715 кольорових зображень, які були зібрані з різноманітних джерел, включаючи інші відомі набори даних, такі як LabelMe, MSRC та PASCAL VOC, а також фотографії, знайдені у вільному доступі в мережі Інтернет. Зображення підбиралися таким чином, щоб представити широкий спектр сцен реального світу, зокрема міські та природні ландшафти, і сцени всередині приміщень. Кожне зображення характеризується наявністю принаймні одного чітко вираженого об'єкта переднього плану. Роздільна здатність більшості зображень становить близько 320x240 пікселів, хоча можливі й незначні відхилення.

Рисунок 3.1 – декілька характерних зображень з датасету

Головною цінністю цього датасету є його детальна анотація, надана для кожного зображення. Ця анотація, перш за все, включає попіксельну семантичну розмітку. Це означає, що кожен піксель на зображенні віднесено до одного з попередньо визначених семантичних класів. В оригінальній роботі Стівена Гулда та його колег було визначено такі основні класи, як небо,

дерево, дорога, трава, вода, будівля та гора. Крім цих фонових категорій, також використовується узагальнений клас для різноманітних об'єктів переднього плану Foreground object.

Окрім семантичних міток, для кожного пікселя також надається інформація про його геометричну орієнтацію відносно гравітаційного напрямку. Ця орієнтація класифікується за трьома категоріями: "догори" (вертикально вгору), "горизонтально" та "донизу" (вертикально вниз). Додатково, для зображень, де це є релевантним, анотовано положення лінії горизонту.

Усі ці дані анотації зберігаються у файлах формату .mat, які містять матриці з мітками класів для кожного пікселя та іншу пов'язану інформацію. При цьому нотація щодо класів також надана у вигляді зображень з масками для сегментування, де кожен клас виділений в окремий колір.

Рисунок 3.2 – приклад маски для одного з зображень

Щодо розподілу на навчальну, валідаційну та тестову вибірки, як це прийнято в сучасних датасетах, то в оригінальній публікації не було запропоновано такого розділу через її відносну старовину. Тому, при розробці скрипту для навчання нейронної мережі, ми самі будемо ділити набір фото на набори.

В цілому, нам в нашій концепції не важливо класифікувати об’єкт або об’єкти детально, а лише зрозуміти де саме він знаходиться, тому цей датасет

нам повністю підходить, додатково надаючи можливість класифікувати текстури.

3.2.2 Вибір архітектури

Після того, як було визначено та підготовлено набір даних для навчання, наступним кроком є вибір архітектури нейронної мережі. 42амец я модель буде відповідати за виконання завдання семантичної сегментації. Враховуючи специфіку завдання, що вимагає як хорошого розуміння контексту зображення, так і точної локалізації меж об'єктів, для даної роботи була обрана нейромережева архітектура U-Net.

Архітектура U-Net є різновидом згорткової нейронної мережі, спеціально створеною для успішного виконання завдань семантичної сегментації зображень. Свою назву «U-Net» архітектура отримала через свою U-подібну схему, яка чітко показує два головні етапи обробки: спадний шлях (енкодер), де зображення поступово стискається для виявлення загальних ознак на різних рівнях, та симетричний висхідний шлях (декодер), що відновлює розмір зображення для точного визначення розташування знайдених об'єктів на рівні кожного пікселя. Інновацією, яка значною мірою забезпечила успіх U-Net, є використання так званих пропускаючих з'єднань. Ці з'єднання напряму передають детальну інформацію про ознаки між відповідними шарами кодера та декодера. Це дає змогу ефективно поєднувати загальні, змістовні ознаки з глибоких шарів, які можуть бути менш точними просторово, з точними деталями зображення, отриманими на ранніх етапах обробки. Таке поєднання інформації забезпечує дуже високу точність у визначенні меж об'єктів, що зробило U-Net одним із стандартів та широко вживаним інструментом у багатьох галузях – спочатку в аналізі медичних зображень, де її вперше успішно застосували, а пізніше і в інших напрямках комп'ютерного зору.

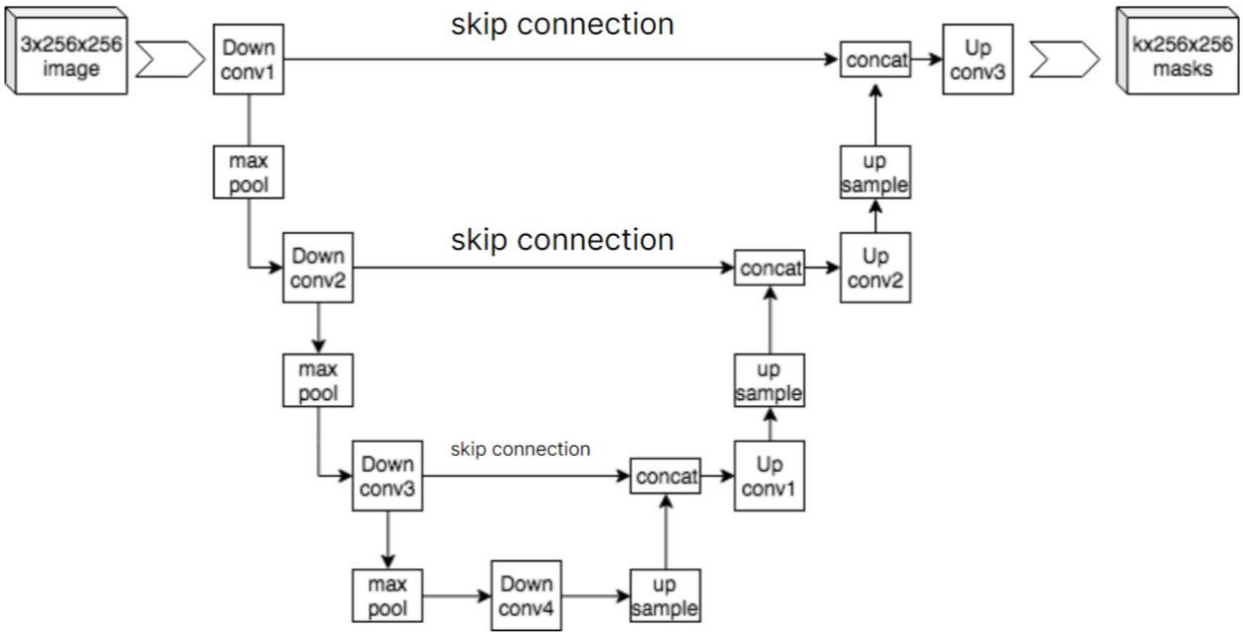

Рисунок 3.3 – загальна архітектура U-Net

3.2.2.1 Енкодер

Кодувальний шлях, або ж енкодер, виконує виявлення важливої контекстної інформації у вхідному зображенні. Для цього він поступово зменшує просторовий розмір оброблюваних даних, так званих карт ознак, одночасно збільшуючи їхню глибину, тобто кількість каналів. За своєю будовою енкодер архітектури U-Net схожий на стандартні згорткові нейронні мережі.

Кожен такий блок містить дві операції згортки, де використовуються фільтри розміром 3x3 пікселі. Важливо розуміти, з якими даними працюють ці фільтри: на самому першому шарі мережі вони аналізують числові значення інтенсивності пікселів вхідного зображення (з урахуванням усіх колірних каналів, наприклад, трьох для RGB, що вимагає від фільтра відповідної глибини). На глибших шарах фільтри працюють вже з картами ознак, отриманими на попередніх етапах, де кожне число відображає силу певної раніше виявленої ознаки; при цьому фільтри також мають глибину, що відповідає кількості каналів такої вхідної карти ознак. У кожній позиції фільтр обчислює зважену суму своїх ваг та відповідних значень з вхідної ділянки, формуючи один елемент вихідної карти ознак.

Після кожної згортки використовується функція активації ReLU. Її завдання – додати в модель нелінійність, що дозволяє мережі навчатися розпізнавати складніші зв'язки в даних, а не лише прості лінійні залежності. Математично функція ReLU для будь-якого вхідного числа x описується так:

$$f(x) = \max(0, x) \tag{3.1}$$

Це означає, що якщо вхідне значення x є додатним, функція повертає саме це значення, а якщо x є від'ємним або нулем, функція повертає нуль. Така, на перший погляд, проста операція дозволяє боротися з проблемою згасання градієнтів та прискорює процес навчання глибоких нейронних мереж.

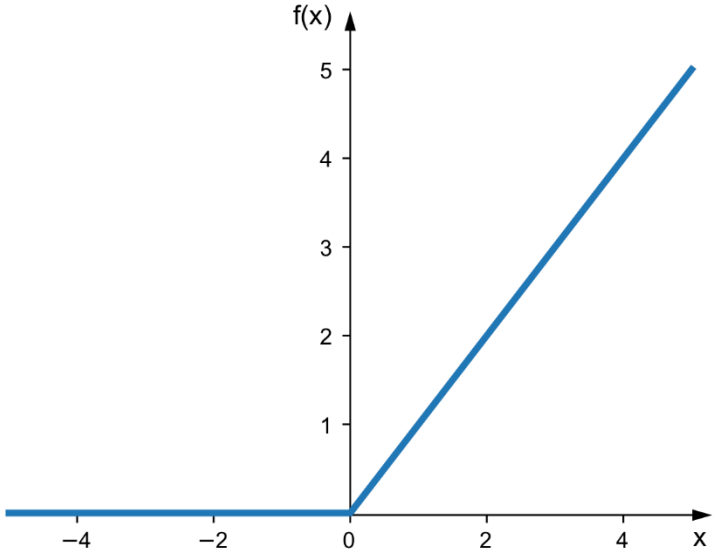

Рисунок 3.4 - графік функції ReLU

Кожен такий типовий блок в енкодері закінчується операцією максимального пулінгу. Для її виконання по карті ознак рухається вікно, найчастіше розміром 2x2 пікселі, зазвичай з кроком 2, що означає обробку непересічних ділянок. З кожної такої ділянки 2x2, що містить чотири числових значення, які представляють силу ознаки, обирається лише одне – те, яке є найбільшим. Саме це обране максимальне значення і стає єдиним пікселем у новій, просторово зменшеній карті ознак. В результаті цієї операції просторові

розміри карти ознак зменшуються. Таким чином, максимальний пулінг допомагає узагальнити інформацію, зберігаючи найсильніші прояви виявлених ознак, та робить модель більш стійкою до невеликих змін у положенні об'єктів на зображенні.

Після кожної операції пулінгу кількість каналів у цих картах зазвичай подвоюється. Наприклад, якщо на одному етапі обробки було 64 канали з інформацією, то на наступному, після пулінгу, їх стане 128, потім 256, і так далі. Завдяки такому збільшенню кількості каналів, мережа може вивчати все більш складні та узагальнені особливості зображення на кожному глибшому рівні енкодера.

Архітектура U-Net, реалізована в нашій роботі, приймає на вхід триканальні кольорові зображення розміром 256x256 пікселів. Розглянемо детальніше компоненти її кодувального шляху (енкодера), співвідносячи їх з програмною реалізацією.

Для побудови енкодера використовуються декілька ключових програмних модулів. Найважливішим є модуль DoubleConv, який повторюється в архітектурі декілька разів. Цей модуль складається з двох послідовних згорткових шарів. Кожен з них – це шар Conv2d, з фільтром 3x3, параметром padding=1 для збереження розміру зображення, за яким іде шар пакетної нормалізації nn.BatchNorm2d та функція активації nn.ReLU.

```
class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
        if not mid_channels:
            mid_channels = out_channels
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )
```

Рисунок 3.5 – клас DoubleConv

Для кожного кроку зменшення розміру зображення в енкодері використовується модуль DownBlock. Він спочатку застосовує операцію максимального пулінгу, щоб зменшити висоту та ширину карти ознак удвічі. Після цього отримана карта ознак обробляється модулем DoubleConv, який ми розглянули вище.

```
class DownBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv(in_channels, out_channels)
        )

    def forward(self, x):
        return self.maxpool_conv(x)
```

Рисунок 3.6 - клас DownBlock

Використовуючи визначені вище модулі DoubleConv та DownBlock, кодувальний шлях моделі UNet формується у її конструкторі. Створюється початковий блок self.inc типу DoubleConv та чотири послідовні блоки зниження роздільної здатності: self.down1, self.down2, self.down3 та self.down4, кожен типу DownBlock. На кожному етапі кількість каналів ознак типово подвоюється, а просторова роздільна здатність зменшується.

```
self.inc = DoubleConv(n_channels, 64)
self.down1 = DownBlock(64, 128)
self.down2 = DownBlock(128, 256)
self.down3 = DownBlock(256, 512)
factor = 2 if bilinear else 1
self.down4 = DownBlock(512, 1024 // factor)
```

Рисунок 3.7 - Ініціалізація шарів енкодера в класі UNet

Під час прямого проходження даних через, вхідне зображення x послідовно обробляється створеними блоками енкодера. Карти ознак, отримані після початкового блоку та кожного з наступних трьох блоків зниження роздільної здатності зберігаються. Ці збережені карти є важливими, оскільки вони передаються до декодувального шляху через пропускаючі з'єднання. Вихід останнього блоку енкодера є результатом роботи всього кодувального шляху і слугує входом для декодера.

3.2.2.2 Декодер та пропускні з’єднання

Після того, як енкодер обробив зображення та вилучив з нього контекстні ознаки, зменшивши їхню просторову роздільну здатність, починає працювати декодер. Основна мета декодера – поступово відновити просторову роздільну здатність карт ознак до розміру вихідного зображення, використовуючи при цьому інформацію, отриману енкодером, для точної локалізації об'єктів та формування фінальної карти сегментації.

Декодер працює дзеркально до енкодера: він послідовно збільшує просторовий розмір карт ознак. Це досягається за допомогою операцій підвищення дискретизації, таких як транспонована згортка або білінійна інтерполяція. На кожному етапі цього шляху кількість каналів у картах ознак зменшується.

Ключову роль у ефективності декодера U-Net відіграють пропускаючі з'єднання. Вони передають карти ознак безпосередньо з відповідних за рівнем шарів енкодера на шари декодера. Карти ознак, отримані в декодері після операції підвищення дискретизації, об'єднуються з картами ознак, що надійшли з енкодера. Таке об'єднання дозволяє декодеру поєднувати глибокі семантичні ознаки, які несуть інформацію про те, що зображено, і отримані з глибших шарів енкодера, з точною просторовою інформацією високої роздільної здатності, яка несе інформацію про те, "де" саме розташовані деталі, і надходить з ранніх шарів енкодера. Після об'єднання ознак застосовуються згорткові шари для їх подальшої обробки.

На завершення декодувального шляху спеціальний вихідний згортковий шар, часто з фільтром 1x1 формує фінальну карту сегментації, де кількість каналів відповідає кількості класів, на які потрібно сегментувати зображення.

Аналогічно до енкодера, декодер вашої U-Net також використовує допоміжні модулі. Основними для декодера є UpBlock та OutConv.

Модуль UpBlock відповідає за один етап підвищення просторової роздільної здатності в декодувальному шляху. Спочатку він збільшує розмір карти ознак, що надходить з попереднього глибшого шару декодера. У вашій реалізації для цього використовується білінійна інтерполяція. Потім збільшена карта ознак об'єднується з відповідною картою ознак з кодувального шляху, яка передається через пропускаюче з'єднання. Нарешті, об'єднана карта ознак обробляється модулем DoubleConv.

```
class UpBlock(nn.Module):
    def __init__(self, in_channels, out_channels, bilinear=True):
        super().__init__()
        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
            self.conv = DoubleConv(in_channels, out_channels, in_channels // 2)
        else:
            self.up = nn.ConvTranspose2d(in_channels , in_channels // 2, kernel_size=2, stride=2)
            self.conv = DoubleConv(in_channels, out_channels)

    def forward(self, x1, x2):
```

Рисунок 3.8 - клас UpBlock

Клас OutConv формує фінальний вихід моделі. Він складається з одного згорткового шару nn.Conv2d з ядром розміром 1x1. Цей шар проектує багатоканальну карту ознак, отриману з останнього блоку декодера, на карту ознак, кількість каналів якої дорівнює кількості цільових класів.

```python
class OutConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(OutConv, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=1)

    def forward(self, x):
        return self.conv(x)
```

Рисунок 3.9 - клас OutConv

Декодувальний шлях у конструкторі класу UNet складається з чотирьох послідовних блоків та фінального вихідного шару.

На кожному етапі декодера модуль UpBlock збільшує просторову роздільну здатність та, після об'єднання з даними з енкодера, модуль DoubleConv всередині UpBlock обробляє ознаки, зменшуючи кількість каналів. Нарешті, self.outc перетворює 64 канали на кількість каналів, що дорівнює кількості класів.

У методі forward дані з "шийки пляшки" та збережені карти ознак з енкодера послідовно обробляються блоками декодера.

Кожен блок UpBlock приймає карту ознак з попереднього шару декодера та відповідну карту ознак з енкодера. Результат роботи останнього UpBlock подається на OutConv для отримання фінальних логітів.

3.2.3 Функція втрат та оптимізатор

Для навчання розробленої моделі U-Net, тобто для кількісної оцінки розбіжності між передбаченими моделлю масками сегментації та еталонними масками з набору даних, а також для подальшої корекції ваг моделі, були обрані відповідні функція втрат та алгоритм оптимізації.

Як функція втрат була використана перехресна ентропія, Cross-Entropy Loss, реалізована в PyTorch як nn.CrossEntropyLoss. Ця функція втрат є звичайним вибором для задач багатокласової семантичної сегментації. Вона вимірює, наскільки розподіл ймовірностей, передбачений моделлю для кожного пікселя по всіх класах, відрізняється від істинного розподілу.

nn.CrossEntropyLoss в PyTorch очікує на вхід необроблені логіти від моделі та цільові маски з індексами класів.

Математично, для одного пікселя, якщо x – це вектор логітів, виданий моделлю, а class – це індекс істинного класу, то функція втрат перехресної ентропії виражена як:

$$\text{Loss}(\text{x}, \text{class}) = -\log\left(\frac{\exp(\text{x}_{\text{class}})}{\sum_{\text{j}} \exp(\text{x}_{\text{j}})}\right), \quad (3.2)$$

де x_{class} - це логіт для правильного класу;

$\sum_j \exp(x_j)$ - це сума експонент всіх логітів;

$\frac{\exp(x_{class})}{\sum_j \exp(x_j)}$ - це ймовірність, присвоєна моделлю правильному класу.

По суті, ця функція штрафує модель, якщо вона присвоює низьку ймовірність правильному класу.

Для оновлення ваг нейронної мережі під час тренування був обраний оптимізатор Adam. Adam є адаптивним алгоритмом оптимізації, який здобув широку популярність завдяки своїй здатності поєднувати переваги двох інших відомих методів: AdaGrad та RMSProp.

Оптимізатор Adam обчислює індивідуальні адаптивні швидкості навчання для кожного параметра моделі. Це відбувається на основі оцінок першого та другого порядкових моментів градієнтів. По суті, Adam відстежує два ключові показники:

1. Експоненційно спадаюче середнє значення попередніх градієнтів: Це схоже на концепцію "імпульсу" в інших оптимізаторах. Накопичення інформації про попередні градієнти допомагає прискорити рух у напрямку мінімуму функції втрат та згладити коливання, що особливо корисно на складних "ландшафтах" функції втрат і може призводити до швидшої збіжності.

2. Експоненційно спадаюче середнє значення попередніх квадратів градієнтів : Цей аспект схожий на принципи, закладені в алгоритмах AdaGrad

та RMSProp. Він дозволяє адаптувати швидкість навчання для кожного параметра окремо.

Такий підхід дозволяє Adam динамічно регулювати швидкість навчання для різних параметрів: ті, що отримують великі або часті оновлення, матимуть меншу ефективну швидкість навчання, тоді як параметри з малими або рідкісними оновленнями отримають більшу. Це робить його особливо корисним для задач зі складними ландшафтами функції втрат або розрідженими градієнтами. Алгоритм також включає механізм корекції зміщення для оцінок моментів, що покращує стабільність на початкових етапах навчання.

3.2.4 Набори даних та тренування

Для об'єктивної оцінки роботи моделі U-Net та уникнення її перенавчання, весь набір даних Stanford Background Dataset в 715 пар "зображення-маска" було розділено на три групи: навчальну, валідаційну та тестову.

Навчальна вибірка призначена для безпосереднього тренування моделі та налаштування її параметрів. Валідаційна вибірка слугує для контролю процесу навчання та вибору найкращої версії моделі, не беручи участі в оновленні її ваг. Тестова вибірка використовується для фінальної, незалежної оцінки продуктивності навченої моделі на нових даних.

Розподіл здійснювався у пропорціях: 70% даних для навчальної вибірки, 20% – для валідаційної, та 10% – для тестової.

```
train_ratio = 0.70
val_ratio = 0.20
test_ratio = 0.10

assert abs(train_ratio + val_ratio + test_ratio - 1.0) < 1e-9

train_val_pairs, test_pairs = train_test_split(
    all_pairs,
    test_size=test_ratio,
    random_state=42,
    shuffle=True
)

train_pairs, val_pairs = train_test_split(
    train_val_pairs,
    test_size=(val_ratio / (train_ratio + val_ratio)),
    random_state=42,
    shuffle=True
)
```

Рисунок 3.10 – код розподілу даних датасету на виборки

Навчання розробленої моделі U-Net здійснювалося з використанням бібліотеки PyTorch та було розділене на чотири послідовні стадії, що включали 5, 20, 35 та ще 5 епох відповідно, із загальною тривалістю 65 епох. Такий поетапний підхід дозволив гнучко керувати процесом навчання. Для всіх стадій було використано розмір міні-батчу 8 зображень. Швидкість навчання для оптимізатора Adam була встановлена на рівні 0.001 для перших трьох стадій (сумарно 60 епох), а на четвертій, заключній стадії в 5 епох, швидкість навчання була зменшена в 10 разів до 0.0001 для більш точного доналаштування ваг моделі.

Процедура навчання на кожній епосі включала два основні етапи: тренування на навчальній вибірці та валідацію на валідаційній вибірці.

Для кожного міні-батчу з навчального завантажувача даних виконувався повний цикл: пряме поширення сигналу через мережу для отримання передбачень, розрахунок значення функції втрат для оцінки помилки, обчислення градієнтів та їх зворотне поширення через мережу, а також крок оптимізатора Adam для відповідного оновлення ваг моделі.

Після завершення тренування на всіх батчах навчальної вибірки модель переводилася в режим оцінки для проведення валідації. На цьому етапі для кожного міні-батчу з валідаційного завантажувача даних також виконувалося пряме поширення сигналу та розрахунок функції втрат, однак без обчислення градієнтів та оновлення ваг. Головною метрикою для оцінки якості сегментації на валідаційній вибірці та для вибору найкращої моделі слугувало середнє значення IoU, також відомого як індекс Жаккара. Ця метрика для кожного класу обчислюється як відношення площі перетину між передбаченою моделлю маскою сегментації та істинною еталонною маскою до загальної площі їх об'єднання.

Для наочної демонстрації динаміки процесу навчання та ефективності розробленої моделі на різних етапах, були побудовані графіки залежності значень функції втрат та метрики IoU від кількості пройдених епох. Ці графіки дозволяють візуально оцінити, як модель навчалася, чи не виникало перенавчання, та як змінювалася її узагальнююча здатність.

На Рисунку 3.11 представлено історію зміни значень функції втрат для навчальної та валідаційної вибірок протягом усіх 65 епох навчання, що включали чотири стадії з різними параметрами.

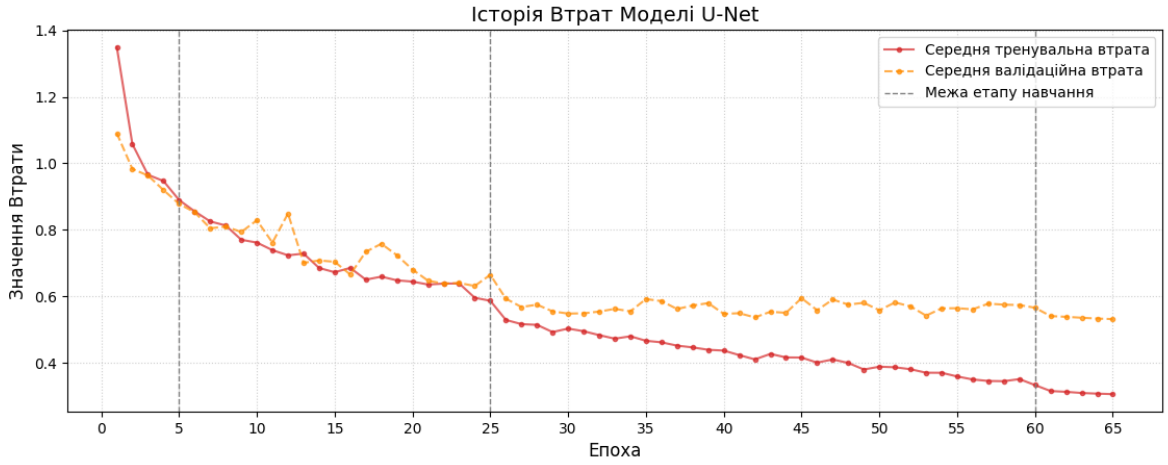

Рисунок 3.11- історія значень функції втрат на навчальній та валідаційній вибірках під час тренування моделі

Цей графік ілюструє, як покращувалася якість сегментації моделі з кожною епохою та на різних стадіях тренувального процесу.

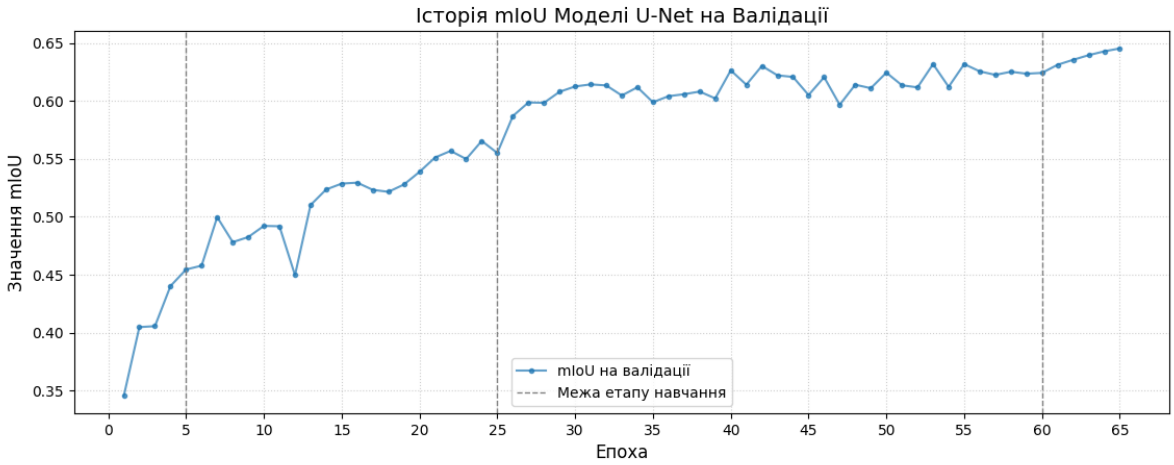

Рисунок 3.12 – історія значень mIoU на валідаційній вибірці під час тренування моделі

Представлені графіки свідчать про загалом успішне тренування моделі протягом 65 епох. Спостерігалося послідовне зниження значень функції втрат на навчальній та валідаційній вибірках, причому валідаційна крива не демонструвала значного зростання, що вказує на відсутність сильного перенавчання. Показник середнього IoU на валідаційній вибірці показував стабільне зростання, досягнувши найкращого значення близько 0.6453 на заключному етапі навчання. Поетапна стратегія тренування, зі зміною швидкості навчання на останній стадії, сприяла доналаштуванню моделі. Загалом, модель продемонструвала хорошу збіжність.

3.2.5 Тестування моделі

Після завершення всіх етапів навчання та вибору найкращої версії моделі U-Net на основі показників на валідаційній вибірці, була проведена її фінальна оцінка на тестовій вибірці. Тестова вибірка складалася з даних, які

модель не бачила під час навчання або валідації, що дозволяє отримати об'єктивну оцінку її здатності до узагальнення.

Для тестування використовувалася функція evaluate_model, яка за своєю логікою схожа на функцію валідації: модель переводилася в режим оцінки, і для кожного міні-батчу з тестового завантажувача даних обчислювалися вихід моделі, значення функції втрат та метрики якості сегментації. Обчислення градієнтів та оновлення ваг на цьому етапі не проводилося.

Результати оцінки найкращої версії моделі на тестовому наборі даних наступні:

Середня тестова втрата: 0.6562

Середній IoU на тестовому наборі: 0.5824

Значення IoU по окремих класах на тестовій вибірці представлені нижче:

sky: 0.7770

tree: 0.6546

road: 0.8110

grass: 0.7549

water: 0.4674

building: 0.6198

mountain: 0.0408

foreground: 0.5336

Тривалість оцінки на всьому тестовому наборі склала 7.36 секунд.

Для наочної демонстрації ефективності розробленої моделі U-Net та для якісної оцінки її роботи, розглянемо декілька прикладів сегментації зображень з тестової вибірки. Візуальний аналіз дозволяє краще зрозуміти сильні та слабкі сторони моделі, а також побачити, як вона справляється з різними об'єктами та сценами. На кожному з представлених нижче прикладів (Рисунки 3.13, 3.14, 3.15) показано: (а) оригінальне вхідне зображення, (б) еталонну маску сегментації та (в) маску, передбачену розробленою моделлю.

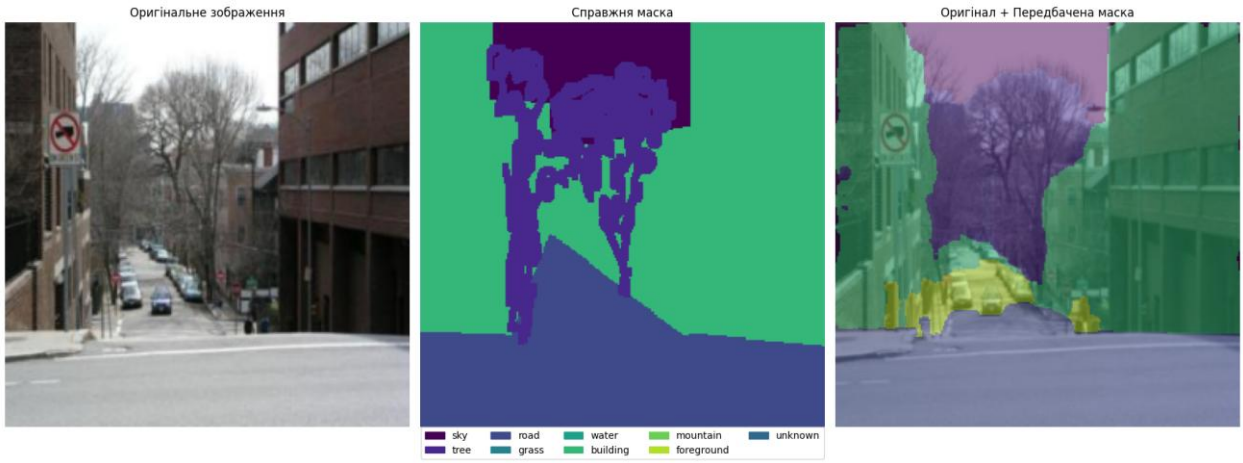

Рисунок 3.13 - приклад сегментації №1: (а) вхідне зображення; (б) еталонна маска; (в) маска, передбачена моделлю

На першому прикладі, що зображує міську вулицю з автомобілями, будівлями та деревами, модель загалом коректно розмітила основні об'єкти, такі як будівлі, дорогу та небо. Однак, спостерігаються певні неточності в сегментації дерев: частина неба була помилково класифікована як дерева, що, ймовірно, пов'язано зі складною структурою переплетених гілок на фоні неба.

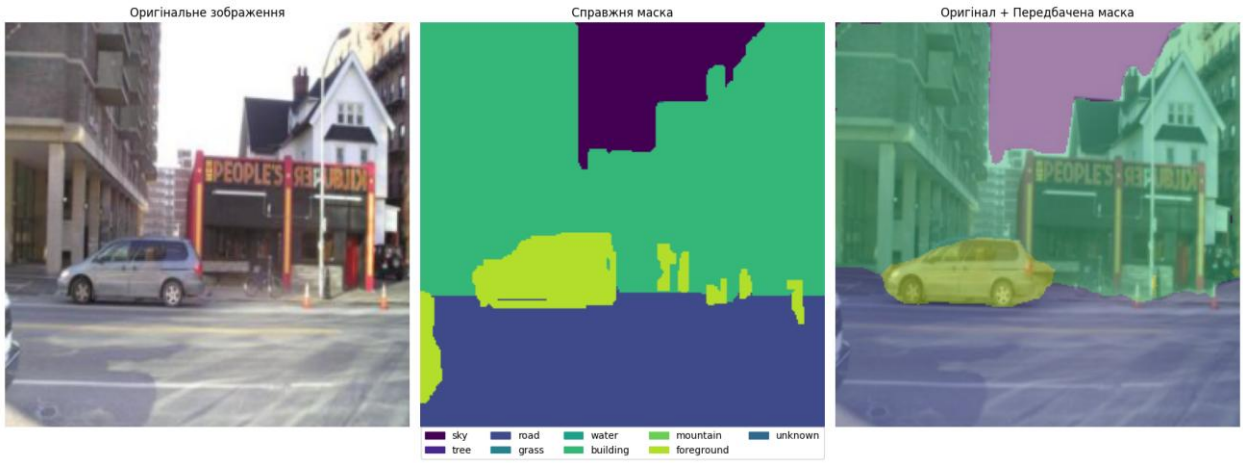

Рисунок 3.14 - приклад сегментації №2: (а) вхідне зображення; (б) еталонна маска; (в) маска, передбачена моделлю

Другий приклад демонструє вулицю з припаркованим автомобілем та яскравою вивіскою. На еталонній масці автомобіль позначено як об'єкт переднього плану, а будівлі, небо та дорога мають відповідні розмітки. У

цьому випадку модель продемонструвала дуже хорошу якість сегментації автомобіля. Будівлі, небо та дорога також розмічені майже без помилок. Незначне змішування класів небо та будівля спостерігається лише у верхньому правому куті зображення, ну і ігнорування деяких малих об’єктів переднього плану, що слабо виражені на самій фотографії, змішуючись з фоном.

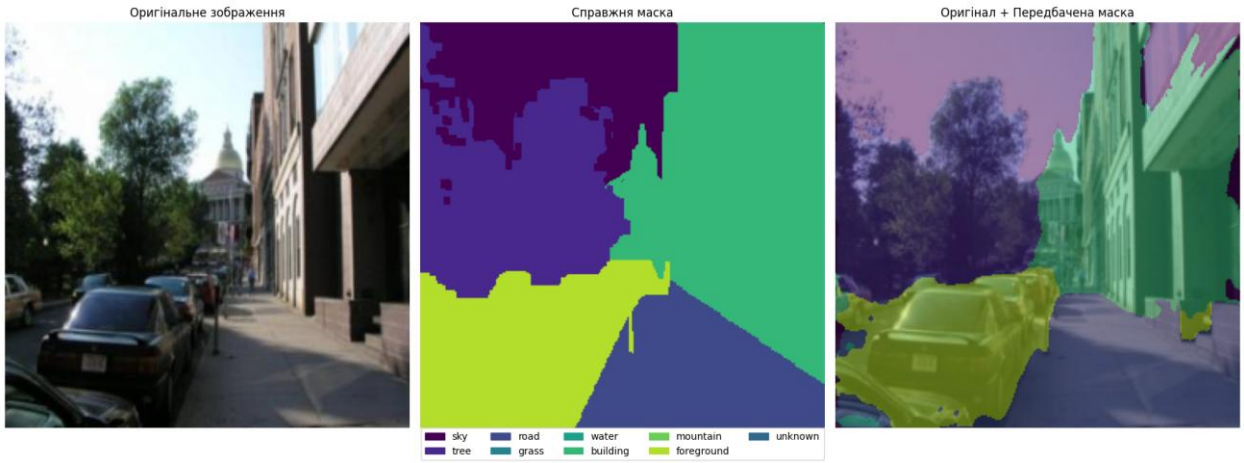

Рисунок 3.15 – приклад сегментації №3: (а) вхідне зображення; (б) еталонна маска; (в) маска, передбачена моделлю

На третьому прикладі показано вулицю з автомобілями, деревами та характерним куполом будівлі на фоні. Модель показала дуже хороші результати в сегментації дерев, будівель та автомобілів. Спостерігаються невеликі артефакти по краях об'єктів праворуч, що, можливо, спричинено тінями або особливостями освітлення. Також є незначний перехід сегментації дороги на частину тротуару.

Проаналізувавши наведені приклади, можна зробити висновок, що розроблена архітектура добре справляється із сегментацією основних та великорозмірних класів. Візуальне порівняння передбачених масок з еталонними підтверджує загалом коректне визначення меж об'єктів.

Водночас, спостерігаються деякі характерні недоліки. Іноді модель плутає небо з деревами, особливо в областях з густим переплетінням гілок. Також можуть виникати невеликі помилки по краях об'єктів, особливо в

місцях з тінями або недостатнім освітленням. В деяких випадках передбачені межі об'єктів виглядають дещо м'якшими або згладженими порівняно з чіткими межами на еталонних масках.

3.3 Розробка методу кодування

3.3.1 Загальна схема

Після успішної ідентифікації цільових областей на зображенні за допомогою розробленої нейромережевої моделі семантичної сегментації, наступним етапом є їх криптографічний захист. Для забезпечення надійного шифрування виділених пікселів було розроблено спеціалізований симетричний алгоритм, що базується на багато-раундовій структурі перетворень. Такий підхід обрано з метою підвищення криптографічної стійкості шляхом ітеративного застосування різних операцій, що ускладнює потенційний криптоаналіз.

Загальна схема багато-раундового шифрування для виділеної області зображення передбачає послідовне виконання фіксованої кількості раундів (у поточній реалізації – дев'ять раундів). Кожен раунд послідовно застосовує дві основні криптографічні операції до поточного масиву пікселів, які підлягають шифруванню.

Спочатку виконується глобальна перестановка пікселів, під час якої просторове розташування всіх обраних пікселів псевдовипадково змінюється. Для кожного раунду ця перестановка керується унікальним початковим значенням для генератора перестановок, що забезпечує різний характер перемішування. Основна мета цієї операції – руйнування статистичних зв'язків між сусідніми пікселями та забезпечення ефекту дифузії.

Після перестановки здійснюється заміна значень пікселів, де модифікуються значення колірних компонент (R, G, B) кожного, вже переставленого, пікселя. Ця модифікація відбувається за допомогою ключового потоку, згенерованого системою зв'язаних хаотичних наметових відображень, параметри якої також є унікальними для кожного раунду.

Важливою особливістю є те, що тип операції заміни чергується між раундами: у раундах з парним індексом застосовується операція побітового XOR кожного байта пікселя з відповідним байтом ключового потоку, тоді як у раундах з непарним індексом використовується операція модульного додавання.

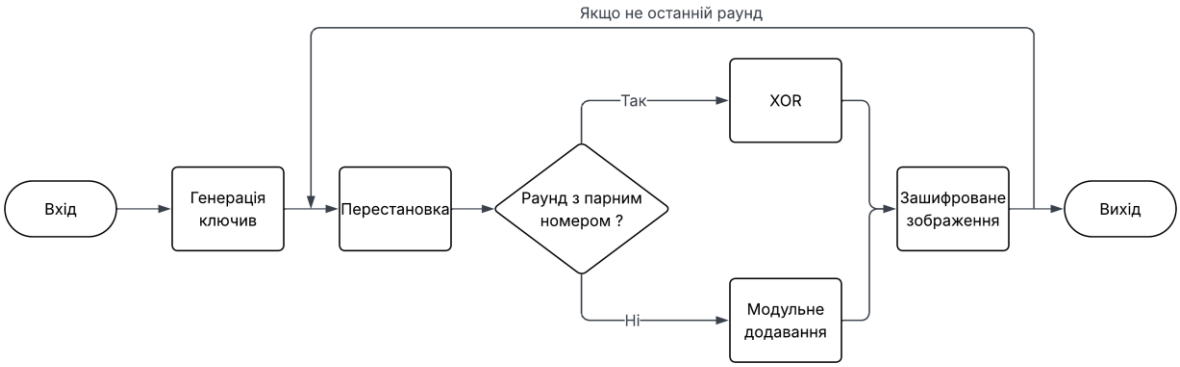

Рисунок 3.16 – загальна схема шифрування

3.3.2 Генерація ключових матеріалів та параметри

Фундаментом розробленого методу шифрування є перетворення секретного пароля на весь набір криптографічних ключів та параметрів, необхідних для подальших операцій. Цей процес інкапсульовано у функції _derive_keys_and_params_pbkdf2, яка послідовно виконує декілька ключових кроків для генерації детермінованого, але стійкого ключового матеріалу.

На вхід ця функція приймає пароль користувача та криптографічну сіль. Сіль являє собою випадкову послідовність байтів фіксованої довжини у 16 байтів, яка генерується при кожному новому сеансі шифрування за допомогою генератора. Використання унікальної солі для кожної операції шифрування (навіть з однаковим паролем) гарантує генерацію унікального розширеного ключового матеріалу, що унеможливлює атаки з використанням попередньо обчислених таблиць, що підвищує загальну безпеку системи. Згенерована сіль зберігається разом із зашифрованими даними, оскільки вона необхідна для відтворення того самого ключового матеріалу під час розшифрування.

Основним інструментом для перетворення пари пароль-сіль на бінарний ключовий матеріал слугує стандартна функція деривації ключа PBKDF2. Вона реалізована в модулі hashlib Python.

У програмній реалізації, всередині функції _derive_keys_and_params_pbkdf2, перед викликом hashlib.pbkdf2_hmac обчислюється загальна необхідна довжина ключового матеріалу. Вона складається з суми довжин, необхідних для параметризації всіх раундів шифрування та для ключа HMAC. Сам виклик функції виглядає наступним чином:

```
total_chaos_ekm_len = chaos_ekm_len_per_round * num_rounds
total_derived_key_len = total_chaos_ekm_len + hmac_key_len

derived_material = hashlib.pbkdf2_hmac(
    'sha512',
    password.encode('utf-8'),
    salt,
    iterations,
    dklen=total_derived_key_len
)
```

Рисунок 3.17 – код перетворення пароля і криптографічної солі на ключовий масив байтів

Отриманий EKM розподіляється на 32-байтовий ключ для HMAC-SHA256 та дев'ять 22-байтних блоків, кожен з яких призначений для параметризації одного раунду шифрування. З кожного такого 22-байтного блоку послідовно вилучаються групи байтів, які шляхом перетворень формують необхідні початкові значення x_0, y_0, керуючі параметри μ_1, μ_2, параметри зв'язку ϵ_1, ϵ_2, кількість ітерацій для системи зв'язаних хаотичних наметових відображень, а також цілочисельне значення perm_seed для ініціалізації генератора перестановок поточного раунду. Для забезпечення

стабільності хаотичного режиму, згенеровані параметри x0, y0, μ1, μ2 додатково обмежуються для перебування у валідних діапазонах.

У результаті, функція _derive_keys_and_params_pbkdf2 повертає список словників з параметрами для кожного з дев'яти раундів та ключ HMAC, забезпечуючи детерміноване розгортання початкового пароля в усі криптографічні компоненти.

3.2.3 Підготовка зображення

Після генерації криптографічних параметрів, наступним кроком є підготовка зображення до вибіркового шифрування. Спочатку вхідне зображення стандартизується, приводиться до 8-бітного формату. Потім, використовуючи маску семантичної сегментації, отриману від нейромережі, та ідентифікатор цільового класу, визначається фінальна бінарна маска, яка вказує на пікселі, що підлягають шифруванню. Ця маска формується з урахуванням обраного режиму: або через пряме виділення пікселів цільових об'єктів, або через визначення та розширення обмежуючих прямокутників навколо кожного виявленого об'єкта.

Для демонстрації ми будемо використовувати саме другий спосіб та і вважаємо його більш ефективним, тому що так ми не будемо надавати потенційному порушнику силует, щоб він не зміг зрозуміти не тільки що знаходиться на зображенні в деталях, а й припустити тип об’єкта.

Нарешті, пікселі, що відповідають цій фінальній масці, вилучаються з зображення та формують окремий масив даних, який і буде безпосередньо оброблятися багато-раундовим алгоритмом шифрування.

3.2.4 Процесс шифрування

3.2.4.1 Перестановка

Першою криптографічною операцією в кожному з дев'яти раундів шифрування є глобальна перестановка пікселів. Ця операція застосовується до поточного масиву даних. Метою цієї перестановки є руйнування просторових

кореляцій між пікселями виділеної області та забезпечення властивості дифузії, за якої зміни в одному невеликому сегменті вхідних даних призводять до змін у багатьох частинах вихідних даних раунду. Це має суттєво ускладнювати статистичний аналіз шифротексту.

Перестановка реалізована у функції _permute_pixels_global, яка приймає на вхід масив пікселів форми (K, 3), де K – кількість пікселів, а 3 – RGB компоненти, цілочисельне початкове значення для генератора псевдовипадкових чисел та булевий прапор decrypt, що вказує напрямок операції. Для кожного раунду шифрування використовується унікальне зерно, отримане з відповідного набору параметрів раунду.

R1	G1	B1
R2	G2	B2
R3	G3	B3
…	…	…

Рисунок 3.18 – приклад початку масиву

Процес перестановки відбувається наступним чином. Спочатку, на основі наданого зерна, ініціалізується генератор псевдовипадкових чисел. У нас це алгоритм PCG64. Це сучасний ГПЧ, відомий своїми хорошими статистичними властивостями та високою швидкістю. Принцип його роботи полягає у двох основних етапах: оновленні великого внутрішнього стану та застосуванні складної функції виводу до цього стану для отримання псевдовипадкового числа.

Внутрішній 128-бітний стан генератора PCG64 оновлюється за принципом, схожим на лінійний конгруентний генератор :

$$S_{n+1} = (s_n * M + I)(mod\ 2^{128}) \quad (3.3)$$

де S_n – поточний стан;

M – константа-множник;

I – константа-інкремент.

Початковий стан S_0 визначається наданим зерном.

Однак, ключовою особливістю PCG64, що відрізняє його від простих LCG, є те, що псевдовипадковий результат генерується не безпосередньо з цього внутрішнього стану Sn. Замість цього, до стану Sn застосовується спеціальна функція виводу. Ця функція складається з ретельно підібраної послідовності побітових операцій, таких як різноманітні XOR-зсуви та бітові ротації. Основне призначення цих перетворень – суттєво покращити статистичні властивості генерованих 64-бітних чисел, забезпечити їхню кращу непередбачуваність та усунути багато відомих недоліків, притаманних простішим LCG.

Важливо, що весь цей процес, починаючи з ініціалізації початкового стану S0 за допомогою наданого зерна, є повністю детермінованим. Це гарантує, що для однакового зерна буде згенеровано однакову послідовність псевдовипадкових чисел, а отже, і однакову карту перемішування, що потрібно для дешифрування.

Після ініціалізації генератора псевдовипадкових чисел, він використовується для створення карти перемішування. Ця карта являє собою послідовність, що містить унікальні індекси всіх пікселів, які обробляються, але розташовані у псевдовипадковому порядку. По суті, ця послідовність визначає нове цільове місце для кожного пікселя з поточного, ще не перемішаного, масиву. Формування нового, переставленого масиву пікселів відбувається шляхом впорядкування вихідних пікселів згідно з цією

згенерованою картою перемішування. Таким чином, кожен піксель переміщується на нову позицію відповідно до цієї карти, забезпечуючи повне та, завдяки початковому значенню генератора, детерміноване перетасування всього набору оброблюваних пікселів.

Рисунок 3.19 – зображення з маскою від нейромережі та зображення з початковим кодуванням об’єкта

3.2.4.2 Генерація хаотичного ключового потоку

Після того, як просторовий порядок пікселів було змінено за допомогою глобальної перестановки, наступним кроком у кожному раунді є генерація безпосереднього ключового матеріалу, який буде використаний для модифікації значень колірних компонент.

Процес створення цього потоку покладається на набір параметрів хаотичної системи, які були згенеровані на етапі виведення ключів з пароля та солі і є специфічними для поточного раунду. Ці параметри включають початкові значення x_0, y_0, керуючі параметри μ_1, μ_2, параметри зв'язку ϵ_1, ϵ_2, та кількість початкових ітерацій.

Для генерації ключового потоку на основі цих параметрів використовується функція _generate_chaotic_keystream_py_list. Вона приймає

згаданий словник round_params та необхідну довжину ключового потоку в байтах, яка розраховується відповідно до обсягу даних (кількості пікселів, помноженої на три колірні компоненти), що обробляються в цьому раунді.

```
num_pixels_in_data = processed_pixels_data.shape[0]
num_bytes_needed_for_round = num_pixels_in_data * 3
keystream_round = _generate_chaotic_keystream_py_list(round_params, num_bytes_needed_for_round)
if len(keystream_round) != num_bytes_needed_for_round:
```

Рисунок 3.20 – код генерації ключового потоку

В основі генератора лежить система двох зв'язаних одновимірних хаотичних наметових відображень. Використання зв'язаних систем спрямоване на отримання більш складної та менш передбачуваної динаміки. Спочатку, використовуючи початкові значення x_0, y_0 та інші параметри з round_params, система виконує skip_iterations початкових ітерацій для досягнення свого атрактора, де її поведінка стає стабільно хаотичною.

Ітераційний процес для станів системи після "прогріву" описується наступними рівняннями, де x_t та y_t є проміжними значеннями, отриманими із застосуванням функції наметового відображення $T(v,\mu)=\mu\cdot\min(v,1-v)$ до поточних станів x_n,y_n:

$$\begin{cases} x_t = \mu_1 * \min(x_n, 1 - x_n) \\ y_t = \mu_2 * \min(y_n, 1 - y_n) \end{cases}, \tag{3.4}$$

Потім ці проміжні значення використовуються для обчислення нових станів x_{n+1} та y_{n+1} з урахуванням параметрів зв'язку ϵ_1, ϵ_2:

$$\begin{cases} x_{n+1} = (x_t + \epsilon_1 * y_t)(mod\ 1) \\ y_{n+1} = (y_t + \epsilon_2 * x'_{n+1})(mod\ 1) \end{cases}, \tag{3.5}$$

де x_n і y_n – поточні значення станів на ітерації n першого та другого зв'язаних хаотичних відображень відповідно:

μ_1 і μ_2 – керуючі параметри для першого та другого наметового відображення відповідно:

x_t і y_t – проміжні значення, отримані після застосування функції наметового відображення до поточних станів;

ϵ_1, ϵ_2 – безрозмірні параметри зв'язку, що визначають ступінь впливу стану одного відображення на інше;

x_{n+1} і y_{n+1} – значення станів двох відображень на наступній ітерації;

$x_{n+1'}$ - оновлене значення x_{n+1}, яке було обчислене на цьому ж поточному кроці ітерації.

На кожній ітерації генерації, отримані стани x_{n+1} та y_{n+1} перетворюються на цілочисельні байти ключового потоку в діапазоні [0,255] шляхом їх масштабування та взяття цілої частини. Згенеровані байти по черзі додаються до списку keystream_round, доки не буде сформовано ключовий потік необхідної довжини. Кінцевою метою цього процесу є отримання ключового потоку з хорошими статистичними властивостями для подальшого використання в операціях заміни.

3.2.4.3 Виняткова диз'юнкція

Після того, як просторовий порядок пікселів у масиві було змінено глобальною перестановкою, та для поточного раунду було згенеровано унікальний хаотичний ключовий потік, виконується заключна операція раунду – заміна значень пікселів. Мета цієї операції – безпосередньо модифікувати значення колірних компонент R, G, B кожного пікселя, тим самим приховуючи вихідну візуальну інформацію та забезпечуючи властивість конфузії - заплутування зв'язку між шифротекстом та ключем/відкритим текстом.

Операція заміни застосовується побайтово до кожної з трьох колірних компонент кожного пікселя в масиві processed_pixels_data, використовуючи відповідний байт з ключового потоку. Тип операції заміни чергується залежно від індексу поточного раунду.

У раундах шифрування з парним індексом, а саме в раундах 0, 2, 4, 6 та 8, для модифікації значень пікселів використовується операція побітового виключного АБО (XOR). Ця операція є фундаментальною в криптографії завдяки своїй простоті, швидкості та властивості самооберненості.

Для кожної колірної компоненти кожного вже переставленого пікселя Pbyte з масиву processed_pixels_data та відповідного байта K_{byte} з ключового потоку keystream_round, обчислюється зашифроване значення C_{byte} за формулою:

$$C_{byte} = P_{byte} \oplus K_{byte}, \quad (3.6)$$

Операція виконується побітово: якщо відповідні біти P_{byte} та K_{byte} однакові, результуючий біт C_{byte} дорівнює 0; якщо біти різні – результуючий біт дорівнює 1.

Ключовою перевагою XOR є те, що для розшифрування застосовується точно така ж операція з тим самим ключовим потоком. Це забезпечує просте та якісне відновлення вихідних значень пікселів на відповідних раундах розшифрування.

Рисунок 3.21 – використання XOR на об’єкті зображення

3.2.4.4 Модульне додавання

У раундах шифрування з непарним індексом, для модифікації значень пікселів застосовується інший тип операції заміни – модульне додавання. Ця операція, хоча й проста, вносить в процес шифрування алгебраїчні властивості, відмінні від побітового XOR, що сприяє ускладненню загальної криптографічної стійкості при чергуванні з XOR у різних раундах.

Аналогічно до операції XOR, для кожної колірної компоненти кожного вже переставленого пікселя Pbyte з масиву processed_pixels_data та відповідного байта Kbyte з ключового потоку keystream_round, обчислюється зашифроване значення Cbyte. Однак, замість побітової операції, виконується арифметичне додавання за модулем 256:

$$C_{byte} = (P_{byte} + K_{byte}) (\mathrm{mod}\ 256) \tag{3.7}$$

Взяття за модулем гарантує, що результат завжди залишатиметься в допустимому діапазоні для байта від 0 до 255. При програмній реалізації цієї операції важливо враховувати типи даних. Оскільки значення пікселів P_{byte} зберігаються як .uint8, а елементи ключового потоку K_{byte} є стандартними цілими числами Python, для уникнення помилок переповнення та забезпечення коректної модульної арифметики, операнди перед додаванням явно перетворюються на стандартний цілочисельний тип int.

Для розшифрування в раундах, де застосовувалося модульне додавання, використовується обернена операція – модульне віднімання. Щоб коректно обробити випадки, коли $C_{byte} < K_{byte}$ (що призвело б до від'ємного результату перед взяттям модуля), до різниці додається 256 перед фінальною операцією за модулем 256:

$$P_{byte} = (C_{byte} - K_{byte} + 256) (\mathrm{mod}\ 256) \tag{3.8}$$

Такий підхід гарантує відновлення вихідного значення Pbyte. Чергування операцій XOR та модульного додавання в різних раундах шифрування є поширеною практикою, спрямованою на ускладнення структури шифру та підвищення його стійкості до різних методів криптоаналізу, оскільки атакуючому доведеться мати справу з комбінацією операцій з різними математичними властивостями.

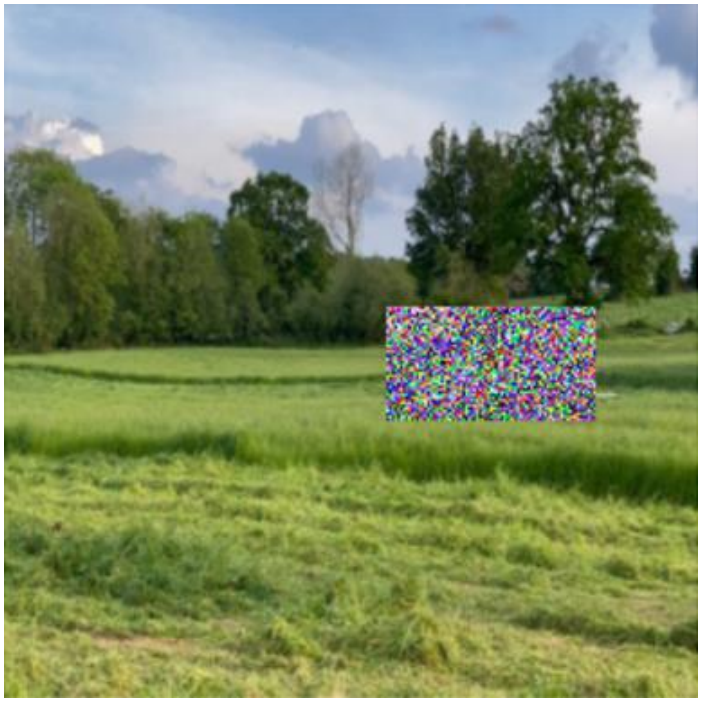

Рисунок 3.22 - використання модульного додавання на об’єкті зображення

3.2.5 Результати і дешифрування

Після послідовного застосування всіх дев'яти раундів криптографічних перетворень виділена область переднього плану на зображенні зазнає значних трансформацій. Вихідна візуальна інформація в цій області стає повністю нерозпізнаваною, перетворюючись на масив даних, що на вигляд нагадує випадковий шум. Приклад такого зашифрованого зображення, де чітко видно перетворену область на тлі незміненого основного зображення, що демонструє селективний характер методу, наведено на Рисунку 3.23. Проте частина зображення, яка не була обрана для шифрування, залишається в своєму первинному, незміненому вигляді. По завершенні процесу шифрування, окрім

самого модифікованого зображення, система також генерує та повертає два важливих елементи даних: унікальну криптографічну сіль, що використовувалася при генерації ключів, та HMAC-тег - код автентифікації повідомлення. Ці два компоненти є частиною зашифрованих даних і необхідні для подальшого розшифрування інформації.

Рисунок 3.23 – результат кодування

Процес розшифрування є симетричним до шифрування. Для нього використовуються наданий пароль, збережена під час шифрування сіль та відповідний HMAC-тег. Спочатку система перевіряє HMAC-тег для гарантії цілісності та автентичності отриманих даних; у разі успіху, застосовуються дев'ять обернених криптографічних раундів з використанням тих самих, детерміновано відновлених параметрів, для повного відновлення вихідних пікселів у зашифрованій області.

Рисунок 3.24 – дешифроване зображення

3.2.6 Огляд ефективності

3.2.6.1 Аналіз кореляції сусідніх пікселів

Одним із критеріїв оцінки якості алгоритму шифрування зображень є його здатність руйнувати статистичні залежності, притаманні вихідним даним. Природні зображення зазвичай характеризуються високим ступенем кореляції між сусідніми пікселями, тобто значення інтенсивності близько розташованих пікселів є схожими. Ефективний алгоритм шифрування має значно зменшувати цю кореляцію, наближаючи її значення до нуля, що ускладнює статистичні атаки та свідчить про хороший ефект дифузії – розсіювання впливу кожного біта відкритого тексту на багато бітів шифротексту.

Для оцінки цієї характеристики розробленого 9-раундового методу шифрування було проведено аналіз кореляції сусідніх пікселів. Дослідження проводилося для виділеної області інтересу (на зображенні, конвертованому у відтінки сірого, до та після застосування процедури шифрування. Кореляція обчислювалася для пар пікселів за трьома основними напрямками: горизонтальним, вертикальним та діагональним. Для кожного напрямку було

побудовано діаграми розсіювання значень інтенсивності пар пікселів та розраховано коефіцієнти кореляції Пірсона.

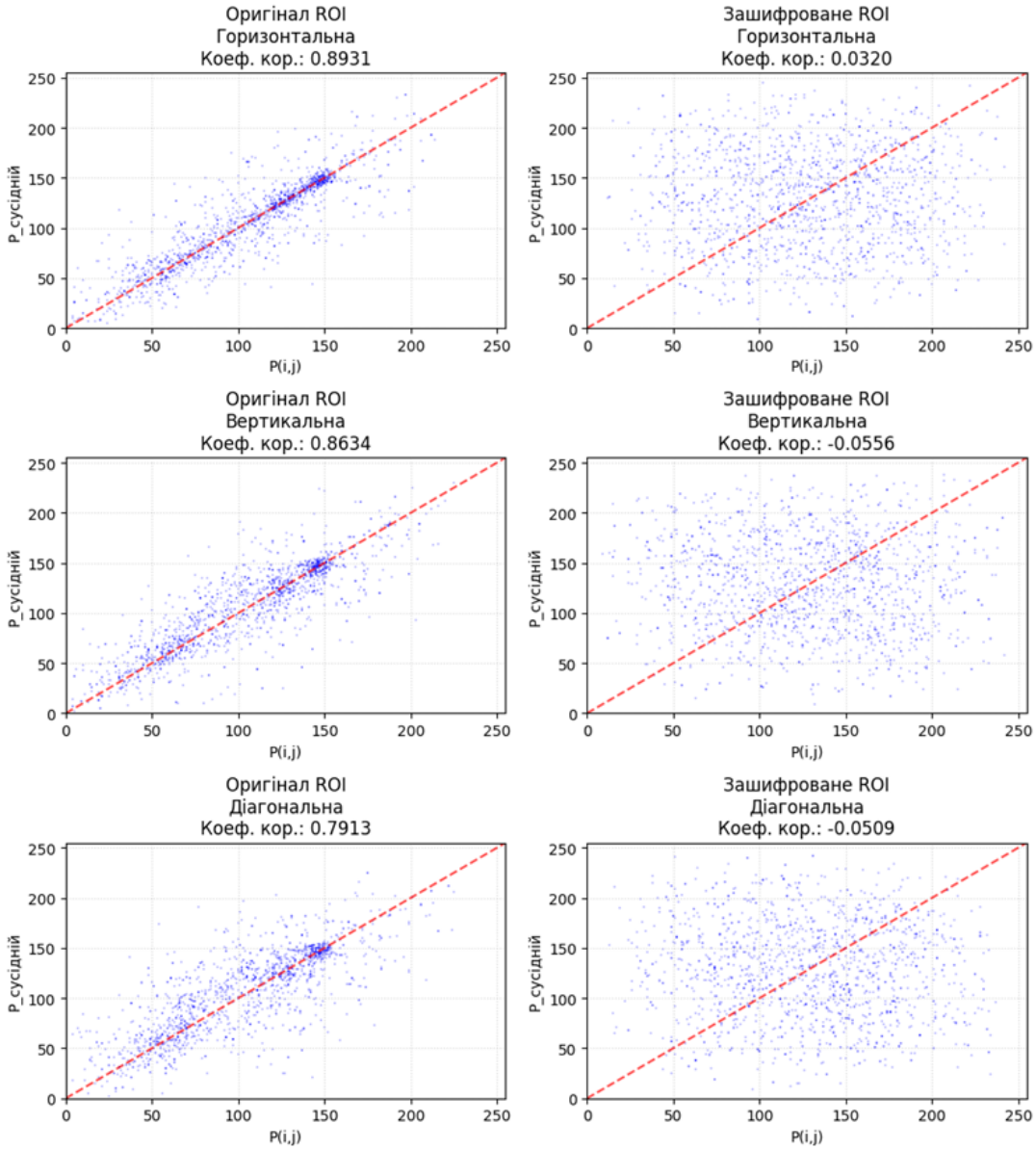

Рисунок 3.25 – графіки аналізу кореляції

Як видно з наведених даних, оригінальна область зображення демонструє очікувано високу позитивну кореляцію між сусідніми пікселями. Це вказує на на сильну статистичну залежність інтенсивностей сусідніх пікселів у вихідному зображенні, що візуально підтверджується скупченням точок вздовж головної діагоналі на відповідних діаграмах розсіювання.

Після застосування розробленого експериментального методу шифрування до тієї ж області ROI, картина кардинально змінюється. Коефіцієнти кореляції для зашифрованої області для тих самих напрямків стають надзвичайно близькими до нуля:

для горизонтального напрямку: 0.0320

для вертикального напрямку: -0.0556

для діагонального напрямку: -0.0509

Діаграми розсіювання для зашифрованої області демонструють хаотичний, рівномірно подібний розподіл точок, що вказує на практично повну відсутність лінійної залежності між інтенсивностями сусідніх пікселів.

Таким чином, отримані результати аналізу кореляції демонструють достатньо високу ефективність запропонованого 9-раундового алгоритму шифрування у руйнуванні статистичних зв'язків вихідного зображення. Значне зниження коефіцієнтів кореляції після шифрування свідчить про досягнення хорошого рівня дифузії, що є важливим показником криптографічної стійкості розробленого методу.

3.2.6.2 Аналіз гістограм інтенсивності колірних каналів

Для оцінки впливу розробленого методу шифрування на статистичний розподіл значень пікселів було проведено аналіз гістограм інтенсивності. Гістограма візуалізує частоту появи кожного рівня яскравості для колірних каналів зображення. Очікується, що ефективне шифрування перетворить характерні нерівномірні гістограми вихідного зображення на значно більш рівномірні, що свідчить про приховування статистичних особливостей оригіналу.

Аналіз проводився для виділеної області інтересу на зображенні до та після застосування 9-раундового шифрування.

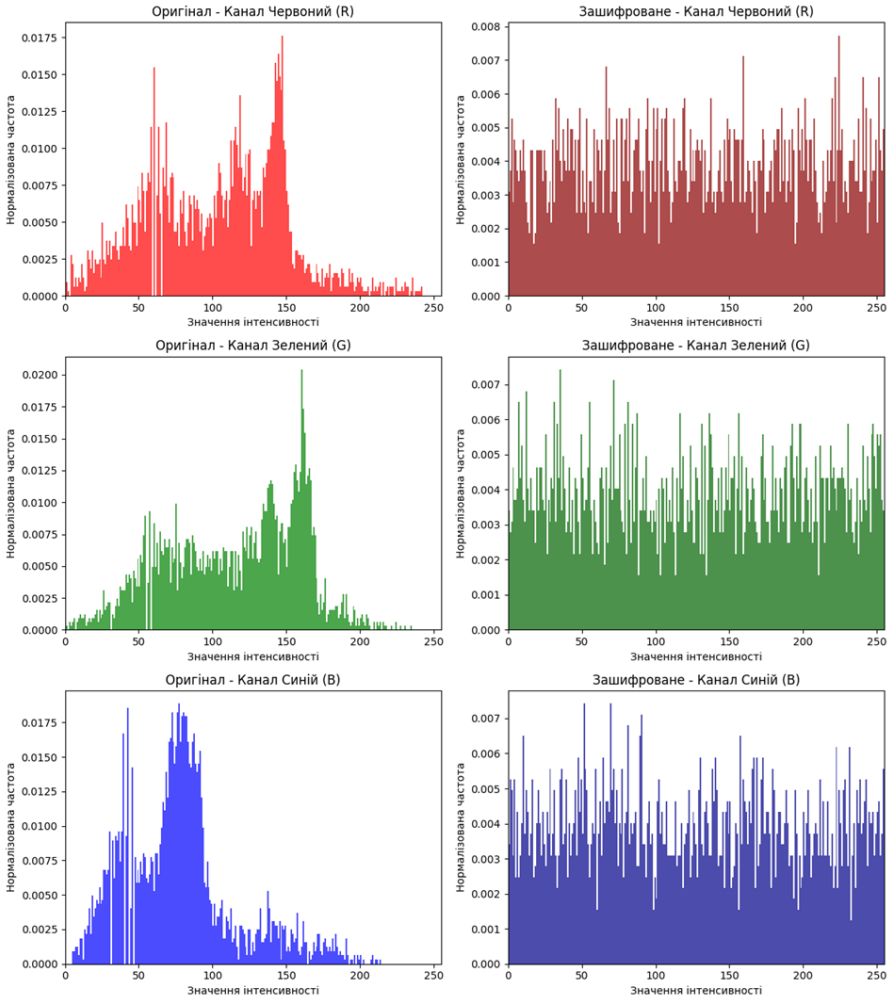

Рисунок 3.26 - гістограми інтенсивності колірних каналів для ROI до та після шифрування

Як видно з гістограм для кожного колірного каналу оригінальної області зображення демонструють нерівномірну структуру з характерними піками, що відображає специфіку колірної гами вихідної сцени. Натомість, після застосування шифрування, гістограми для всіх трьох каналів стають значно більш пласкими та рівномірними. Вихідні піки згладжуються, і значення інтенсивностей розподіляються по всьому діапазону [0, 255] без явних домінуючих рівнів.

Така зміна гістограм підтверджує, що розроблений метод шифрування досить ефективно руйнує вихідну статистичну структуру колірних каналів зображення. Рівномірний розподіл інтенсивностей у зашифрованій області значно ускладнює спроби відновлення інформації на основі аналізу частот

появи значень пікселів і є позитивним показником криптографічної якості алгоритму. Для подальшого кількісного аналізу цих змін доцільно також розглянути числові метрики, такі як середнє значення, стандартне відхилення інтенсивностей та інформаційна ентропія для кожного каналу.

3.2.6.3 Чутливість до змін у відкритому тексті

Для оцінки дифузійних властивостей розробленого шифру, тобто його здатності поширювати вплив невеликих змін у відкритому тексті на шифротекст, було розраховано стандартні метрики NPCR та UACI. Високі значення цих показників свідчать про хороший лавинний ефект. Експеримент полягав у порівнянні двох шифротекстів: одного, отриманого з оригінальної області інтересу, та іншого – з тієї ж ROI, але з одним попередньо зміненим пікселем. Обидва шифрування виконувалися з однаковими ключем та сіллю. Результати усереднювалися по 10 таких випробуваннях.

Усереднені показники для розробленого 9-раундового методу склали:

NPCR: 0.0103%

UACI: 0.0010%

Ці результати, є, м’яко кажучи, значно нижчими за оптимальні. Такий низький рівень дифузії пояснюється тим, що поточна структура раундових операцій – глобальна перестановка цілих пікселів з подальшою їх незалежною побайтовою заміною – призводить до того, що зміна одного пікселя відкритого тексту впливає лише на один відповідний піксель у шифротексті на кожному раунді. Хоча позиція та значення цього зміненого пікселя багаторазово модифікуються протягом раундів, сама зміна не поширюється на інші пікселі для створення каскадного ефекту.

Це вказує на обмежену здатність поточного алгоритму розсіювати локальні зміни у відкритому тексті, що є важливим аспектом для врахування. Дана характеристика може бути визначена як напрямок для подальших досліджень та потенційного вдосконалення методу. При цьому, як буде

показано нижче, шифр демонструє високу чутливість до змін у самому секретному ключі.

3.2.6.4 Аналіз чутливості шифротексту до змін у ключі

Важливою характеристикою надійного шифру є його висока чутливість до змін у секретному ключі. Для перевірки цієї властивості було проведено експеримент. Одне й те саме вихідне зображення шифрувалося двічі: перший раз з паролем P1 та сіллю S, отримуючи шифротекст C1; другий раз – з паролем P2 та тією ж сіллю S, отримуючи шифротекст C2. Для порівняння отриманих шифротекстів C1 та C2, як і в минулий раз, розраховувалися стандартні метрики NPCR - коефіцієнт зміни кількості пікселів та UACI - уніфікована середня зміна інтенсивності.

Результати такого аналізу, що візуально порівнюють отримані значення NPCR та UACI з їх теоретично оптимальними відповідниками, представлені на наступному рисунку.

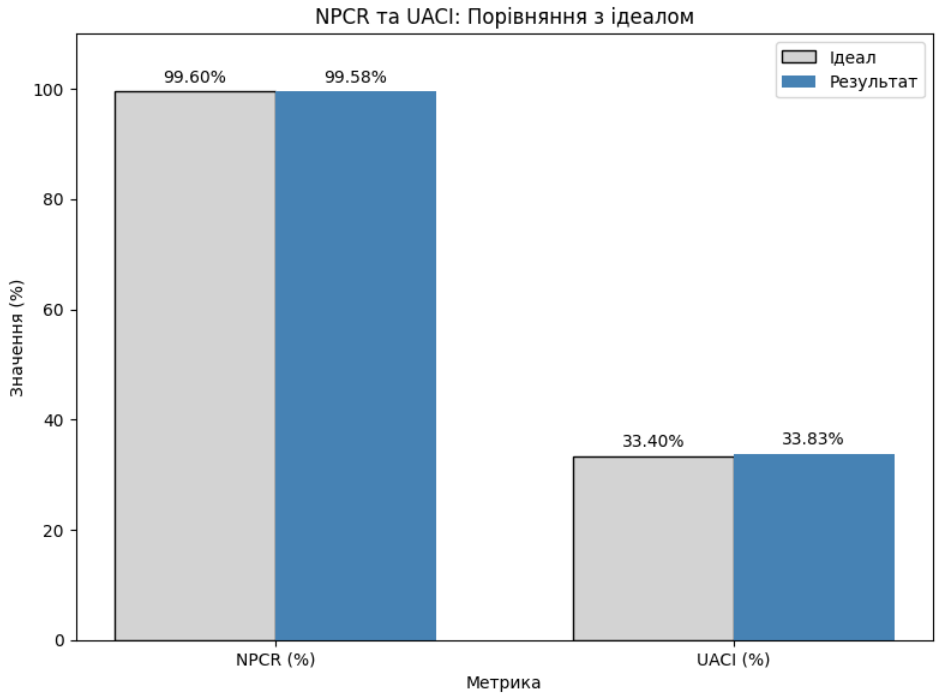

Рисунок 3.27 - порівнюють отриманих значень NPCR та UACI з їх теоретично оптимальними відповідниками

В ході одного з проведених тестів були зафіксовані такі показники:

NPCR (між C1 та C2): 99.6289%

UACI (між C1 та C2): 33.6325%

Ці значення дуже близькі до ідеальних теоретичних показників. Це свідчить про те, що зміна лише одного символу в паролі призводить до того, що переважна більшість пікселів у шифротексті змінює своє значення, причому ці зміни є суттєвими за величиною. Важливо зазначити, що хоча наведені цифри є результатом одного демонстраційного тесту, отримані дані чітко вказують на сильну залежність шифротексту від ключа. Така висока чутливість підтверджує ефективність використаної функції деривації ключів та багато-раундової структури шифрування у забезпеченні лавинного ефекту відносно ключа.

3.4 Результат роботи

Таким чином, в рамках цієї частини роботи була розроблена та описана програмна система для селективного захисту зображень. Продемонстровано, що нейромережевий модуль на базі U-Net забезпечує виділення областей інтересу, а запропонований 9-раундовий метод шифрування достатньо ефективно кодує ці ROI, демонструючи високу чутливість до змін ключа та здатність руйнувати вихідні статистичні властивості зображення. Водночас, виявлено потенціал для подальшого вдосконалення як точності сегментації для складних випадків, так і дифузійних властивостей шифру відносно змін у відкритому тексті.

ВИСНОВКИ

В рамках робот було вирішено завдання дослідження та розробки програмного інструменту для вибіркового захисту зображень. Запропонований підхід використовує можливості машинного навчання для аналізу візуального вмісту та ідентифікації важливих областей, до яких потім застосовується спеціально розроблений метод шифрування.

Було проведено аналіз існуючих методів захисту візуальної інформації, що дозволило обґрунтувати переваги селективного підходу та визначити напрямки для створення нового алгоритму. Для автоматичного виявлення областей інтересу була адаптована та навчена нейромережева модель архітектури U-Net. Тестування на наборі даних Stanford Background Dataset показало, що модель здатна адекватно виділяти об'єкти переднього плану, хоча точність може сильно знижуватися для сцен зі складною структурою.

Центральною частиною роботи стала розробка нового 9-раундового симетричного алгоритму шифрування для виділених ROI. Цей алгоритм перетворює наданий користувачем пароль та унікальну випадкову сіль на весь необхідний ключовий матеріал за допомогою стандартної функції посилення ключа. Кожен з дев'яти раундів шифрування включає дві основні операції: глобальну перестановку всіх пікселів в ROI та побайтову заміну їх значень. Для операції заміни використовуються ключові потоки, що генеруються системою зв'язаних хаотичних відображень, налаштування якої є індивідуальними для кожного раунду. При цьому тип операції заміни чергується між раундами. Для перевірки цілісності та автентичності зашифрованих даних використовується стандартний механізм коду автентифікації повідомлень.

Експериментальне дослідження властивостей розробленого шифру показало його працездатність. Статистичний аналіз підтвердив, що після шифрування значно знижується кореляція сусідніх пікселів та вирівнюються гістограми колірних каналів, що свідчить про руйнування вихідної структури

зображення. Алгоритм продемонстрував високу чутливість до змін у секретному ключі: навіть незначна модифікація пароля призводить до кардинально іншого шифротексту, а розшифрування неправильним ключем блокується завдяки перевірці автентичності. Разом з тим, аналіз чутливості до змін у самому зображенні виявив, що поточна структура раундів має обмежену здатність поширювати локальні зміни відкритого тексту на весь шифротекст.

Загалом, розроблений програмний інструмент підтвердив ефективність запропонованої концепції селективного захисту візуальної інформації.

Подальші покращення можуть бути спрямовані на вдосконалення точності нейромережевої сегментації і модифікацію та ускладнення алгоритму шифрування для покращення його дифузійних властивостей відносно змін у відкритому тексті.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Image file type and format guide. *MDN Web Docs*. *Media types and formats for image, audio, and video content*. URL: https://Https://developer.mozilla.org/en-US/docs/Web/Media/Guides/Formats/Image_types.
2. Codecs in common media types. *MDN Web Docs*. *Media types and formats for image, audio, and video content*. URL: https://developer.mozilla.org/en-US/docs/Web/Media/Guides/Formats/codecs_parameter.
3. Chapter 3 IMAGE AND VIDEO COMPRESSION. *KDK College of Engineering*. URL: https://www.kdkce.edu.in/pdf/RAB-8ETRX-DCE-U3-COMPRESSION.pdf.
4. Gimp raster to vector. *Kitchenfity*. 28.05.2024. URL: https://kitchenfity.weebly.com/blog/gimp-raster-to-vector.
5. Selective Encryption of VVC Encoded Video Streams for the Internet of Video Things // arXiv.
6. PyTorch: An Imperative Style, High-Performance Deep Learning Library // arXiv. URL: https://arxiv.org/pdf/1912.01703.
7. Adam: A Method for Stochastic Optimization // arXiv. URL: https://arxiv.org/abs/1412.6980.
8. Chaudhari R. E., Dhok S. B. Review of Fractal Transform based Image and Video Compression. *International Journal of Computer Applications*. 2012. 19 листоп. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d500876ac7b59028adbac3b1b994b1de71ac1b2b.
9. Ericsson Mobility Report. Ericsson Mobility Report November 2023 edition. *AIRlabs Australia*. 13.12.2023. URL: https://airlabs.at/wp-

content/uploads/2024/03/Ericsson_Vortrag_EMR-November2023_AIRLabs_20231213.pdf.

10. Ficzere P. The possibilities of intelligent manufacturing methods. *Research Gate*. 02.05.2020. URL: https://www.researchgate.net/publication/357738098_The_possibilities_of_intelligent_manufacturing_methods.

11. Goodfellow I., Bengio Y., Courville A. Deep Learning.

12. Gould S., Fulton R., Koller D. Decomposing a Scene into Geometric and Semantically Consistent Regions. *ResearchGate*. URL: https://www.researchgate.net/publication/224135950_Decomposing_a_Scene_into_Geometric_and_Semantically_Consistent_Regions.

13. Grech V. Publishing on the WWW. Part 9 - Video codecs and decompressors. *PubMed Central*. 05.10.2003. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC3232547/.

14. Jay Joshi, Upena D. Dalal. Selective encryption using ISMACryp in real time video streaming of H.264/AVC for DVB-H application. *ResearchGate*. URL: https://www.researchgate.net/publication/230886073_Selective_encryption_using_ISMACryp_in_real_time_video_streaming_of_H264AVC_for_DVB-H_application.

15. Krawczyk H., Bellare M., Canetti R. HMAC: Keyed-Hashing for Message Authentication. *www.rfc-editor.org*. URL: https://www.rfc-editor.org/rfc/rfc2104.html.

16. Ranjith Bhat, Raghu N. Comparative Analysis of CryptoGAN: Evaluating Quality Metrics and Security in GAN-based Image Encryption. *ResearchGate*. URL: https://www.researchgate.net/publication/385570884_CryptoGAN_a_new_frontier_in_generative_adversarial_network-_driven_image_encryption.

17. Ranjith B., Raghu N. CryptoGAN: a new frontier in generative adversarial network-driven image encryption. *IAES International Journal of Artificial Intelligence*. 2024. С. 1–8. URL: https://ijai.iaescore.com/index.php/IJAI/article/view/25206/14282.

18. Ronneberger O., Fischer P., Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *ResearchGate*. URL: https://www.researchgate.net/publication/305193694_U-Net_Convolutional_Networks_for_Biomedical_Image_Segmentation.

19. Steven H. Strogatz. Nonlinear Dynamics and Chaos. CRC Press, 2018.

20. Recommendation for Password-Based Key Derivation Part 1: Storage Applications / M. Sönmez Turan та ін. *NIST Special Publication 800-132*. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf.

21. Tang L. Methods for Encrypting and Decrypting MPEG Video Data Efficiently / Carnegie Mellon University. URL: https://dl.acm.org/doi/pdf/10.1145/244130.244209.

22. A Comparative Analysis on Blockchain versus Centralized Authentication Architectures for IoT-Enabled Smart Devices in Smart Cities: A Comprehensive Review, Recent Advances, and Future Research Directions / A. M. Usman та ін. *MDPI Open Access Journals*. URL: https://www.mdpi.com/1424-8220/22/14/5168.

23. Wu Y. NPCR and UACI Randomness Tests for Image Encryption April 2011 Authors: Yue Wu Raytheon BBN Technologies. *ResearchGate*. URL: https://www.researchgate.net/publication/259190481_NPCR_and_UACI_Randomness_Tests_for_Image_Encryption.

24. Xiaowu Li, Huiling Peng. Chaotic medical image encryption method using attention mechanism fusion ResNet model. *PubMed Central*. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC10373303/.

25. A novel image encryption method based on improved two-dimensional logistic mapping and DNA computing / Yuanlin Chen та ін. *Frontiers*. URL: https://www.frontiersin.org/journals/physics/articles/10.3389/fphy.2024.1469418/full.

ДОДАТОК А
ПРОГРАМНИЙ КОД НАВЧАННЯ МОДЕЛІ СЕМАНТИЧНОЇ СЕГМЕНТАЦІЇ

```
NUM_EPOCHS = 35
save_dir = "saved_models"

if not os.path.exists(save_dir):
    os.makedirs(save_dir)
    print(f"Створено папку для збереження моделей: {save_dir}")

def calculate_iou(preds, targets, num_classes,
ignore_index=None):
    if torchmetrics is None:
        return None, None

    if preds.ndim == 4:
        preds_indices = torch.argmax(preds, dim=1)
    else:
        preds_indices = preds

    preds_indices = preds_indices.long()
    targets = targets.long()

    preds_indices_cpu = preds_indices.cpu()
    targets_cpu = targets.cpu()

    try:
        iou_metric = torchmetrics.JaccardIndex(
            task="multiclass",
            num_classes=num_classes,
            ignore_index=ignore_index,
            average='none'
        )
        iou_per_class = iou_metric(preds_indices_cpu,
targets_cpu)

        valid_iou_values = []
        for i, iou_val in enumerate(iou_per_class):
            if ignore_index is None or i != ignore_index:
                if not torch.isnan(iou_val):
                    valid_iou_values.append(iou_val)

        if valid_iou_values:
            mean_iou =
torch.tensor(valid_iou_values).mean().item()
        else:
            mean_iou = 0.0

        return iou_per_class, mean_iou
```

```
    except Exception as e:
        print(f"Помилка при обчисленні IoU: {e}")
        return None, None

def train_one_epoch(model, dataloader, criterion, optimizer,
device):
    model.train()

    running_loss = 0.0
    processed_batches = 0

    epoch_start_time = time.time()

    for i, (images, masks) in enumerate(dataloader):
        images = images.to(device)
        masks = masks.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, masks)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        processed_batches += 1

        if (i + 1) % 4 == 0 or (i + 1) == len(dataloader):
            print(f"  Тренувальний батч
[{i+1}/{len(dataloader)}], Поточна втрата: {loss.item():.4f}")

    epoch_end_time = time.time()
    epoch_duration = epoch_end_time - epoch_start_time

    avg_epoch_loss = running_loss / processed_batches
    print(f"  Середня тренувальна втрата за епоху:
{avg_epoch_loss:.4f}")
    print(f"  Тривалість тренувальної епохи:
{epoch_duration:.2f} секунд")

    return avg_epoch_loss, epoch_duration


def validate_one_epoch(model, dataloader, criterion, device,
num_classes, ignore_index_for_iou=None):
    model.eval()

    running_loss = 0.0
    processed_batches = 0

    all_preds_for_iou = []
    all_targets_for_iou = []
```

```
    epoch_start_time = time.time()

    with torch.no_grad():
        for i, (images, masks) in enumerate(dataloader):
            images = images.to(device)
            masks = masks.to(device)

            outputs = model(images)
            loss = criterion(outputs, masks)

            running_loss += loss.item()
            processed_batches += 1

            preds_indices = torch.argmax(outputs, dim=1)
            all_preds_for_iou.append(preds_indices.cpu())
            all_targets_for_iou.append(masks.cpu())

            if (i + 1) % 4 == 0 or (i + 1) == len(dataloader):
                print(f"  Валідаційний батч
[{i+1}/{len(dataloader)}], Поточна втрата: {loss.item():.4f}")

    epoch_end_time = time.time()
    epoch_duration = epoch_end_time - epoch_start_time

    avg_epoch_loss = running_loss / processed_batches

    iou_per_class, mean_iou = None, None
    if all_preds_for_iou and all_targets_for_iou:
        all_preds_tensor = torch.cat(all_preds_for_iou, dim=0)
        all_targets_tensor = torch.cat(all_targets_for_iou,
dim=0)
        iou_per_class, mean_iou =
calculate_iou(all_preds_tensor, all_targets_tensor, num_classes,
ignore_index_for_iou)

    return avg_epoch_loss, epoch_duration, iou_per_class,
mean_iou


print(f"--- Початок тренування моделі на {NUM_EPOCHS} епох ---")
print(f"--- Пристрій: {device} ---")

train_losses_history = []
val_losses_history = []
miou_history = []

best_miou = 0.0
best_model_path_miou = ""

ignore_idx_iou = UNKNOWN_TARGET_ID if 'UNKNOWN_TARGET_ID' in
globals() else None
if ignore_idx_iou is not None:
```

```
    print(f"Для IoU буде ігноруватися клас з ID:
{ignore_idx_iou} ({TARGET_ID_TO_NAME.get(ignore_idx_iou, 'N/A')
if 'TARGET_ID_TO_NAME' in globals() else 'N/A'})")

total_training_start_time = time.time()

for epoch in range(NUM_EPOCHS):
    print(f"\nЕпоха {epoch+1}/{NUM_EPOCHS}")

    avg_train_loss, train_epoch_duration =
train_one_epoch(model, train_dataloader, criterion, optimizer,
device)
    train_losses_history.append(avg_train_loss)

    avg_val_loss, val_epoch_duration, iou_per_class, mean_iou =
validate_one_epoch(
        model, val_dataloader, criterion, device,
NUM_TARGET_CLASSES, ignore_idx_iou
    )
    val_losses_history.append(avg_val_loss)

    print(f"  Середня тренувальна втрата: {avg_train_loss:.4f}
(Тривалість: {train_epoch_duration:.2f}с)")
    print(f"  Середня валідаційна втрата: {avg_val_loss:.4f}
(Тривалість: {val_epoch_duration:.2f}с)")

    if mean_iou is not None:
        miou_history.append(mean_iou)
        print(f"  Середній IoU (mIoU) на валідації:
{mean_iou:.4f}")
        if iou_per_class is not None and 'TARGET_ID_TO_NAME' in
globals(): # Перевірка наявності TARGET_ID_TO_NAME
            print("  IoU по класах:")
            for class_id, iou_val in enumerate(iou_per_class):
                if ignore_idx_iou is None or class_id !=
ignore_idx_iou:
                    class_name = TARGET_ID_TO_NAME.get(class_id,
f"Клас {class_id}")
                    print(f"    {class_name}: {iou_val:.4f}")

        if mean_iou > best_miou:
            best_miou = mean_iou
            if best_model_path_miou and
os.path.exists(best_model_path_miou):
                try:
                    os.remove(best_model_path_miou)
                except OSError as e:
                    print(f"  Помилка видалення попередньої
найкращої моделі {best_model_path_miou}: {e}")

            best_model_path_miou = os.path.join(save_dir,
f"unet_best_miou_epoch{epoch+1}_miou{mean_iou:.4f}.pth")
            torch.save(model.state_dict(), best_model_path_miou)
```

```
            print(f"  => Нову найкращу модель (по mIoU)
збережено: {best_model_path_miou}")
    else:
        miou_history.append(0.0)
        print("  mIoU не обчислено (можливо, torchmetrics не
встановлено або виникла помилка).")

total_training_end_time = time.time()
total_training_duration = total_training_end_time -
total_training_start_time
print(f"\n--- Тренування моделі завершено за
{total_training_duration:.2f} секунд
({total_training_duration/60:.2f} хвилин) ---")

if best_model_path_miou:
    print(f"Найкращу модель (по mIoU) збережено у:
{best_model_path_miou} з mIoU: {best_miou:.4f}")
else:
    final_model_path = os.path.join(save_dir,
f"unet_final_epoch{NUM_EPOCHS}_train_loss{avg_train_loss:.4f}_va
l_loss{avg_val_loss:.4f}.pth")
    torch.save(model.state_dict(), final_model_path)
    print(f"Збережено останню модель: {final_model_path}")


plt.figure(figsize=(18, 6))

plt.subplot(1, 2, 1)
plt.plot(range(1, NUM_EPOCHS + 1), train_losses_history,
marker='o', label='Тренувальна втрата')
plt.plot(range(1, NUM_EPOCHS + 1), val_losses_history,
marker='o', label='Валідаційна втрата')
plt.xlabel('Епоха')
plt.ylabel('Втрата (CrossEntropy)')
plt.title('Історія Втрат')
plt.xticks(range(1, NUM_EPOCHS + 1))
plt.legend()
plt.grid(True)

if torchmetrics is not None and any(m > 0 for m in miou_history
if m is not None):
    plt.subplot(1, 2, 2)
    plt.plot(range(1, NUM_EPOCHS + 1), miou_history, marker='o',
label='mIoU на валідації', color='green')
    plt.xlabel('Епоха')
    plt.ylabel('mIoU')
    plt.title('Історія mIoU на Валідації')
    plt.xticks(range(1, NUM_EPOCHS + 1))
    plt.legend()
    plt.grid(True)

plt.tight_layout()
plt.show()
```

ДОДАТОК Б
ОСНОВНІ ФУНКЦІЇ МЕТОДУ КОДУВАННЯ

```
NUM_ENCRYPTION_ROUNDS = 9
CHAOS_PARAMS_EKM_LEN_PER_ROUND = 22
HMAC_KEY_LEN = 32
PBKDF2_ITERATIONS = 50000
SALT_LEN_BYTES = 16

def _derive_keys_and_params_pbkdf2(password: str, salt: bytes,
                                   num_rounds: int =
NUM_ENCRYPTION_ROUNDS,
                                   chaos_ekm_len_per_round: int
= CHAOS_PARAMS_EKM_LEN_PER_ROUND,
                                   hmac_key_len: int =
HMAC_KEY_LEN,
                                   iterations: int =
PBKDF2_ITERATIONS):

    total_chaos_ekm_len = chaos_ekm_len_per_round * num_rounds
    total_derived_key_len = total_chaos_ekm_len + hmac_key_len

    derived_material = hashlib.pbkdf2_hmac(
        'sha512',
        password.encode('utf-8'),
        salt,
        iterations,
        dklen=total_derived_key_len
    )

    ekm_for_all_chaos_rounds =
derived_material[:total_chaos_ekm_len]
    hmac_key = derived_material[total_chaos_ekm_len:]

    all_rounds_params = []
    ekm_offset = 0

    for r_idx_param_gen in range(num_rounds):
        params_this_round = {}
        current_param_offset_for_this_round_ekm = 0

        ekm_this_round = ekm_for_all_chaos_rounds[ekm_offset :
ekm_offset + chaos_ekm_len_per_round]
        ekm_offset += chaos_ekm_len_per_round

        def get_bytes_from_round_ekm_local(num_bytes):
            nonlocal current_param_offset_for_this_round_ekm
            if current_param_offset_for_this_round_ekm +
num_bytes > len(ekm_this_round):
                raise ValueError(
```

```
                    f"Недостатньо ЕКМ для параметрів раунду
{r_idx_param_gen + 1}: потрібно {num_bytes}б, "
                    f"доступно {len(ekm_this_round) -
current_param_offset_for_this_round_ekm} з
{len(ekm_this_round)}."
                )
            data_bytes =
ekm_this_round[current_param_offset_for_this_round_ekm :
current_param_offset_for_this_round_ekm + num_bytes]
            current_param_offset_for_this_round_ekm += num_bytes
            return data_bytes

        params_this_round['x0'] =
(int.from_bytes(get_bytes_from_round_ekm_local(4), 'big') %
99998 + 1) / 100000.0
        params_this_round['y0'] =
(int.from_bytes(get_bytes_from_round_ekm_local(4), 'big') %
99998 + 1) / 100000.0
        params_this_round['mu1'] = 1.8 +
(int.from_bytes(get_bytes_from_round_ekm_local(2), 'big') %
20000) / 100000.0
        params_this_round['mu2'] = 1.8 +
(int.from_bytes(get_bytes_from_round_ekm_local(2), 'big') %
20000) / 100000.0
        params_this_round['eps1'] = 0.001 +
(int.from_bytes(get_bytes_from_round_ekm_local(2), 'big') %
14901) / 100000.0
        params_this_round['eps2'] = 0.001 +
(int.from_bytes(get_bytes_from_round_ekm_local(2), 'big') %
14901) / 100000.0
        params_this_round['skip_iterations'] = 500 +
(int.from_bytes(get_bytes_from_round_ekm_local(2), 'big') %
2501)
        params_this_round['perm_seed'] =
int.from_bytes(get_bytes_from_round_ekm_local(4), 'big')

        if current_param_offset_for_this_round_ekm !=
chaos_ekm_len_per_round:
            print(f"Попередження: Для параметрів раунду
{r_idx_param_gen + 1} використано
{current_param_offset_for_this_round_ekm}б, "
                  f"очікувалося {chaos_ekm_len_per_round}б.")

        params_this_round['mu1'] =
min(max(params_this_round['mu1'], 1.8), 1.99999)
        params_this_round['mu2'] =
min(max(params_this_round['mu2'], 1.8), 1.99999)
        params_this_round['x0'] = max(1e-5,
min(params_this_round['x0'], 1.0 - 1e-5))
        params_this_round['y0'] = max(1e-5,
min(params_this_round['y0'], 1.0 - 1e-5))

        all_rounds_params.append(params_this_round)
```

```
    return all_rounds_params, hmac_key

def calculate_hmac_tag(key: bytes, data: bytes) -> bytes:
    h = hmac.new(key, data, digestmod=hashlib.sha256)
    return h.digest()

def verify_hmac_tag(key: bytes, data: bytes, expected_tag:
bytes) -> bool:
    calculated_tag = calculate_hmac_tag(key, data)
    return hmac.compare_digest(calculated_tag, expected_tag)

def _generate_chaotic_keystream_py_list(params: dict,
num_bytes_needed: int) -> list[int]:
    x_n, y_n = params['x0'], params['y0']
    mu1, mu2 = params['mu1'], params['mu2']
    eps1, eps2 = params['eps1'], params['eps2']
    keystream_bytes_list = []
    def tent_map_step(val, mu_param): val=max(0.0,min(1.0,val));
return mu_param*min(val,1.0-val)
    for _ in range(params['skip_iterations']):
        x_t=tent_map_step(x_n,mu1); y_t=tent_map_step(y_n,mu2)
        x_n1=(x_t+eps1*y_t); x_n1-=math.floor(x_n1)
        y_n1=(y_t+eps2*x_n1); y_n1-=math.floor(y_n1)
        x_n,y_n=x_n1,y_n1
    count_gen=0
    while count_gen < num_bytes_needed:
        x_t=tent_map_step(x_n,mu1); y_t=tent_map_step(y_n,mu2)
        x_n1=(x_t+eps1*y_t); x_n1-=math.floor(x_n1)
        y_n1=(y_t+eps2*x_n1); y_n1-=math.floor(y_n1)
        x_n,y_n=x_n1,y_n1
        keystream_bytes_list.append(int(x_n*255.999999))
        count_gen+=1
        if count_gen < num_bytes_needed:
            keystream_bytes_list.append(int(y_n*255.999999));
count_gen+=1
    return keystream_bytes_list[:num_bytes_needed]

def _permute_pixels_global(pixels_array: np.ndarray, seed: int,
decrypt: bool = False) -> np.ndarray:
    if pixels_array.size == 0: return pixels_array
    if not (pixels_array.ndim == 2 and pixels_array.shape[1] ==
3):
        if pixels_array.ndim == 1 and pixels_array.shape[0] % 3
== 0:
            pixels_to_permute = pixels_array.reshape(-1, 3)
        else: raise ValueError(f"Неправильна форма для
_permute_pixels_global: {pixels_array.shape}, очікується (N,3)
або плаский масив кратно 3.")
    else:
        pixels_to_permute = pixels_array
    rng = np.random.default_rng(seed=seed)
    num_pixels = pixels_to_permute.shape[0]
```

```
    p_indices = rng.permutation(num_pixels)
    if decrypt:
        inv_p_indices = np.empty_like(p_indices);
inv_p_indices[p_indices] = np.arange(num_pixels)
        output_pixels_2d = pixels_to_permute[inv_p_indices, :]
    else:
        output_pixels_2d = pixels_to_permute[p_indices, :]
    return output_pixels_2d

def encrypt_foreground(image_np: np.ndarray,
                       segmentation_mask_np: np.ndarray,
                       foreground_id: int,
                       method="chaotic_stream_cipher",
                       password=None,
                       decrypt=False,
                       encryption_mode="bounding_box",
padding=5,
                       salt_param: bytes = None,
                       hmac_tag_to_verify: bytes = None,
                       num_rounds: int = NUM_ENCRYPTION_ROUNDS
                       ):
    if image_np.max() <= 1.0 and (image_np.dtype == np.float32
or image_np.dtype == np.float64):
        image_uint8 = (image_np.copy() * 255).astype(np.uint8)
    elif image_np.dtype == np.uint8:
        image_uint8 = image_np.copy()
    else:
        temp_img = image_np.copy().astype(np.float64)
        if temp_img.min() < -1e-6 or temp_img.max() > 255.0 +
1e-6 or \
           (temp_img.max() > 1.0 + 1e-6 and
np.any(np.abs(temp_img - np.round(temp_img)) > 1e-6)):
            min_v, max_v = np.percentile(temp_img,1),
np.percentile(temp_img,99)
            if abs(max_v - min_v) < 1e-9 : temp_img.fill(0.0)
            else: temp_img = (temp_img - min_v) / (max_v -
min_v)
            temp_img = np.clip(temp_img, 0.0, 1.0)
            image_uint8 = (temp_img * 255).astype(np.uint8)
        elif temp_img.max() <= 1.0 + 1e-6 and temp_img.min() >=
-1e-6 :
            image_uint8 = (np.clip(temp_img, 0.0, 1.0) *
255).astype(np.uint8)
        else:
            image_uint8 = np.clip(temp_img, 0,
255).astype(np.uint8)

    processed_image_uint8 = image_uint8.copy()

    active_salt = None
    if not decrypt:
        active_salt = salt_param if salt_param is not None else
os.urandom(SALT_LEN_BYTES)
```

```
    elif salt_param is not None:
        active_salt = salt_param

    base_foreground_mask = (segmentation_mask_np ==
foreground_id)
    if not np.any(base_foreground_mask):
        if decrypt: return processed_image_uint8
        return {'encrypted_image': processed_image_uint8,
'salt': active_salt, 'hmac_tag': None}

    final_encryption_mask = np.zeros_like(base_foreground_mask,
dtype=bool)
    if encryption_mode == "mask":
        final_encryption_mask = base_foreground_mask
    elif encryption_mode == "bounding_box":
        if not SCIPY_AVAILABLE:
            print("ПОМИЛКА SciPy: Використовую шифрування по
масці.")
            final_encryption_mask = base_foreground_mask
        else:
            labeled_mask, num_features_ndi =
ndi.label(base_foreground_mask)
            if num_features_ndi == 0:
                if decrypt: return processed_image_uint8
                return {'encrypted_image':
processed_image_uint8, 'salt': active_salt, 'hmac_tag': None}
            img_h, img_w = image_uint8.shape[:2]
            any_box_created = False
            for i_box in range(1, num_features_ndi + 1):
                obj_px = np.argwhere(labeled_mask == i_box)
                if obj_px.size > 0:
                    y_min, x_min = obj_px.min(axis=0); y_max,
x_max = obj_px.max(axis=0)
                    y_mp, x_mp = max(0, y_min - padding), max(0,
x_min - padding)
                    y_xp, x_xp = min(img_h - 1, y_max +
padding), min(img_w - 1, x_max + padding)
                    final_encryption_mask[y_mp:y_xp + 1,
x_mp:x_xp + 1] = True
                    any_box_created = True
            if not any_box_created:
                if decrypt: return processed_image_uint8
                return {'encrypted_image':
processed_image_uint8, 'salt': active_salt, 'hmac_tag': None}
    else:
        print(f"Попередження: Невідомий encryption_mode
'{encryption_mode}'. Використовую шифрування по масці.")
        final_encryption_mask = base_foreground_mask

    current_fg_pixels_np =
processed_image_uint8[final_encryption_mask].copy()
    if current_fg_pixels_np.size == 0:
        if decrypt: return processed_image_uint8
```

```
        return {'encrypted_image': processed_image_uint8,
'salt': active_salt, 'hmac_tag': None}

    if method == "chaotic_stream_cipher":
        if password is None or not isinstance(password, str) or
len(password) == 0:
            raise ValueError("Пароль не надано або має невірний
тип.")

        if decrypt and active_salt is None:
             raise ValueError("Сіль (salt_param) обов'язкова для
розшифрування.")

        all_rounds_params, hmac_key =
_derive_keys_and_params_pbkdf2(
            password, active_salt, num_rounds=num_rounds,
iterations=PBKDF2_ITERATIONS
        )

        processed_pixels_data = current_fg_pixels_np.copy()

        if not decrypt:
            for i_round in range(num_rounds):
                round_params = all_rounds_params[i_round]
                processed_pixels_data =
_permute_pixels_global(processed_pixels_data,
round_params['perm_seed'], decrypt=False)

                num_pixels_in_data =
processed_pixels_data.shape[0]
                num_bytes_needed_for_round = num_pixels_in_data
* 3
                keystream_round =
_generate_chaotic_keystream_py_list(round_params,
num_bytes_needed_for_round)
                if len(keystream_round) !=
num_bytes_needed_for_round:
                    raise ValueError(f"Раунд {i_round+1} (шифр):
Довжина потоку ({len(keystream_round)}) != даних
({num_bytes_needed_for_round})")

                temp_round_output =
np.empty_like(processed_pixels_data)
                keystream_idx = 0
                for r_idx_pixel in range(num_pixels_in_data):
                    if i_round % 2 == 0:
                        temp_round_output[r_idx_pixel, 0] =
processed_pixels_data[r_idx_pixel, 0] ^
keystream_round[keystream_idx]; keystream_idx += 1
                        temp_round_output[r_idx_pixel, 1] =
processed_pixels_data[r_idx_pixel, 1] ^
keystream_round[keystream_idx]; keystream_idx += 1
```

```
                        temp_round_output[r_idx_pixel, 2] =
processed_pixels_data[r_idx_pixel, 2] ^
keystream_round[keystream_idx]; keystream_idx += 1
                    else:
                        px_r =
int(processed_pixels_data[r_idx_pixel, 0])
                        ks_r =
int(keystream_round[keystream_idx]); keystream_idx += 1
                        temp_round_output[r_idx_pixel, 0] =
(px_r + ks_r) % 256

                        px_g =
int(processed_pixels_data[r_idx_pixel, 1])
                        ks_g =
int(keystream_round[keystream_idx]); keystream_idx += 1
                        temp_round_output[r_idx_pixel, 1] =
(px_g + ks_g) % 256

                        px_b =
int(processed_pixels_data[r_idx_pixel, 2])
                        ks_b =
int(keystream_round[keystream_idx]); keystream_idx += 1
                        temp_round_output[r_idx_pixel, 2] =
(px_b + ks_b) % 256
                processed_pixels_data = temp_round_output

            final_encrypted_pixels = processed_pixels_data
            processed_image_uint8[final_encryption_mask] =
final_encrypted_pixels
            hmac_tag = calculate_hmac_tag(hmac_key,
final_encrypted_pixels.tobytes())
            return {'encrypted_image': processed_image_uint8,
'salt': active_salt, 'hmac_tag': hmac_tag}

        else:
            if hmac_tag_to_verify is None:
                raise ValueError("HMAC тег потрібен для
розшифрування та перевірки.")

            if not verify_hmac_tag(hmac_key,
current_fg_pixels_np.tobytes(), hmac_tag_to_verify):
                raise ValueError("Перевірка HMAC не пройдена!
Дані пошкоджено, ключ або сіль невірні.")

            for i_round in range(num_rounds - 1, -1, -1):
                round_params = all_rounds_params[i_round]

                num_pixels_in_data =
processed_pixels_data.shape[0]
                num_bytes_needed_for_round = num_pixels_in_data
* 3
```

```
                keystream_round =
_generate_chaotic_keystream_py_list(round_params,
num_bytes_needed_for_round)
                if len(keystream_round) !=
num_bytes_needed_for_round:
                     raise ValueError(f"Раунд {i_round+1}
(розшифр.): Довжина потоку ({len(keystream_round)}) != даних
({num_bytes_needed_for_round})")

                temp_round_output =
np.empty_like(processed_pixels_data)
                keystream_idx = 0
                for r_idx_pixel in range(num_pixels_in_data):
                    if i_round % 2 == 0:
                        temp_round_output[r_idx_pixel, 0] =
processed_pixels_data[r_idx_pixel, 0] ^
keystream_round[keystream_idx]; keystream_idx += 1
                        temp_round_output[r_idx_pixel, 1] =
processed_pixels_data[r_idx_pixel, 1] ^
keystream_round[keystream_idx]; keystream_idx += 1
                        temp_round_output[r_idx_pixel, 2] =
processed_pixels_data[r_idx_pixel, 2] ^
keystream_round[keystream_idx]; keystream_idx += 1
                    else:
                        px_r =
int(processed_pixels_data[r_idx_pixel, 0])
                        ks_r =
int(keystream_round[keystream_idx]); keystream_idx += 1
                        temp_round_output[r_idx_pixel, 0] =
(px_r - ks_r + 256) % 256

                        px_g =
int(processed_pixels_data[r_idx_pixel, 1])
                        ks_g =
int(keystream_round[keystream_idx]); keystream_idx += 1
                        temp_round_output[r_idx_pixel, 1] =
(px_g - ks_g + 256) % 256

                        px_b =
int(processed_pixels_data[r_idx_pixel, 2])
                        ks_b =
int(keystream_round[keystream_idx]); keystream_idx += 1
                        temp_round_output[r_idx_pixel, 2] =
(px_b - ks_b + 256) % 256
                processed_pixels_data = temp_round_output

                processed_pixels_data =
_permute_pixels_global(processed_pixels_data,
round_params['perm_seed'], decrypt=True)
            recovered_pixels_np = processed_pixels_data
            processed_image_uint8[final_encryption_mask] =
recovered_pixels_np
            return processed_image_uint8
```