**Міністерство освіти і науки України**
**Луцький національний технічний університет**
**Факультет комп'ютерних та інформаційних технологій**
**Кафедра інженерії програмного забезпечення**

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

# РОЗРОБКА ДОДАТКУ ДЛЯ АНАЛІЗУ ТА РЕДАГУВАННЯ ЗОБРАЖЕННЯ З ВИКОРИСТАННЯМ PYTHON ТА ІНТЕГРАЦІЄЮ ШТУЧНОГО ІНТЕЛЕКТУ

# DEVELOPMENT OF AN APPLICATION FOR IMAGE ANALYSIS AND EDITING USING PYTHON WITH ARTIFICIAL INTELLIGENCE INTEGRATION

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-41
**Близнюк Максим Сергійович**

Керівник:
**Сичук Віктор Анатолійович**

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна
_____

Луцьк – 2025 року

**ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ

Завідувач кафедри
_____
«*10*» ___*12*___ 20*24* р.

**ЗАВДАННЯ**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ**

Близнюку Максиму Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка додатку для аналізу та редагування зображення з використанням Python та інтеграцією штучного інтелекту»*
Керівник роботи: *к.т.н. доцент Сичук В. А.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*

3. Вихідні дані до роботи: *Python, бібліотекf PyQt5 та PyTorch, torchvision, diffusers та Pillow, методичні вказівки до виконання кваліфікаційної роботи бакалавра*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз предметної області, формування вимог, UML-діаграми, стек технологій, реалізація додатку, тестування системи, розробка інсталяційного пакета, підготовка документації*

5. Перелік графічного матеріалу: *16 рисунків*

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Аналіз предметної області | *Сичук В. А.* | | |
| Специфікація вимог до розробленої системи | *Сичук В. А.* | | |
| Розробка об'єкта проектування | *Сичук В. А.* | | |
| Нормоконтроль | *Вознюк А. В.* | | |
| Гарант ОП | *Ліщина Н. М.* | | |
| Показник запозичень тексту | 1,53 % | | |
| Академічна доброчесність | *Сичук В. А.* | | |

7. Дата видачі завдання *«20» грудня 2024 р.*

| № | Назва етапів кваліфікаційної роботи бакалавра | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Огляд літературних джерел по темі кваліфікаційної роботи бакалавра | до 04.02.2025 р. | виконано |
| 2 | Аналіз проблеми розробки та впровадження об'єкту проектування | до 01.03.2025 р. | виконано |
| 3 | Обгрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання | до 15.03.2025 р. | виконано |
| 4 | Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних | до 29.03.2025 р. | виконано |
| 5 | Практична реалізація об'єкта проектування та розробка бази даних | до 26.04.2025 р. | виконано |
| 6 | Тестування та налагодження об'єкта проектування | до 03.05.2025 р. | виконано |
| 7 | Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру | до 10.06.2025 р. | виконано |

Здобувач вищої освіти _____ Максим БЛИЗНЮК

Керівник кваліфікаційної роботи _____ Віктор СИЧУК

# АНОТАЦІЯ

Близнюк М. С. Розробка додатку для аналізу та редагування зображення з використанням Python та інтеграцією штучного інтелекту. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності 121 Інженерія програмного забезпечення. Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків та списку використаних джерел.

У першому розділі проаналізовано сучасні підходи до обробки зображень із застосуванням штучного інтелекту, а також обґрунтовано актуальність створення локального настільного додатку, що забезпечує конфіденційність і автоматизовану обробку візуального контенту. У другому розділі сформовано функціональні та нефункціональні вимоги, обґрунтовано вибір технологій, зокрема Python, PyQt5, PyTorch, torchvision та diffusers, а також представлено архітектурне рішення з використанням UML-діаграм. У третьому розділі описано практичну реалізацію системи: створення користувацького інтерфейсу, впровадження детекції об’єктів за допомогою моделі Faster R-CNN, реалізація генеративного інпейнтінгу на базі Stable Diffusion Inpaint, забезпечення стабільної роботи завдяки багатопотоковості, логуванню та обробці винятків.

У висновках підбито підсумки кожного етапу реалізації розробки, акцентовано на практичній цінності створеного рішення для редагування зображень із застосуванням штучного інтелекту.

Ключові слова: обробка зображень, штучний інтелект, детекція об’єктів, генеративний інпейнтінг, Stable Diffusion, Faster R-CNN, PyQt5, Python, інсталятор.

# ABSTRACT

Blyzniuk M. Development of an Application for Image Analysis and Editing Using Python with Artificial Intelligence Integration. Manuscript. Qualification work of the bachelor's program "Software Engineering" in the specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, and a list of references.

The first chapter analyzes modern approaches to image processing using artificial intelligence and justifies the relevance of developing a local desktop application that ensures confidentiality and automates visual content editing. The second chapter defines the functional and non-functional requirements, justifies the selection of technologies including Python, PyQt5, PyTorch, torchvision, and diffusers, and presents the architectural design supported by UML diagrams. The third chapter describes the practical implementation of the system: creating a user interface, object detection using the Faster R-CNN model, generative inpainting based on Stable Diffusion Inpaint, and ensuring robust performance through multithreading, logging, and exception handling mechanisms.

The conclusions summarize the outcomes of each stage of the development process and highlight the practical value of the created solution for AI-powered image editing.

Keywords: image processing, artificial intelligence, object detection, generative inpainting, Stable Diffusion, Faster R-CNN, PyQt5, Python, installer.

# ЗМІСТ

# ВСТУП

Актуальність теми. В епоху тотальної цифровізації та стрімкого зростання обсягів візуального контенту, зображення перетворилися на один з ключових засобів комунікації та самовираження. Щодня генерується та обробляється величезна кількість графічних даних, що постійно зростає із поширенням соціальних мереж, цифрової фотографії, онлайн-медіа та використанням у професійних галузях. Це створює нагальну потребу в ефективних та інтуїтивно зрозумілих інструментах для аналізу, модифікації та покращення зображень. Традиційні методи редагування, доступні у класичних графічних редакторах, таких як Adobe Photoshop або GIMP, значною мірою базуються на ручних операціях, що вимагає значних зусиль, часу. Аналогічно, процеси аналізу зображень, наприклад, автоматична детекція або класифікація об’єктів, при великих обсягах даних стають рутинними та неефективними без належної автоматизації. Зі стрімким розвитком технологій штучного інтелекту, зокрема глибокого навчання, відкриваються принципово нові, більш потужні та автоматизовані підходи до обробки зображень.

Такі галузі, як комп’ютерний зір та генеративні моделі, дозволяють не лише автоматизувати рутинні операції, але й виконувати маніпуляції зі зображеннями, які раніше були неможливими або надзвичайно складними. Зокрема, технології детекції об’єктів та генеративного інпейнтінгу відкривають шлях до революційних змін у способах взаємодії з візуальним контентом. Проте, існуючі рішення часто мають суттєві обмеження: професійні редактори надають лише базовий функціонал, а потужні онлайн-сервіси, що базуються на хмарних генеративних моделях, обтяжені платними підписками, вимагають постійного інтернет-з’єднання та породжують проблеми конфіденційності даних. Таким чином, розробка локального, доступного та водночас високоякісного інструменту, який поєднує можливості автоматичної детекції об’єктів та інтелектуального інпейнтінгу, є нагальним та актуальним науково-практичним завданням.

Мета кваліфікаційної роботи полягає у розробці додатку для аналізу та редагування зображення з використанням Python та інтеграцією штучного інтелекту.

Об’єктом кваліфікаційної роботи є додаток для аналізу та редагування зображення. Він буде здійснювати автоматичне визначення об’єктів на зображенні та їх подальше редагування за допомогою генеративного інпейнтінгу на основі текстового опису.

Предметом кваліфікаційної роботи є практична реалізація додатку для аналізу та редагування зображення, яка базується на використанні Python.

Для досягнення мети було поставленно наступні завдання:

– здійснити аналіз методів обробки зображень;

– визначити вимоги;

– побудувати UML-діаграми;

– обрати стек технологій;

– реалізувати додаток;

– протестувати розроблений додаток;

– розробити інсталяційний пакет;

– розробити технічну документацію.

# РОЗДІЛ 1
# АНАЛІЗ СУЧАСНИХ МЕТОДІВ ОБРОБКИ ТА РЕДАГУВАННЯ ЗОБРАЖЕНЬ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

## 1.1 Аналіз сучасного стану проблеми

В епоху цифрових технологій зображення стали одним з основних способів передачі інформації та самовираження. Щодня створюється та обробляється величезна кількість візуального контенту, що постійно зростає із поширенням соціальних мереж. Це створює зростаючу потребу в ефективних інструментах для аналізу, модифікації та покращення зображень.

Традиційні методи редагування зображень, доступні у класичних графічних редакторах (наприклад, Adobe Photoshop, GIMP), значною мірою базуються на ручних операціях. Такі завдання, як видалення або заміна об’єктів на складному фоні, вимагають більших зусиль, часу. Процеси аналізу зображень, наприклад, підрахунок однотипних об’єктів або виявлення певних елементів, також можуть бути рутинними та неефективними при великих обсягах даних.

З розвитком технологій штучного інтелекту, зокрема глибокого навчання (Deep Learning), з’явилися нові, більш потужні та автоматизовані підходи до обробки зображень. Комп’ютерний зір, як галузь ШІ, досяг значного прогресу в задачах класифікації, визначення, сегментації та генерації зображень. Це відкриває перспективи для створення інструментів, які можуть автоматизувати або спростити складні операції редагування та аналізу.

Аналіз науково-технічної літератури та існуючих рішень показує, що активно досліджуються та розробляються методи, що використовують глибокі нейронні мережі для вирішення конкретних завдань обробки зображень. Зокрема, значний прогрес досягнуто у сферах:

Визначення об’єктів (Object Detection) – задача визначення наявності та місцезнаходження (зазвичай за допомогою обмежувального прямокутника)

одного або кількох об’єктів певних класів на зображенні. Сучасні моделі, такі як Faster R-CNN [2], YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector) , показали високу точність та швидкість на великих бенчмарках, таких як COCO [1] та PASCAL VOC. Ці моделі здатні автоматично ідентифікувати об’єкти, що раніше вимагало ручної анотації або використання простіших, менш надійних методів. Перевагою є їхня здатність обробляти зображення з різноманітними об’єктами та сценами. Недоліками можуть бути високі обчислювальні вимоги та чутливість до якості зображення або нетипових ракурсів.

Генерація та модифікація зображень (Image Generation and Manipulation) – галузь, що охоплює створення нових зображень з нуля, зміну стилю існуючих зображень, а також їхнє редагування, включаючи завдання інпейнтінгу (inpainting) – заповнення пропущених або видалених областей зображення реалістичним вмістом [3]. Останніми роками особливу популярність здобули генеративні моделі, такі як Генеративно-змагальні мережі (GANs) та, особливо, Дифузійні моделі (Diffusion Models). Останні, зокрема їхні латентні варіанти, як Stable Diffusion, продемонстрували вражаючі можливості у створенні різноманітних високоякісних зображень за текстовим описом (text-to-image) та у задачах інпейнтінгу. Перевагою цих моделей є висока якість та реалістичність згенерованого контенту, а також можливість керувати процесом генерації за допомогою тексту та масок. Недоліками є надзвичайно високі обчислювальні вимоги, особливо для навчання та виконання інференсу на високоякісних зображеннях, а також необхідність у значних обсягах даних для навчання.

Існують різноманітні програмні продукти та онлайн-сервіси, що інтегрують окремі функції на основі штучного інтелекту, проте їхній функціонал часто має певні обмеження. Наприклад, професійні графічні редактори, такі як Adobe Photoshop або GIMP [4], хоча й містять функції типу «Content-Aware Fill», здебільшого базуються на простіших алгоритмах інпейнтінгу або базових елементах машинного навчання, що не завжди забезпечує високу якість результату на складних зображеннях. З іншого боку,

онлайн-сервіси, що функціонують на великих генеративних моделях (наприклад, DALL-E 2, Midjourney), надають широкі можливості текст-керованої генерації та інпейнтінгу, але їхнє використання часто обмежується платною підпискою, а обробка даних відбувається на віддалених серверах, що може викликати занепокоєння щодо конфіденційності, швидкості та стабільності з'єднання.

На відміну від цих рішень, розроблений у межах даної кваліфікаційної роботи програмний продукт є комплексним, зручним та ефективним інструментом для автоматизованої обробки й редагування зображень, що поєднує в собі переваги потужних ШІ-моделей з гнучкістю локального виконання. Так, уся обробка зображень відбувається безпосередньо на пристрої користувача, що гарантує повну конфіденційність даних і незалежність від доступу до мережі, усуваючи ризики, пов'язані з передачею чутливої інформації на сторонні сервери. Крім того, додаток інтегрує найсучасніші ШІ-моделі для розпізнавання об'єктів, зокрема MobileNetV3 + SSD, та генерації зображень за допомогою Stable Diffusion, що дозволяє досягати значно вищої якості інпейнтінгу та гнучкості в маніпуляціях із зображеннями порівняно з обмеженим функціоналом традиційних редакторів. Оптимізована архітектура та використання апаратних ресурсів користувача також забезпечують високу швидкість обробки, що є критично важливим для пакетної роботи з зображеннями або великими файлами, де онлайн-сервіси часто стикаються із затримками. Важливою перевагою є також відсутність підписок, що робить передові ШІ-технології доступними для широкого кола користувачів без додаткових фінансових витрат.

Аналіз існуючих аналогів показує, що незважаючи на наявність потужних AI-моделей та окремих інструментів, існує потреба у створенні локального, настільного додатку з дружнім інтерфейсом, який інтегрував би функціонал автоматичної детекції об'єктів та їхньої заміни за текстовим описом. Більшість наявних рішень або є вузькоспеціалізованими, або вимагають

онлайн-підключення та оплати, або не надають зручного механізму виділення об’єктів для подальшого редагування на основі їх автоматичного розпізнавання.

Розробка нового додатку, який поєднує ці можливості в єдиному інтерфейсі, має на меті зробити потужні інструменти аналізу та генеративного редагування зображень більш доступними для широкого кола користувачів, дозволяючи автоматизувати процес ідентифікації елементів на зображенні та їх творчої модифікації. Такий додаток може знайти застосування як у професійній діяльності (ретуш фотографій, створення візуального контенту), так і для освітніх та творчих цілей. Обґрунтуванням доцільності створення нової системи є можливість надати користувачу єдиний, локально працюючий інструмент, що мінімізує ручні операції та використовує переваги сучасних моделей глибокого навчання для виконання складних задач аналізу та редагування зображень. Ну а можливість обрати потрібну модель для генерації прямо в додатку – дає доступ до оптимізації під конкретні задачі.

## 1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Для успішної розробки додатку для аналізу та редагування зображення необхідно виконати наступні конкретні завдання:

– здійснити аналіз сучасних методів обробки та редагування зображень;

– визначити функціональні та нефункціональні вимоги до розроблюваного додатку, побудувати UML-діаграми для систематизації процесу розробки;

– обрати стек технологій для реалізації додатку, описати ключові алгоритми роботи;

– здійснити практичну реалізації додатку, який буде відповідати всім поставленим вимогам;

– провести тестування розробленого додатку для оцінки його функціональності, якості роботи інтегрованих моделей та продуктивності;

– написати технічну документацію для додатку, з описом мінімальних та рекомендованих системних вимог;

– описати процес компіляції виконавчого файлу для інсталяції додатку.

**Висновки до розділу 1**

У першому розділі кваліфікаційної роботи проведено сучасних методів обробки та редагування зображень. Визначено, що традиційні підходи мають суттєві обмеження, які можуть бути подолані за рахунок використання сучасних методів штучного інтелекту. Проведено огляд ключових напрямків розвитку комп'ютерного зору, зокрема методів детекції об'єктів (Faster R-CNN) та генеративного редагування (Stable Diffusion Inpainting), які продемонстрували високу ефективність у вирішенні відповідних задач.

Проаналізовано існуючі програмні рішення та сервіси, що використовують технології ШІ для обробки зображень, та виявлено, що існує потреба у створенні локального, настільного додатку, який поєднує функціонал автоматичної детекції об'єктів та їх заміни за текстовим описом у зручному для користувача інтерфейсі. Обґрунтовано доцільність розробки такої системи.

Визначено сім конкретних завдань, які необхідно вирішити в процесі виконання кваліфікаційної роботи для досягнення поставленої мети. Ці завдання охоплюють як теоретичні аспекти вибору та інтеграції моделей ШІ, так і практичну розробку програмного забезпечення та його тестування.

# РОЗДІЛ 2
# СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

## 2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Етап аналізу та формулювання вимог є фундаментальним для успішної розробки будь-якого програмного забезпечення. Саме на цьому етапі закладається основа для ефективного функціонування системи, забезпечується її відповідність потребам користувачів та бізнес-цілям, а також мінімізуються ризики виникнення помилок та невідповідностей на подальших етапах життєвого циклу проєкту. Чітке визначення того, що має робити система, які її функції та як вона повинна взаємодіяти з користувачем, дозволяє розробити архітектуру, що ефективно підтримуватиме цілі та задачі, а також спрощує процес тестування та впровадження.

Вимоги до розроблюваного додатку були виявлені та проаналізовані такими методами.

Аналіз предметної області – дослідження сучасних методів аналізу та редагування зображень, що здійснювалося у першому розділі кваліфікаційної роботи, дозволило визначити ключовий функціонал, який є актуальним та затребуваним. Аналіз існуючих аналогів дозволив провести вивчення переваг та недоліків наявних програмних продуктів зі схожим функціоналом допоміг сформулювати вимоги до унікальних аспектів розроблюваної системи (наприклад, поєднання визначення та інпейнтінгу в одному локальному додатку).

Формалізовані методи синтезу моделей програмного забезпечення – синтез моделей програмного забезпечення виконується шляхом застосування правил та нотацій мови UML для побудови діаграм, що візуалізують структуру та поведінку системи. На основі проаналізованих вимог були побудовані діаграми прецедентів та класів, які слугують формалізованим описом проєктного рішення.

Основна ціль розроблюваної системи – надання користувачеві зручного настільного інструменту для аналізу зображень шляхом автоматичного визначення об’єктів та їхнього редагування за допомогою генеративного інпейнтінгу на основі текстового опису.

Задачі системи, що випливають з поставленої цілі:

– забезпечення можливості завантаження та відображення зображень;

– автоматичної детекції об’єктів на зображенні;

– відображення результатів детекції та забезпечення можливості вибору об’єктів користувачем;

– виконання інтелектуальної заміни вибраного об’єкта за текстовим промптом;

– забезпечення неблокуючої роботи інтерфейсу під час обчислювальних операцій;

– надання можливості збереження відредагованого зображення.

Вимоги до продукту включають функціональні та нефункціональні аспекти.

Функціональні вимоги (надалі будуть іменуватися скорочено – ФВ):

– ФВ.1. Система повинна надавати користувачеві можливість завантажувати зображення у форматах PNG, JPG, JPEG, BMP, WEBP;

– ФВ.2. Система повинна відображати завантажене зображення у головному вікні з можливістю прокрутки, якщо зображення перевищує розміри області відображення;

– ФВ.3. Система повинна надавати функціонал для ініціації процесу автоматичної детекції об’єктів на поточному зображенні;

– ФВ.4. Система повинна відображати список виявлених об’єктів, включаючи їх клас та, за можливості, оцінку впевненості;

– ФВ.5. Система повинна візуально виділяти вибраний користувачем об’єкт зі списку на зображенні (наприклад, обведенням обмежувальним прямокутником);

– ФВ.6. Система повинна надавати поле для введення текстового опису (промпта) для задачі генеративного редагування;

– ФВ.7. Система повинна надавати функціонал для ініціації процесу генеративного інпейнтінгу (заміни) вибраного об’єкта на основі поточного зображення, маски вибраного об’єкта та введеного текстового промпта;

– ФВ.8. Система повинна оновлювати відображуване зображення після успішного завершення процесу інпейнтінгу;

– ФВ.9. Система повинна надавати функціонал для збереження поточного зображення у файл, включаючи підтримку основних форматів зображень (PNG, JPG, etc.);

– ФВ.10. Система повинна надавати функціонал для скидання зображення до його початкового (оригінального) стану після завантаження;

– ФВ.11. Система повинна відображати хід виконання тривалих операцій (детекція, генерація) за допомогою індикатора прогресу;

– ФВ.12. Система повинна відображати інформацію про статус та повідомлення про помилки в рядку стану або спливаючих вікнах.

Нефункціональні вимоги:

– НФВ.1. Час виконання операцій детекції та генерації повинен бути прийнятним для настільного додатку, хоча і може залежати від потужності обладнання користувача (рекомендовано наявність GPU для інпейнтінгу). Інтерфейс користувача повинен залишатися чутливим під час виконання цих операцій;

– НФВ.2. Система повинна коректно обробляти типові сценарії використання. У випадку виникнення помилок (наприклад, під час завантаження моделі, обробки зображення), користувач повинен бути проінформований;

– НФВ.3. Інтерфейс користувача має бути інтуїтивно зрозумілим та простим у використанні для користувачів без спеціальних знань у галузі ШІ;

– НФВ.4. Додаток повинен бути сумісним з основними операційними системами (Windows, Linux), де підтримуються використані бібліотеки. Процес встановлення або запуску має бути максимально простим;

– НФВ.5. Додаток повинен ефективно використовувати системні ресурси, особливо пам’ять (RAM та VRAM), наскільки це можливо при роботі з великими моделями глибокого навчання;

– НФВ.6. Оскільки додаток є локальним настільним інструментом і не працює з конфіденційними даними чи мережевим доступом, специфічні вимоги до безпеки даних або розмежування прав доступу відсутні.

Була побудована діаграма прецедентів (рис. 2.1), з метою наочного представлення основного функціоналу системи «AI Image Editor System» з точки зору взаємодії користувача з нею.

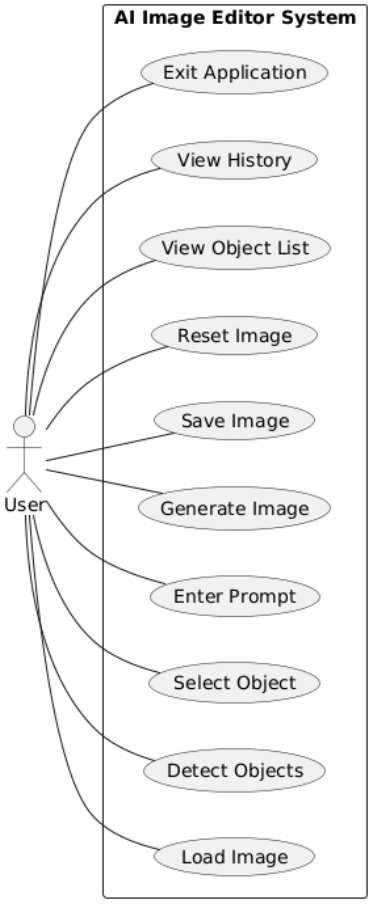


Рисунок 2.1 – Діаграма прецедентів

Діаграма відображає єдиного актора – користувача (User), який взаємодіє з системою «AI Image Editor System». Визначено основні прецеденти, що відповідають функціональним вимогам: завантаження, визначення, вибір об’єкта, введення промпта, генерація, збереження, скидання, перегляд списків. Це демонструє основний функціонал системи з точки зору користувацьких сценаріїв використання.

Також була побудована діаграма класів (рис. 2.2) для моделювання структури програмного забезпечення на рівні класів і їхніх взаємозв’язків.

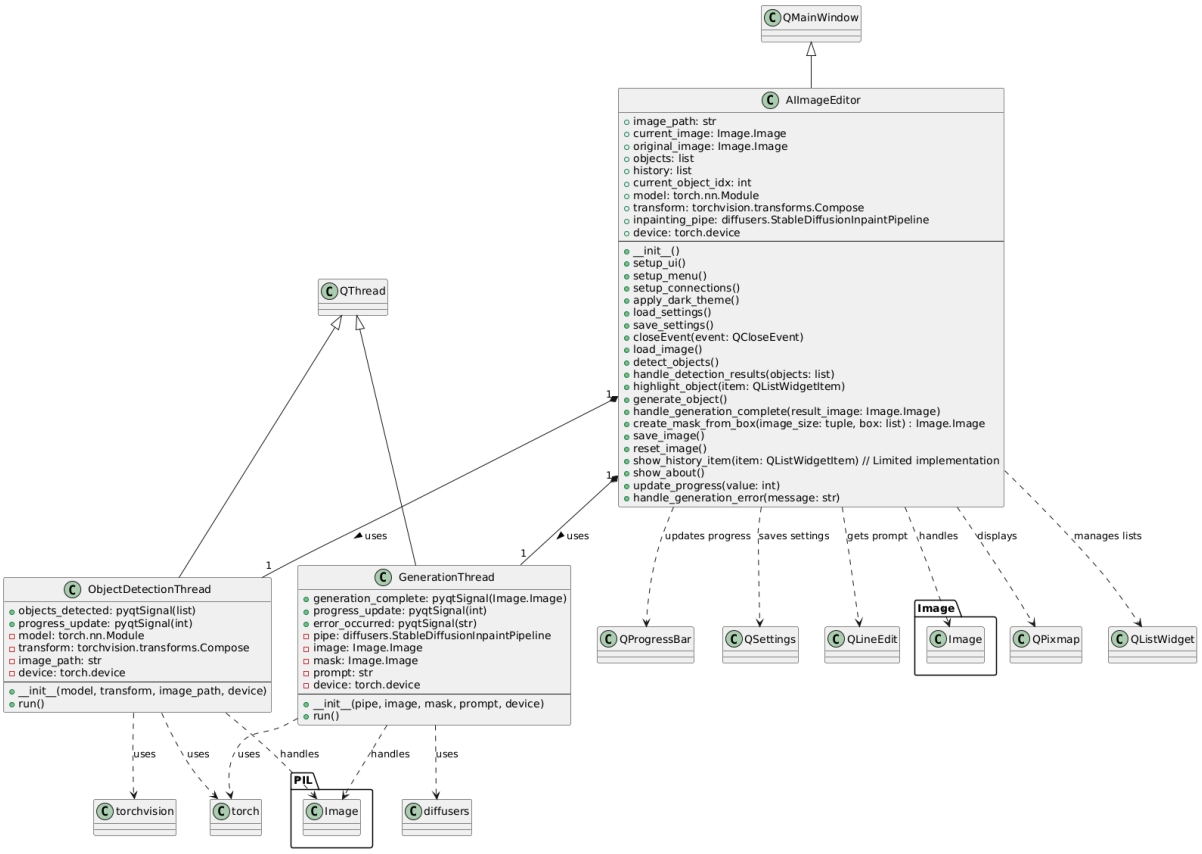


Рисунок 2.2 – Діаграма класів

Діаграма представляє основні класи програмного забезпечення. AIImageEditor є головним класом, що відповідає за інтерфейс та загальне керування. Він містить посилання на поточне та оригінальне зображення, списки об’єктів та історії, а також посилання на моделі ШІ (хоча самі моделі тут представлені як поля, а не окремі класи моделей для спрощення).

ObjectDetectionThread та GenerationThread є окремими класами-потоками, які виконують ресурсоємні операції. Зв’язки типу агрегації (*--) від AIImageEditor до потоків вказують, що головне вікно створює та керує цими потоками. Зв’язки залежності (..>) показують, які зовнішні бібліотеки використовуються певними класами. Діаграма відображає модульну структуру системи та розподіл відповідальності.

Аналіз отриманого результату моделювання, аргументоване пояснення отриманого рішення.

Застосування UML-моделювання дозволило візуалізувати структуру системи та взаємозв’язки між її компонентами. Діаграма прецедентів чітко визначає функціональні межі додатку з точки зору користувача, охоплюючи всі основні операції. Діаграма класів деталізує внутрішню структуру, показуючи ключові класи, які реалізують ці функції, та їх залежності від зовнішніх бібліотек.

Отримане рішення базується на архітектурі з явним розділенням між основним потоком інтерфейсу користувача та фоновими потоками для обчислювально складних задач. Клас AIImageEditor виступає як контролер, що обробляє дії користувача та координує роботу інших компонентів. Окремі класи-потоки (ObjectDetectionThread, GenerationThread) інкапсулюють логіку взаємодії з моделями ШІ, виконуючи її паралельно до роботи GUI, що є критично важливим для забезпечення чутливості інтерфейсу користувача під час виконання операцій, які можуть тривати від кількох секунд до кількох хвилин (особливо інпейнтінг на CPU або слабкому GPU). Використання сигналів та слотів PyQt є ефективним механізмом для безпечної передачі результатів виконання з фонових потоків у головний потік GUI для оновлення інтерфейсу.

Відповідно до поставлених функціональних вимог, система забезпечує наступні функціональні характеристики:

– початкове введення інформації здійснюється шляхом вибору файлу зображення через стандартний діалог файлової системи;

– зміна існуючих даних реалізується через функціонал інпейнтінгу, де вибрана область зображення замінюється згенерованим вмістом;

– непрямий «пошук» реалізовано у вигляді автоматичного визначення об’єктів, що ідентифікує та локалізує об’єкти певних класів на зображенні;

– здійснюється базовий контроль типу файлу при завантаженні зображення. Введення промпта користувачем є вільним текстом, контроль його коректності чи змісту моделями ШІ не здійснюється на рівні додатку;

– виявлені об’єкти виводяться у вигляді списку. Результат інпейнтінгу виводиться шляхом оновлення візуального представлення зображення. Статус та помилки виводяться у рядку стану та спливаючих вікнах.

Проєктування користувацького інтерфейсу (UI) та досвіду взаємодії (UX) було зосереджено на створенні інтуїтивно зрозумілого та функціонального простору для роботи із зображеннями та функціями ШІ. Основна схема інтерфейсу реалізована за допомогою віджета QSplitter, який розділяє головне вікно на дві основні панелі.

Ліва панель призначена для відображення зображення та основних елементів керування (рис. 2.3).

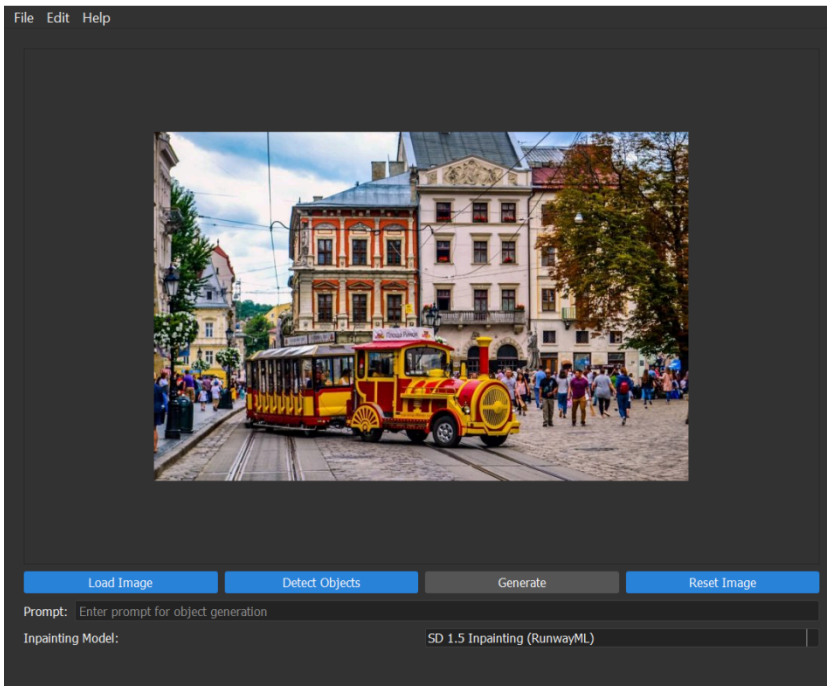


Рисунок 2.3 – Ліва панель

Центральне місце займає QScrollArea з QLabel для відображення зображення, що дозволяє працювати із зображеннями різного розміру. Під областю зображення розташовані кнопки для основних дій (Load Image, Detect Objects, Generate, Reset Image), поле введення текстового промпта (QLineEdit) та елемент вибору моделі (QComboBox – з перспективою розширення). Також тут розташований QProgressBar для індикації виконання тривалих операцій.

Права панель призначена для відображення списків результатів аналізу та історії дій (рис. 2.4).

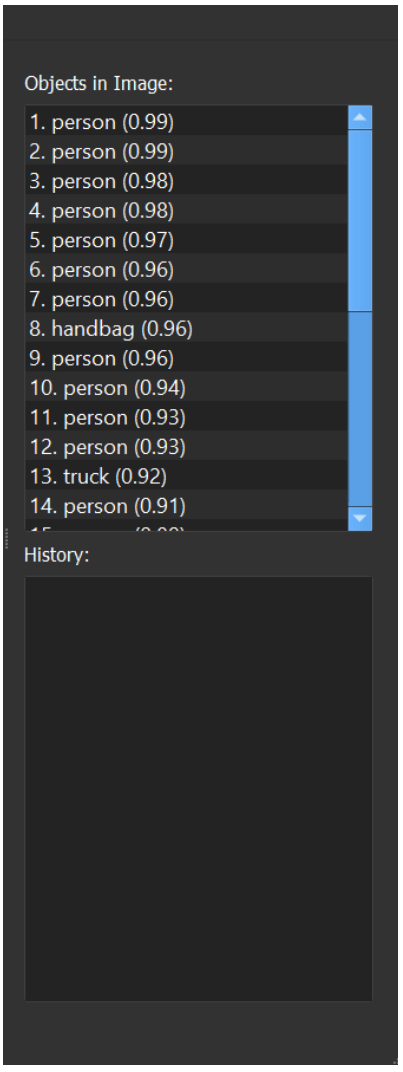


Рисунок 2.4 – Права панель

Вона містить QListWidget для відображення виявлених об’єктів та QListWidget для відображення історії операцій.

Таке компонування дозволяє одночасно бачити зображення, керувати процесами та переглядати результати аналізу та історію. Вибір темної теми оформлення (apply_dark_theme) спрямований на зменшення навантаження на очі користувача та створення більш сучасного вигляду, що є поширеним у графічних редакторах.

Основний інформаційний потік починається з завантаження зображення користувачем. Далі зображення надходить на аналіз до модуля детекції. Результати аналізу (список об’єктів) повертаються в інтерфейс та відображаються користувачеві. Користувач, виходячи зі своїх інформаційних потреб (знайти конкретний об’єкт, дізнатись, що є на зображенні), взаємодіє зі списком об’єктів. При виборі об’єкта, інформація про його місцезнаходження використовується для візуального виділення на зображенні та формування маски. Далі, користувач формує текстовий запит (промпт), виходячи зі своєї інформаційної потреби щодо бажаного вмісту заміненої області. Зображення, маска та промпт передаються модулю генерації. Результат генерації (нове зображення) повертається користувачеві шляхом оновлення відображення. Вся послідовність ключових дій (завантаження, генерація) фіксується в історії для інформаційного забезпечення користувача про виконані модифікації.

Інформаційні потреби користувачів, які задовольняє додаток, включають:

– ідентифікація об’єктів на зображенні;

– локалізація об’єктів на зображенні;

– можливість вибіркової модифікації частин зображення на основі їх вмісту та бажаного результату;

– візуальний контроль над процесами аналізу та редагування;

– відстеження послідовності внесених змін (базова історія).

Таким чином, проєктування системи базувалося на аналізі потреб користувачів та можливостей обраних методів ШІ, що відображено у специфікації вимог, моделях UML та схемі користувацького інтерфейсу.

## 2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Вибір відповідних засобів розробки є критично важливим етапом, що безпосередньо впливає на ефективність, масштабованість та якість кінцевого програмного продукту. На цьому етапі було проаналізовано різні мови програмування, фреймворки для розробки графічного інтерфейсу та спеціалізовані бібліотеки для роботи зі штучним інтелектом, з метою обґрунтованого вибору стеку технологій, що найкращим чином відповідає потребам та цілям.

Вибір мови програмування визначає не лише синтаксис, але й усю екосистему розробки, доступність спеціалізованих бібліотек та загальну зручність написання та підтримки коду. Для проєктів з інтеграцією штучного інтелекту найпопулярнішими варіантами є Python, C++ та Java. C++ відомий своєю високою продуктивністю, що може бути критично важливою для обчислювально інтенсивних завдань, проте розробка на C++ є значно складнішою та тривалішою. Java, будучи кросплатформною мовою з великою екосистемою, все ж таки менш поширена для сучасної розробки ШІ порівняно з Python [13].

Натомість Python став де-факто стандартом у наукових дослідженнях та розробці штучного інтелекту, і його вибір для даного проєкту є найбільш доцільним та стратегічно обґрунтованим. Ця мова славиться своїм винятково простим та читабельним синтаксисом, що дозволяє значно пришвидшити процес розробки прототипів та реалізацію складних алгоритмів. Головною перевагою Python [14] є його неймовірно багата екосистема спеціалізованих бібліотек, таких як PyTorch, TensorFlow, scikit-learn та NumPy, які надають вже готові, високооптимізовані інструменти для роботи з машинним навчанням, обчисленнями та аналізом даних. Ця доступність передових інструментів ШІ без необхідності написання низькорівневого коду дозволяє розробникам зосередитись безпосередньо на логіці та функціоналі додатку. Крім того, Python

має потужні фреймворки для розробки графічних інтерфейсів, що робить його універсальним рішенням для створення повноцінних настільних додатків. Таким чином, для даного проєкту, який передбачає активне використання бібліотек машинного навчання та швидку інтеграцію готових моделей, мова Python є не просто доцільною, а ідеальною, забезпечуючи легкий доступ до передових інструментів ШІ та значно прискорюючи весь процес розробки.

Вибір фреймворку для розробки графічного інтерфейсу безпосередньо впливає на вигляд, відчуття та зручність взаємодії користувача з додатком. Для Python існує кілька популярних варіантів, кожен з яких має свої особливості. Tkinter є стандартним фреймворком Python, який є простим у вивченні, але його можливості обмежені, а зовнішній вигляд часто виглядає застарілим. Kivy добре підходить для розробки мультиплатформових додатків, включаючи мобільні, проте він має власну мову дизайну (Kv) і менш типовий «настільний» вигляд. PyGTK, прив’язаний до бібліотеки GTK+, менш популярний у Python-спільноті порівняно з PyQt.

Для створення повнофункціонального, сучасного та кросплатформенного настільного додатку, який потребує гнучкого компонування та просунутих елементів інтерфейсу, було обрано бібліотеку PyQt5. PyQt5 є однією з найбільш потужних та кросплатформенних бібліотек для розробки GUI у Python, що базується на високо оціненій бібліотеці Qt. Вона надає надзвичайно широкі можливості для створення сучасних, інтуїтивно зрозумілих та гнучких інтерфейсів, які відповідають стандартам операційних систем. Однією з ключових переваг PyQt5 є підтримка візуального дизайнера форм (Qt Designer), що значно прискорює створення макетів інтерфейсу. Крім того, її механізм сигналів і слотів є елегантним та надійним способом обробки подій, забезпечуючи чистий та зрозумілий код. PyQt5 (з ліцензією GPL/Commercial) та PySide (з ліцензією LGPL) є двома варіантами використання Qt з Python, і для даного проєкту саме PyQt5 відповідає всім вимогам та є відмінно документованою.

Ці бібліотеки є ключовими інструментами для реалізації функціоналу аналізу та редагування зображень. Серед провідних бібліотек для ШІ на Python [15] виділяються PyTorch [11] та TensorFlow, які є провідними фреймворками для глибокого навчання, надаючи потужні інструменти для побудови, навчання та використання нейронних мереж з підтримкою GPU-прискорення. Для базової обробки зображень популярними є Pillow [10] (PIL) та OpenCV [9]. Для специфічних завдань комп'ютерного зору та генеративних моделей існують спеціалізовані бібліотеки, що розширюють можливості основних фреймворків.

Для реалізації функціоналу ШІ була обрана екосистема PyTorch через її надзвичайну популярність у дослідницькій спільноті, що забезпечує доступ до найновіших моделей та активної підтримки, а також завдяки її гнучкості та зручності використання. Для ефективної детекції об'єктів використано torchvision – бібліотеку, що входить до екосистеми PyTorch, яка містить популярні моделі комп'ютерного зору, включаючи легко інтегровані попередньо навчені моделі, такі як fasterrcnn_resnet50_fpn. Це дозволяє швидко підготувати зображення до обробки, виконуючи необхідні перетворення, такі як transforms.Compose([transforms.ToTensor()]).

Особливо варто відзначити вибір бібліотеки diffusers від Hugging Face [7] для генеративного інпейнтінгу. Ця бібліотека є надзвичайно потужним інструментом, що надає готові до використання пайплайни для роботи з передовими дифузійними моделями, зокрема StableDiffusionInpaintPipeline. Її інтеграція значно спрощує використання такої складної генеративної моделі, інкапсулюючи всю складність внутрішніх перетворень, виконання моделі та повернення згенерованого зображення, вимагаючи від розробника лише передачі текстового промпта, вхідного зображення та маски.

Для базових операцій з зображеннями, які не залежать від PyTorch, використовується бібліотека Pillow (PIL). Вона є незамінною для відкриття та збереження зображень (Image.open(), image.save()), конвертації форматів (.convert(«RGB»)), зміни розміру (.resize()) та для малювання на зображенні

(наприклад, обмежувальних прямокутників) за допомогою модуля ImageDraw. Pillow виступає необхідним проміжним етапом для ефективного перетворення зображень між файловим форматом, форматом PyTorch тензорів та форматом PyQt (QPixmap, QImage), забезпечуючи безшовну взаємодію між усіма компонентами. Бібліотека OpenCV, хоча й є потужною для комп'ютерного зору, не була критично необхідною для поставлених завдань, оскільки функціонал, наданий Pillow та torchvision/diffusers, є достатнім для обробки зображень у контексті даного проєкту.

На основі обґрунтованого вибору, для розробки додатку буде використовуватись наступний стек технологій: Python 3.x, PyQt5, PyTorch, torchvision, diffusers та Pillow.

На рисунку 2.5 зображений алгоритм визначення об'єктів із застосуванням штучного інтелекту. На схемі покроково показано, як саме система виявляє об'єкти на зображенні.

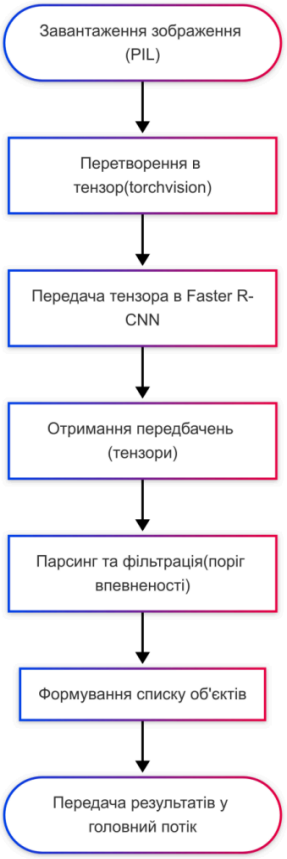


Рисунок 2.5 – Блок-схема алгоритму визначення об'єктів

Алгоритм визначення об’єктів базується на використанні моделі Faster R-CNN. Математична модель алгоритма це глибока згорткова нейронна мережа з архітектурою Faster R-CNN, що включає Feature Pyramid Network (FPN) та Region Proposal Network (RPN).

На рисунку 2.6 зображена блок-схема алгоритму інпейнтінгу – процесу автоматичного заповнення або редагування вибраної області на зображенні за допомогою штучного інтелекту. Схема демонструє, як система створює згенеровану частину картинки на основі вхідних даних.

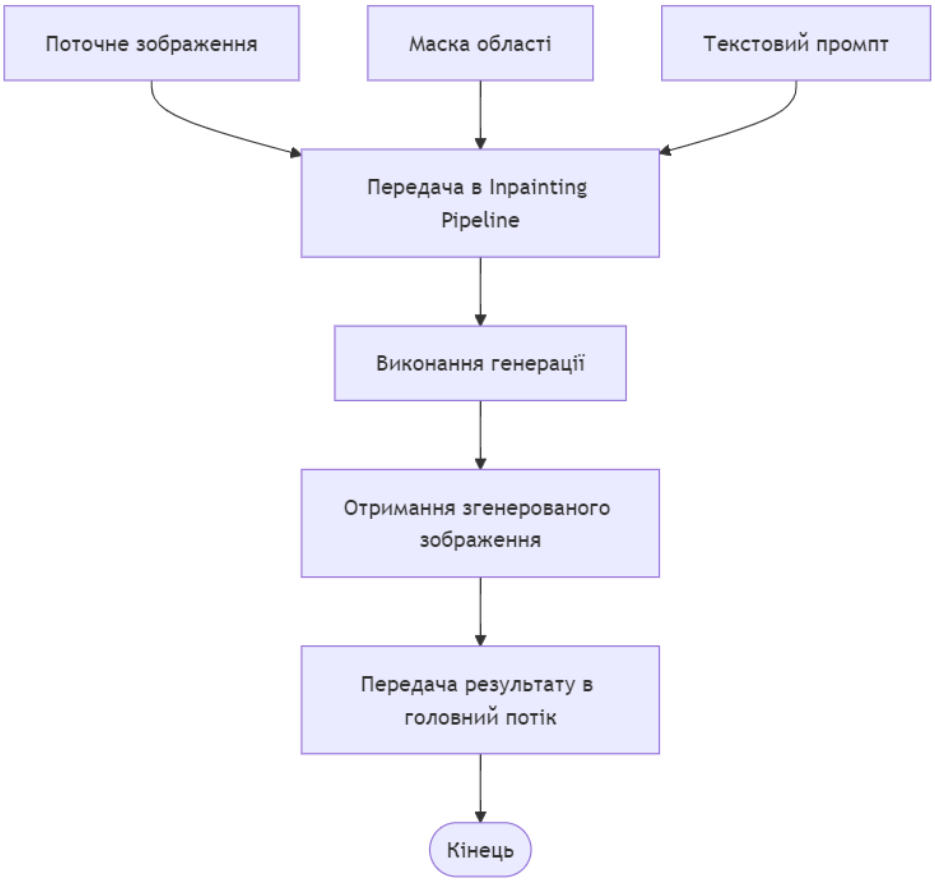


Рисунок 2.6 – Блок-схема алгоритму інпейнтінгу

Алгоритм генеративного інпейнтінгу базується на використанні пайплайна Stable Diffusion Inpainting. Математична модель алгоритму це латентна дифузійна модель Stable Diffusion, адаптована для задачі інпейнтінгу,

що використовує механізми додавання/видалення шуму та керування за текстом і маскою.

Також в додатку реаліхований алгоритм керування інтерфейсом та багатопотоковістю. Він забезпечує взаємодію користувача з алгоритмами ШІ. Використання багатопотоковості за допомогою QThread для виконання тривалих операцій (визначення, генерація) у фоні, не блокуючи головний потік GUI. Комунікація між потоками відбувається за допомогою сигналів та слотів PyQt.

На додаток до діаграм прецедентів та класів, представлених вище, що описують статичну структуру та варіанти використання системи, наведені вище блок-схеми ілюструють динаміку та послідовність кроків у ключових алгоритмах.

Також можна представити спрощену діаграму компонентів, що ілюструє основні модулі системи та їх залежності (рис. 2.7).

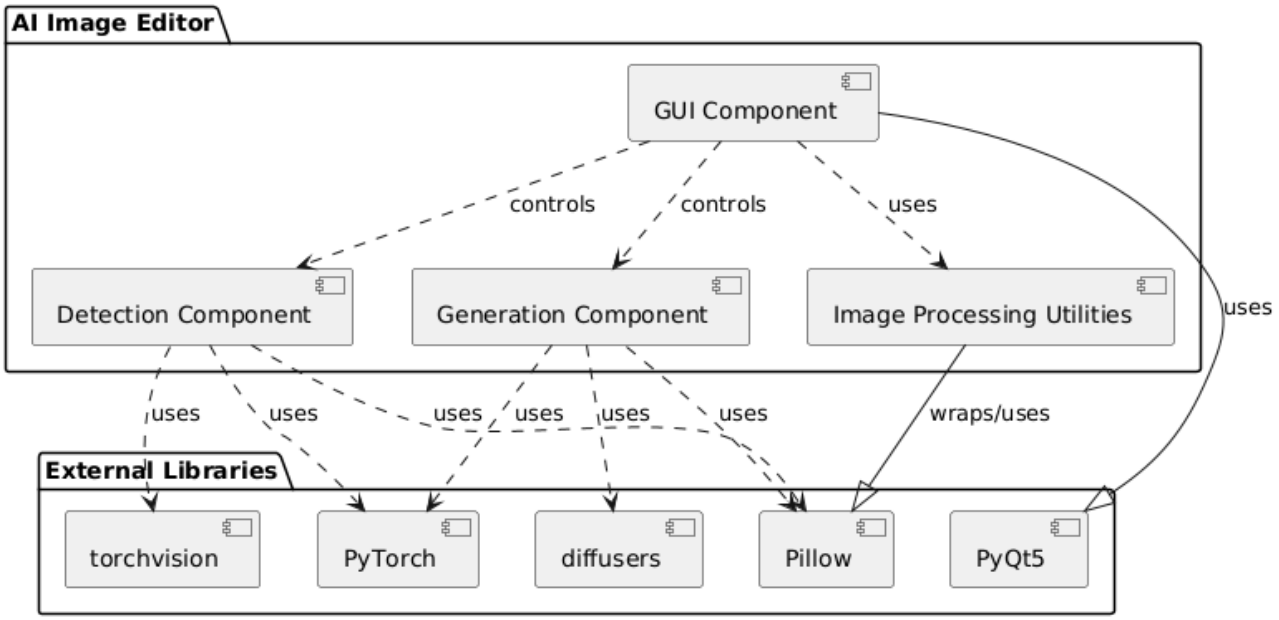


Рисунок 2.7 – Діаграма компонентів

Діаграма компонентів показує, що система складається з внутрішніх компонентів (GUI, Detection, Generation, Image Processing Utilities), які використовують функціонал зовнішніх бібліотек. Компонент GUI залежить від

PyQt5 та взаємодіє з компонентами детекції та генерації. Компоненти визначення та генерації залежать від PyTorch, torchvision/diffusers та Pillow. Компонент Image Processing Utilities є тонкою обгорткою над Pillow для специфічних завдань обробки в рамках додатку [6]. Ця діаграма ілюструє розподіл функціоналу між основними частинами системи та їхні зовнішні залежності.

## Висновки до розділу 2

У другому розділі кваліфікаційної роботи було проведено аналіз, визначення вимог і проєктування програмної системи, що є критично важливим для створення чіткого плану реалізації. На початку обґрунтовано вибір UML як мови моделювання. Діаграми прецедентів та класів допомогли формалізувати функціональні вимоги та структуру компонентів, а діаграма компонентів – показати модульну архітектуру й зовнішні залежності.

На основі аналізу предметної області сформовано специфікацію вимог: функціональні (завантаження, збереження, детекція об’єктів, інпейнтінг, багатопоточність) і нефункціональні (продуктивність, зручність, надійність). Обґрунтовано вибір стеку технологій: Python за гнучкість і бібліотеки ШІ, PyQt5 – для інтерфейсу, PyTorch, torchvision і diffusers – для AI-функцій, Pillow – для обробки зображень.

Також описано роль інструментів і подано блок-схеми основних алгоритмів: завантаження, детекції, інпейнтінгу та багатопотокового керування. Це забезпечує цілісне розуміння логіки роботи додатку.

# РОЗДІЛ 3
# РОЗРОБКА ДОДАТКУ З ШТУЧНИМ ІНТЕЛЕКТОМ ДЛЯ АНАЛІЗУ І РЕДАГУВАННЯ ЗОБРАЖЕННЯ

## 3.1 Практична реалізація об’єкта проєктування

Для успішної розробки додатку для аналізу та редагування зображення потрібно виконати такі етапи:

– розробити графічний інтерфейс користувача для настільного додатку за допомогою бібліотеки PyQt5, який забезпечить зручну взаємодію з користувачем;

– реалізувати функціонал завантаження зображень у різних форматах та їх відображення у вікні додатку з можливістю масштабування та прокрутки;

– інтегрувати попередньо навчену модель глибокого навчання для автоматичної детекції об’єктів на завантаженому зображенні;

– реалізувати механізм відображення результатів детекції об’єктів у вигляді списку та візуального виділення виявлених об’єктів на зображенні (обмежувальними прямокутниками);

– інтегрувати попередньо навчену генеративну модель на основі дифузійних моделей (Stable Diffusion Inpainting) для виконання задачі інпейнтінгу;

– реалізувати функціонал генеративного редагування (інпейнтінгу) вибраного користувачем об’єкта на зображенні на основі введеного текстового опису (промпта);

– забезпечити виконання ресурсоємних операцій (визначення об’єктів, генерація зображень) у фонових потоках для запобігання блокуванню графічного інтерфейсу користувача;

– реалізувати функціонал збереження відредагованого зображення у файл.

Практична реалізація програмного додатку «AI Image Editor» здійснювалася з використанням обраного стеку технологій: мови програмування Python 3.x та бібліотек PyQt5, PyTorch, torchvision, diffusers та

Pillow. Процес розробки включав послідовне створення компонентів користувацького інтерфейсу, інтеграцію моделей штучного інтелекту, реалізацію логіки взаємодії та обробки даних, а також налаштування багатопоточності для забезпечення ефективної роботи додатку. Розробка велася у стандартному середовищі розробки Python (А також VS Code [8] та Thonny [12]), що надає зручні інструменти для написання, налагодження та виконання коду.

Застосування об’єктно-орієнтованого підходу в мові Python дозволило структурувати код у вигляді класів, що відповідають за певні аспекти функціоналу. Головний клас додатку AIImageEditor успадковується від QMainWindow PyQt5 та інкапсулює логіку управління інтерфейсом та координації роботи інших компонентів. Окремі потоки для ресурсоємних операцій реалізовані у класах ObjectDetectionThread та GenerationThread, що успадковуються від QThread.

Розробка графічного інтерфейсу здійснювалася за допомогою бібліотеки PyQt5. Структура інтерфейсу, описана у другому розділі, була реалізована програмно у методі setup_ui() класу AIImageEditor. Використовувалися менеджери компонування (QVBoxLayout, QHBoxLayout) та роздільник QSplitter для гнучкого розташування віджетів.

Наприклад лістинг 3.1 ілюструє створення головного вікна, встановлення компонування та роздільника. Відображення зображення реалізовано за допомогою QLabel всередині QScrollArea, що дозволяє коректно працювати із зображеннями, більшими за розмір вікна.

Лістинг 3.1 – Створення елементів інтерфейсу

```
# Фрагмент з методу AIImageEditor.setup_ui()
self.central_widget = QWidget()
self.setCentralWidget(self.central_widget)
main_layout = QVBoxLayout()
self.central_widget.setLayout(main_layout)
self.splitter = QSplitter(Qt.Horizontal)
main_layout.addWidget(self.splitter)
# Ліва панель для зображення та керування
```

```
self.left_panel = QWidget()
self.left_layout = QVBoxLayout(self.left_panel)
# ... додавання віджетів до left_layout ...
# Права панель для списків
self.right_panel = QWidget()
self.right_layout = QVBoxLayout(self.right_panel)
# ... додавання віджетів до right_layout ...
self.splitter.addWidget(self.left_panel)
self.splitter.addWidget(self.right_panel)
self.splitter.setSizes([700, 300]) # Встановлення початкових розмірів
панелей
```

кінець лістингу 3.1

Взаємодія користувача з елементами інтерфейсу реалізована за допомогою механізму сигналів та слотів PyQt. У методі setup_connections() здійснюється з'єднання сигналів від віджетів (наприклад, clicked для QPushButton, itemClicked для QListWidget) з відповідними методами (слотами) класу AIImageEditor (ліст. 3.2).

Лістинг 3.2 – Створення елементів інтерфейсу для керування додатком

```
# Фрагмент з методу AIImageEditor.setup_connections()

self.load_button.clicked.connect(self.load_image)
self.detect_button.clicked.connect(self.detect_objects)
self.generate_button.clicked.connect(self.generate_object)
self.object_list.itemClicked.connect(self.highlight_object)

# ... інші з'єднання ...
```

кінець лістингу 3.2

Завантаження зображення виконується у методі load_image() (ліст. 3.3). Використовується QFileDialog.getOpenFileName() для вибору файлу користувачем. Далі бібліотека Pillow (PIL.Image) застосовується для відкриття файлу та конвертації його у формат RGB, що є стандартним для подальшої обробки та моделей ШІ. Зображення зберігається як оригінальне (self.original_image) та поточне (self.current_image). Для відображення в інтерфейсі, зображення з файлу конвертується у формат QPixmap та встановлюється у self.image_label.

Лістинг 3.3 – Метод load_image для завантаження фото

```
# Фрагмент з методу AIImageEditor.load_image()
file_path, _ = QFileDialog.getOpenFileName(
    self, "Select Image", "", "Images (*.png *.jpg *.jpeg *.bmp *.webp)"
)
if file_path:
    try:
        self.image_path = file_path
        self.original_image = Image.open(file_path).convert("RGB")
        self.current_image = file_path # Зберігаємо шлях або об'єкт PIL

        pixmap = QPixmap(file_path)
        self.image_label.setPixmap(pixmap)
        self.image_label.setScaledContents(False)
        # ... увімкнення кнопок, очищення списків ...
    except Exception as e:
        QMessageBox.critical(self, "Error", f"Failed to load image:
{str(e)}")
```

кінець лістингу 3.3

Модуль визначення реалізовано у класі ObjectDetectionThread. У конструкторі головного вікна (AIImageEditor.__init__) завантажується попередньо навчена модель Faster R-CNN з torchvision.models та визначається пристрій для обчислень (CPU або GPU). Перетворення зображення для моделі також ініціалізується за допомогою torchvision.transforms (ліст. 3.4).

Лістинг 3.4 – Процес вибору пристрою для обчислень

```
# Фрагмент з методу AIImageEditor.__init__()
self.device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
print(f"Using device: {self.device}")

try:
    self.model = models.detection.fasterrcnn_resnet50_fpn(
        weights=models.detection.FasterRCNN_ResNet50_FPN_Weights.DEFAULT
    )
    self.model.eval()
    self.model.to(self.device)
    self.transform = transforms.Compose([transforms.ToTensor()])
except Exception as e:
    QMessageBox.critical(self, "Error", f"Failed to load object detection
model: {str(e)}")
    sys.exit(1)
```

кінець лістингу 3.4

Безпосереднє виконання визначення відбувається у методі run() потоку ObjectDetectionThread. Зображення завантажується, перетворюється у тензор, переміщується на відповідний пристрій, і передається моделі для інференсу. Результати фільтруються за порогом впевненості (0.5) та відправляються назад у головний потік за допомогою сигналу objects_detected (ліст. 3.5).

Лістинг 3.5 – Процес передачі фото для обробки алгоритмом

```
# Фрагмент з методу ObjectDetectionThread.run()
try:
    image = Image.open(self.image_path).convert("RGB")
    # ... resize image if too large ...

    img_tensor = self.transform(image).unsqueeze(0).to(self.device)

    with torch.no_grad():
        predictions = self.model(img_tensor)[0]
    objects = []
    for label, score, box in zip(predictions['labels'],
predictions['scores'], predictions['boxes']):
        if score > 0.5:
            objects.append((label.item(), score.item(),
box.cpu().numpy()))
    self.objects_detected.emit(objects)
except Exception as e:
    print(f"Detection error: {str(e)}")
    # ... handle error ...
```

кінець лістингу 3.5

Результати детекції обробляються у головному потоці (handle_detection_results()), оновлюючи список об'єктів у QListWidget.

Функціонал інпейнтінгу реалізовано у класі GenerationThread з використанням бібліотеки diffusers. У методі generate_object() головного вікна, перед запуском потоку, завантажується пайплайн Stable Diffusion Inpainting, якщо він ще не завантажений (ліст. 3.6).

Лістинг 3.6 – Перевірка наявності моделі

```
# Фрагмент з методу AIImageEditor.generate_object()
if not DIFFUSION_AVAILABLE:
    QMessageBox.warning(self, "Warning", "Diffusion models not available.
```

```
Please install 'diffusers' library.")
    return
if self.inpainting_pipe is None:
    try:
        self.status_bar.showMessage("Loading inpainting model...")
        QApplication.processEvents() # Оновлення інтерфейсу
        self.inpainting_pipe =
StableDiffusionInpaintPipeline.from_pretrained(
            "runwayml/stable-diffusion-inpainting",
torch_dtype=torch.float16 # Використання float16 для економії пам'яті GPU
        ).to(self.device)
        self.status_bar.showMessage("Inpainting model loaded.")
    except Exception as e:
        QMessageBox.critical(self, "Error", f"Failed to load inpainting
model: {str(e)}")
        self.status_bar.showMessage("Ready")
        return
```

кінець лістингу 3.6

Безпосереднє виконання генерації відбувається у методі run() потоку GenerationThread. Вхідне зображення та маска (створена на основі обмежувального прямокутника вибраного об'єкта за допомогою допоміжної функції create_mask_from_box з використанням Pillow) разом із текстовим промптом передаються пайплайну для генерації (ліст. 3.7).

Лістинг 3.7 – Процес генерації зображення

```
# Фрагмент з методу GenerationThread.run()
try:
    # ... resize image and mask if needed ...
    # Generate the new image
    result = self.pipe(
        prompt=self.prompt,
        image=self.image,
        mask_image=self.mask,
        guidance_scale=7.5,
        num_inference_steps=30 # Кількість кроків інференсу
    ).images[0]
    self.generation_complete.emit(result)
except Exception as e:
    self.error_occurred.emit(f"Generation error: {str(e)}")
    print(f"Generation error: {str(e)}")
```

кінець лістингу 3.7

Після завершення генерації, згенероване зображення (у форматі PIL Image) передається назад у головний потік за допомогою сигналу generation_complete та оновлює поточне зображення додатку у методі handle_generation_complete().

Для виконання тривалих операцій визначення та генерації без блокування головного потоку інтерфейсу користувача, були використані окремі потоки ObjectDetectionThread та GenerationThread (ліст. 3.8). Ці класи успадковуються від QThread та виконують основну обчислювальну логіку у своєму методі run(). Взаємодія з головним потоком GUI здійснюється виключно через механізм сигналів та слотів PyQt, що є безпечним способом міжпотокової взаємодії в Qt. Використання pyqtSignal дозволяє передавати результати або інформацію про прогрес з фонового потоку у головний.

Лістинг 3.8 – Процес забезпечення багатопоточності

```
# Фрагмент сигналів у класах потоків
class ObjectDetectionThread(QThread):
    objects_detected = pyqtSignal(list)
    progress_update = pyqtSignal(int)
class GenerationThread(QThread):
    generation_complete = pyqtSignal(Image.Image)
    progress_update = pyqtSignal(int)
    error_occurred = pyqtSignal(str)
```

кінець лістингу 3.8

Додатково реалізовано функціонал збереження зображення (save_image()), скидання до оригіналу (reset_image()), базове ведення історії дій у списку (history_list). Обробка потенційних помилок під час завантаження моделей, зображень або виконання ШІ-операцій здійснюється за допомогою блоків try...except, а повідомлення користувачеві виводяться через QMessageBox або QStatusBar.

## 3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Забезпечення стабільної та коректної роботи програмного забезпечення є фундаментальним етапом у життєвому циклі розробки. У процесі створення додатку «AI Image Editor» було застосовано комплексний підхід до тестування та налагодження, що дозволило не лише виявити, а й своєчасно та ефективно усунути потенційні недоліки, значно підвищивши загальну надійність, продуктивність та користувацьку зручність системи.

Під час розробки додатку застосовувалися різноманітні методи тестування, які дозволили всебічно оцінити функціональність та стабільність системи. Зокрема, хоча формалізовані юніт-тести з використанням спеціальних фреймворків, таких як unittest або pytest, не були основним фокусом, тестування окремих невеликих функцій та методів проводилося ізольовано, що дозволило перевірити їхню коректну роботу на типових та граничних вхідних даних. Наприклад, критично важливою була перевірка допоміжної функції create_mask_from_box, яка відповідає за генерацію маски на основі координат виділеної області. Її коректність була ретельно протестована на прямокутниках різного розміру та розташування, включаючи випадки, коли прямокутник виходив за межі зображення, що гарантувало абсолютно точне формування маски для інпейнтінгу.

Паралельно активно проводилося інтеграційне тестування, що полягало у системній перевірці взаємодії між різними компонентами системи, яка є критично важливою для функціонування комплексного додатку. Цей процес включав ретельну перевірку коректності передачі зображення та текстового промпта від графічного інтерфейсу користувача до фонових потоків, що виконують обчислення на основі штучного інтелекту. Послідовно тестувалося, чи коректно ШІ-потоки обробляють отримані дані та чи повертають результати, такі як нове зображення після інпейнтінгу, назад до головного потоку для відображення в інтерфейсі. Особлива увага приділялася взаємодії між вибором

об’єкта у списку виявлених об’єктів та його адекватним, візуально коректним виділенням безпосередньо на зображенні, що забезпечує інтуїтивну та зручну взаємодію з користувачем.

Окрім цього, здійснювалося функціональне тестування, під час якого було проведено детальний перегляд виконання кожного функціонального прецедента, які були раніше детально описані у другому розділі. Ми системно тестували повні сценарії роботи: від успішного завантаження зображення різних форматів до послідовного виконання детекції об’єктів, вибору одного або кількох об’єктів, введення текстового промпта для інпейнтінгу, запуску процесу генерації та подальшого збереження відредагованого зображення. Додатково перевірялися сценарії скидання зображення до початкового стану та коректний вихід з програми, що дозволило переконатися у повній відповідності поведінки системи визначеним функціональним вимогам. Тестування інтерфейсу користувача (GUI Testing) стало ще одним важливим аспектом, що включав ретельну оцінку коректності відображення всіх елементів інтерфейсу, їхнього розташування та динамічної адаптації при зміні розміру вікна. Перевірялася швидкість та адекватність реакції кнопок та інших віджетів на дії користувача, працездатність області прокрутки зображення для великих файлів, а також візуальна індикація прогресу виконання тривалих операцій, що є життєво важливим для комфорту користувача. Не менш важливим було тестування продуктивності, під час якого вимірювався час виконання найбільш ресурсоємних операцій, таких як детекція об’єктів та генерація зображень, на різних тестових зображеннях та на різному апаратному забезпеченні, включаючи порівняння роботи на CPU та GPU. Ці тести підтвердили значне, багаторазове прискорення роботи системи при використанні графічного процесора.

Налагодження програми здійснювалося з використанням стандартних, проте надзвичайно ефективних, інструментів розробника. Зокрема, активно використовувалася функція print() для оперативного виведення значень змінних та поточного стану програми безпосередньо у консоль, що дозволяло швидко

відстежувати хід виконання коду та виявляти місця, де дані або логіка відхилялися від очікуваних. Для системнішого відстеження та аналізу подій застосовувався стандартний модуль logging, що дозволяв записувати детальну інформацію про виконання програми та потенційні помилки у файл або консоль, що виявилося особливо зручним для відстеження послідовності подій, особливо при виникненні проблем, які було важко відтворити у ручному режимі. Інтегрований налагоджувач, доступний у середовищі розробки (наприклад, PyCharm), був незамінним для глибокого аналізу.

Встановлення точок зупину, покрокове виконання коду та можливість перегляду значень змінних у реальному часі дозволяли точно виявляти та аналізувати причини складних помилок у логіці програми, особливо при роботі з тензорами та багатопотоковістю. Для забезпечення високої відмовостійкості програми широко використовувалися блоки try...except, що дозволило коректно перехоплювати та обробляти винятки під час виконання потенційно проблемних операцій, таких як завантаження неіснуючих файлів, звернення до відсутніх моделей або обробка некоректного формату зображення. Це запобігає аварійному завершенню роботи програми та дозволяє інформувати користувача про проблему у зручній та зрозумілій формі, наприклад, через діалогові вікна QMessageBox.

Для коректної та ефективної роботи розробленого програмного додатку «AI Image Editor» критично важливим є дотримання певних мінімальних та рекомендованих системних вимог до апаратного та програмного забезпечення комп’ютера. Мінімальні системні вимоги включають операційну систему Windows 10/11, Ubuntu 20.04+ або іншу, сумісну з Python 3.x та PyQt5. Достатнім процесором буде будь-який сучасний багатоядерний процесор, наприклад, Intel Core i3 або AMD Ryzen 3, а мінімальний об’єм оперативної пам’яті (RAM) становить 8 ГБ. Будь-яка відеокарта, сумісна з операційною системою, є прийнятною, хоча слід зазначити, що GPU-прискорення для функцій штучного інтелекту на цьому рівні може бути відсутнє або істотно обмежене, що призведе до значно довшого часу обробки. Для встановлення

Python, бібліотек та збереження файлів моделей потрібно мінімум 5 ГБ вільного місця на диску. Програмне забезпечення повинно включати встановлений Python 3.x та бібліотеки PyQt5, PyTorch, torchvision, diffusers, Pillow.

Для досягнення оптимальної продуктивності та комфортної роботи з додатком рекомендовані системні вимоги передбачають використання операційної системи Windows 10/11 (64-bit) або Ubuntu 20.04+ (64-bit). Процесор бажано мати сучасний багатоядерний (Intel Core i5/i7/i9, AMD Ryzen 5/7/9 або аналогічний) для значного прискорення обчислень на CPU та загальної швидкодії системи. Рекомендований об’єм оперативної пам’яті (RAM) становить 16 ГБ або більше. Особливо важливою є наявність графічного процесора NVIDIA з підтримкою CUDA та мінімум 8 ГБ відеопам’яті (VRAM); настійно рекомендовано 12 ГБ VRAM або більше для комфортної та швидкої роботи з моделлю Stable Diffusion та зображеннями вищої роздільної здатності, оскільки наявність сумісного GPU значно, в рази або навіть десятки разів, прискорює процес детекції та генерації зображень. Також потрібно 10 ГБ або більше вільного місця на диску. Програмне забезпечення має включати встановлений Python 3.x (рекомендовано версії 3.8-3.10) та актуальні версії бібліотек PyQt5, PyTorch (обов’язково з підтримкою CUDA), torchvision, diffusers, Pillow.

Процес інтенсивної розробки та тестування неминуче виявляв низку потенційних недоліків та вузьких місць, які були системно усунені, що дозволило значно покращити стабільність та користувацький досвід. Однією з найбільш критичних проблем на початкових етапах була проблема блокування інтерфейсу користувача. Перші імплементації операцій детекції об’єктів та генерації зображень, що виконувалися у головному потоці, призводили до повного «зависання» інтерфейсу на час їх виконання, роблячи додаток абсолютно невідзивним. Цей недолік було ефективно усунено шляхом архітектурного рішення: усі ресурсоємні обчислення були перенесені у окремі фонові потоки (QThread). Взаємодію з графічним інтерфейсом було реалізовано виключно через безпечний механізм сигналів та слотів PyQt, що не лише

забезпечило неблокуючу роботу інтерфейсу, але й дозволило інтегрувати індикацію прогресу виконання операцій, наприклад, через оновлення QProgressBar.

Ще одним викликом стала проблема помилок при завантаженні зображень різних форматів. На початку розробки, використання лише вбудованого функціоналу PyQt для завантаження зображень обмежувало підтримувані формати, що призводило до помилок при спробі відкриття зображень у деяких поширених форматах. Цю проблему було вирішено шляхом інтеграції потужної бібліотеки Pillow (PIL), яка не лише забезпечила сумісність з значно ширшим спектром графічних форматів, але й спростила подальшу попередню обробку зображень перед передачею їх ШІ-моделям.

Особливо актуальною виявилася проблема низької продуктивності ШІ-моделей на CPU. Хоча додаток розроблявся з підтримкою роботи на центральному процесорі, час виконання інференсу, особливо для генерації зображень за допомогою Stable Diffusion, виявився дуже тривалим і неприйнятним для комфортного використання. Хоча це обмеження самого апаратного забезпечення, а не програми, в додатку було реалізовано автоматичне визначення наявності GPU з підтримкою CUDA та його пріоритетне використання для всіх ресурсоємних обчислень. Крім того, для подальшої оптимізації на GPU використано torch_dtype=torch.float16 при завантаженні моделі генерації, що дозволило суттєво зменшити споживання відеопам’яті (VRAM) та прискорити обчислення. У системних вимогах тепер чітко зазначено рекомендоване обладнання для комфортної та швидкої роботи.

І нарешті, була вирішена проблема помилок при некоректних вхідних даних. Під час тестування були виявлені випадки аварійного завершення програми при спробах завантажити неіснуючі файли, файли некоректного формату або при проблемах із завантаженням самих моделей ШІ. Для підвищення стабільності та надійності було додано комплексний блок обробки винятків (try...except) при всіх потенційно проблемних операціях файлового вводу/виводу та зверненні до моделей. Це дозволяє програмі коректно реагувати

на такі ситуації, виводячи відповідні повідомлення користувачеві через QMessageBox замість непередбачуваного аварійного завершення.

Крім описаних вище аспектів тестування та налагодження, варто розглянути додаткові особливості реалізації та вирішені інженерні виклики. Зокрема, інтеграція моделей глибокого навчання з бібліотек torchvision та diffusers мала свої технічні нюанси. Це вимагало глибокого розуміння очікуваного формату вхідних даних моделей – тензорів певного розміру та типу – а також формату їхніх вихідних даних для коректного парсингу результатів. Наприклад, для Faster R-CNN потрібно було обробляти словники з тензорами, що містили boxes, labels та scores. Важливим аспектом стало керування пам’яттю, особливо VRAM на GPU, що вимагало уважного вибору типу даних (float16 для моделі генерації) та своєчасного звільнення ресурсів, забезпечуючи стабільну роботу навіть на відеокартах з обмеженим об’ємом пам’яті.

Іншим ключовим викликом стала реалізація безпечної міжпотокової взаємодії у PyQt. Використання багатопоточності у додатках з графічним інтерфейсом вимагає надзвичайної обережності, оскільки прямий доступ з фонового потоку до елементів інтерфейсу головного потоку є небезпечним і може призвести до аварійних завершень. У розробленому додатку ця фундаментальна проблема була вирішена виключно через надійний та безпечний механізм сигналів та слотів PyQt. Фоновим потокам дозволено лише «випромінювати» сигнали, що містять результати обчислень або інформацію про поточний статус, тоді як головний потік «приймає» ці сигнали у відповідних слотах для безпечного оновлення інтерфейсу, таких як оновлення QProgressBar, додавання елементів у QListWidget або оновлення відображуваного зображення.

Нарешті, було реалізовано ефективну обробку зображень різної роздільної здатності та масштабування. Відомо, що моделі ШІ часто мають фіксовані або обмежені вимоги до розміру вхідного зображення, наприклад, 512x512 пікселів для Stable Diffusion Inpainting. Розроблений додаток враховує це, автоматично виконуючи інтелектуальне масштабування вхідних зображень до відповідних

розмірів перед передачею їх моделям, зберігаючи при цьому оригінальні пропорції, щоб уникнути спотворень. При відображенні зображень в інтерфейсі використовується QScrollArea та налаштування setScaledContents(False) для QLabel, що дозволяє коректно показувати зображення будь-якого розміру та забезпечує можливість їх зручного перегляду, якщо вони більші за область відображення, зберігаючи при цьому їх вихідну якість та деталізацію.

Таким чином, ретельне тестування та систематичне налагодження були невід'ємною та інтегральною частиною всього процесу розробки, що дозволило не лише виявити та усунути основні недоліки, але й значно підвищити надійність та зручність використання програмного продукту. Визначені детальні системні вимоги для роботи додатку враховують специфіку використаних ШІ-моделей та апаратного прискорення. Додатково було розкрито та успішно вирішено ключові інженерні виклики, пов'язані з інтеграцією моделей глибокого навчання, реалізацією безпечної багатопотокової взаємодії та ефективною обробкою зображень різної роздільної здатності, що підкреслює практичну цінність та глибину реалізації даного проєкту.

### 3.3 Розробка інсталяційного пакета

Для коректної та ефективної роботи розробленого програмного додатку «AI Image Editor» критично важливим є дотримання певних мінімальних та рекомендованих системних вимог до апаратного та програмного забезпечення комп'ютера. Мінімальні вимоги включають наявність сучасної багатоядерної операційної системи (Windows 10/11, Ubuntu 20.04+), процесора рівня Intel Core i3 / AMD Ryzen 3,8 ГБ оперативної пам'яті та 5 ГБ вільного місця на диску. Проте, для оптимальної продуктивності та комфортної роботи з моделлю Stable Diffusion, яка є ресурсоємною, настійно рекомендовано наявність графічного процесора NVIDIA з підтримкою CUDA та мінімум 8 ГБ, а бажано 12 ГБ або більше відеопам'яті (VRAM), що забезпечує значне, в рази або десятки разів, прискорення обчислень.

Розгортання складних програмних продуктів, що включають великі моделі машинного навчання та численні зовнішні Python-залежності, є значним інженерним викликом. Традиційні підходи дистрибуції, такі як ручне встановлення через командний рядок або пакування у єдиний виконуваний файл, мають обмеження, пов’язані з вимогою до технічних знань користувача або неприпустимим збільшенням розміру дистрибутива. Для проєкту «AI Image Editor» було розроблено власний графічний інсталятор, що забезпечує гнучке та зручне розгортання.

Цей інсталятор, реалізований як автономний Python-скрипт, який забезпечує автоматизований процес встановлення. Його архітектура охоплює дії, необхідні для підготовки середовища: від перевірки та встановлення базових залежностей до завантаження та розгортання великих моделей ШІ. На початковому етапі виконання скрипт ініціює функцію install_basic_dependencies(), яка перевіряє наявність та встановлює PyQt5 для графічного інтерфейсу інсталятора та requests для HTTP-запитів. За відсутності цих пакетів, інсталятор автоматично викликає pip для їх інсталяції, гарантуючи коректний запуск GUI.

Графічний інтерфейс інсталятора, розроблений на PyQt5, надає користувачеві ключову інформацію та елементи управління. Вікно AIImageEditorInstaller містить QProgressBar для візуалізації загального прогресу встановлення та завантаження, а також QTextEdit для детального текстового логу поточних операцій. Інтерфейс також дозволяє обрати пристрій для PyTorch (CPU або CUDA) через QComboBox, динамічно адаптуючи команди встановлення PyTorch для сумісності з апаратним забезпеченням.

Керування залежностями та завантаження моделей становить ядро функціональності інсталятора. Він містить список необхідних Python-бібліотек для основного додатку (наприклад, torch, torchvision, diffusers, pillow) та встановлює їх за потреби. Ключовою особливістю є динамічне завантаження великих попередньо навчених моделей ШІ – StableDiffusionInpaintPipeline та FasterRCNN_ResNet50_FPN – безпосередньо з Hugging Face. Це дозволяє

зменшити розмір початкового дистрибутива, оскільки моделі не включаються до нього, а завантажуються одноразово. Моделі зберігаються у вказаній користувачем або стандартній кеш-директорії, а прогрес завантаження відображається у реальному часі.

Для забезпечення неблокуючого користувацького досвіду, всі тривалі операції (встановлення пакетів та завантаження моделей) виконуються в окремих фонових потоках (QThread). Це зберігає чутливість графічного інтерфейсу та дозволяє оновлювати його під час виконання цих завдань. Взаємодія між фоновими потоками та GUI реалізована через безпечний механізм сигналів та слотів PyQt. Інсталятор також включає механізми обробки винятків для реагування на проблеми, такі як відсутність мережевого з’єднання або помилки встановлення пакетів, відображаючи інформативні повідомлення через QMessageBox та виводячи деталі у лог.

Процес встановлення додатку «AI Image Editor» відбувається у кілька послідовних кроків, значна частина яких автоматизується спеціально розробленим інсталятором для зручності користувача.

Перш за все, переконайтеся, що на вашому комп’ютері встановлено інтерпретатор Python версії від 3.8 до 3.10. Якщо Python ще не інстальовано, його слід завантажити з офіційного сайту. Під час процесу встановлення Python критично важливо обов’язково встановити прапорець «Add Python to PATH»; це інтегрує Python та його інструменти, зокрема менеджер пакетів pip, у системні змінні середовища, що дозволить запускати їх з будь-якого місця у командному рядку. Після завершення встановлення рекомендується відкрити командний рядок (для Windows це можна зробити, натиснувши Win + R, ввівши cmd і натиснувши Enter; для Ubuntu слід відкрити «Термінал») та перевірити коректність інсталяції Python і pip, виконавши команди python --version та pip --version відповідно. У відповідь ви маєте побачити встановлені версії.

Наступним кроком є завантаження файлів самого додатку (рис. 3.1). Для цього необхідно перейти на сторінку проєкту «AI Image Editor» на GitHub [5]. На сторінці проєкту слід знайти кнопку «Code», яка зазвичай має зелений колір,

та натиснути її. У випадаючому меню, що з’явиться, оберіть опцію «Download ZIP» (Завантажити ZIP-архів).

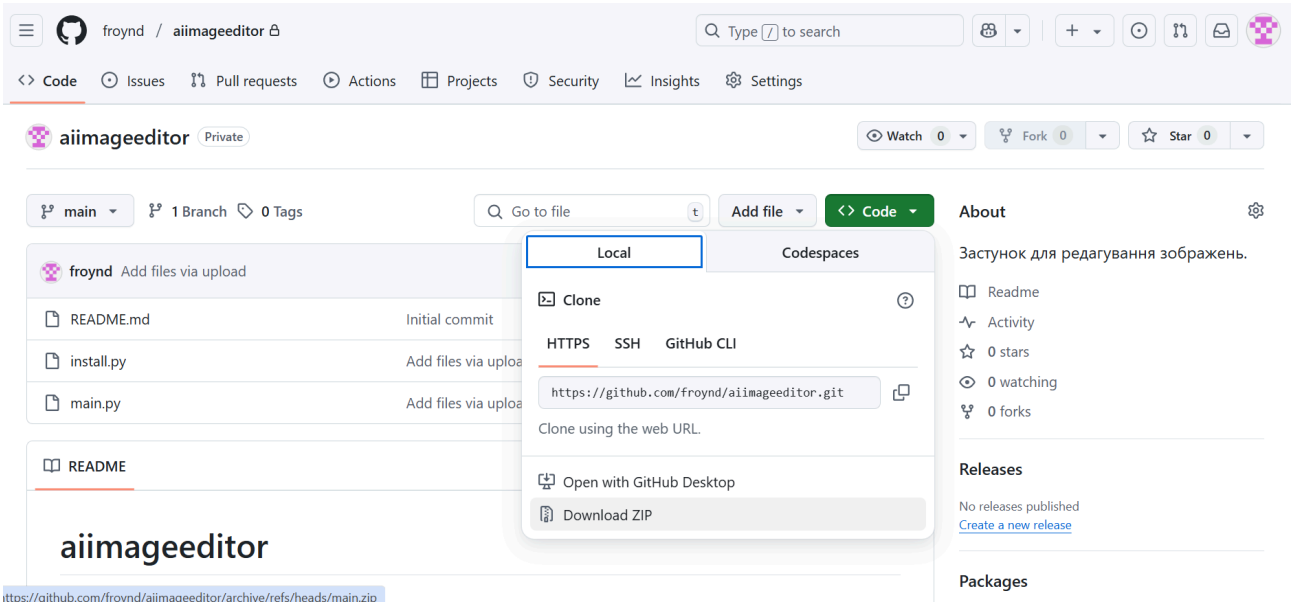


Рисунок 3.1 – Завантаження з Github

Після завершення завантаження, отриманий архів необхідно розпакувати в будь-яку зручну для вас директорію на вашому комп’ютері, наприклад, C:\AI_Image_Editor\ для користувачів Windows або /home/user/AI_Image_Editor/ для користувачів Ubuntu.

Тепер можна перейти до запуску графічного інсталятора. Відкрийте директорію, куди ви щойно розпакували файли проєкту, і знайдіть файл з назвою install.py, який є інсталятором. Для користувачів Windows достатньо двічі клацнути на файлі. Якщо файл не запускається автоматично, слід відкрити командний рядок, перейти у директорію проєкту за допомогою команди cd C:\AI_Image_Editor\ (замініть на ваш фактичний шлях) і виконати команду python install.py. Для користувачів Ubuntu/Linux необхідно відкрити термінал, перейти у директорію проєкту за допомогою команди cd /home/user/AI_Image_Editor/ (також замініть на ваш шлях) і виконати команду python3 install.py, або просто python install.py, якщо у вас встановлено лише одну версію Python.

Після успішного запуску test.py на екрані з’явиться графічне вікно інсталятора «AI Image Editor Installer». Процес встановлення буде проходити у кілька етапів, які інсталятор автоматично керує. Спершу інсталятор перевірить та, за потреби, встановить необхідні базові залежності, такі як PyQt5 та requests, про що ви побачите відповідні повідомлення у текстовому лозі інсталятора (рис. 3.2).

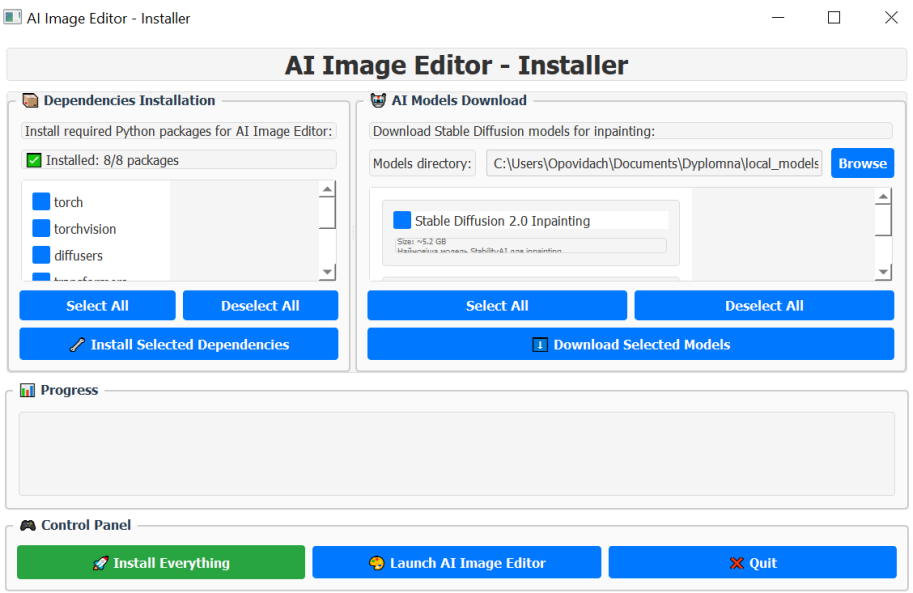


Рисунок 3.2 – Інтерфейс встановлювача

Далі, у вікні інсталятора з’явиться випадаючий список, де користувачу буде запропоновано обрати пристрій для PyTorch – «CPU» або «CUDA». Якщо у вас є сумісна відеокарта NVIDIA з підтримкою CUDA та достатнім об’ємом VRAM (рекомендовано 8 ГБ або більше), обов’язково оберіть «CUDA», що забезпечить максимальну швидкість роботи програми. Якщо ж відповідної відеокарти немає або виникають проблеми з її використанням, слід обрати «CPU»; програма працюватиме, але значно повільніше. Після вибору пристрою необхідно натиснути кнопку «Install» (Встановити), щоб розпочати основний процес. Інсталятор почне встановлювати всі необхідні Python-бібліотеки для функціонування «AI Image Editor», включаючи PyTorch, torchvision, diffusers, Pillow та інші. Після цього він перейде до завантаження великих попередньо

навчених моделей штучного інтелекту, зокрема Stable Diffusion Inpaint та Faster R-CNN, безпосередньо з репозиторіїв Hugging Face. Цей етап є найтривалішим і може зайняти від кількох хвилин до десятків хвилин, залежно від швидкості вашого інтернет-з'єднання та продуктивності комп'ютера; прогрес завантаження буде візуально відображатися на прогрес-барі, а детальний лог – у текстовому вікні. По завершенні всіх операцій ви побачите повідомлення про успішне встановлення, після чого зможете закрити вікно інсталятора, натиснувши кнопку «Close» (Закрити).

В пункті Dependencies Installation реалізовано функціонал який дозволяє користувачу встановити необхідні бібліотеки (рис. 3.3). Вибір бібліотек реалізовано у форматі checkbox. Також зверху є пункт Installed який позначає скільки в користувача встановлено бібліотек. Для встановлення необхідно вибрати потрібні бібліотеки та натиснути кнопку Install Selected Dependencies.

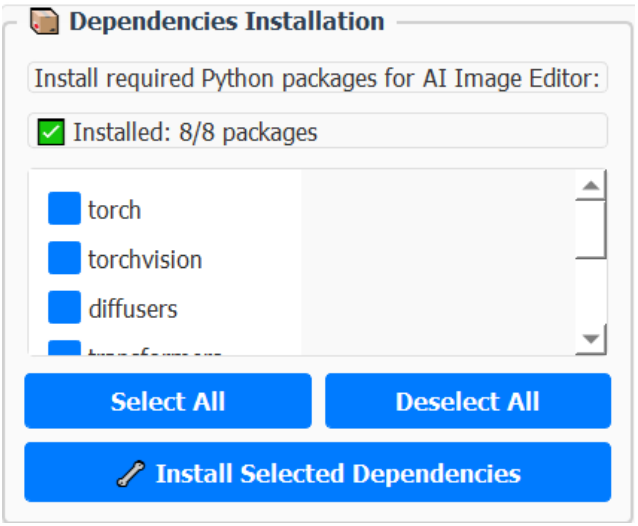


Рисунок 3.3 – Пункт Dependencies Installation

### 3.4 Документація програмного забезпечення, його інсталяція та структура проєкту

Для кращого розуміння принципів роботи додатку, його можливостей та порядку взаємодії з користувачем було розроблено повноцінну технічну

документацію. Вона містить детальний опис архітектури системи, інструкції з встановлення, пояснення щодо використання кожного елементу інтерфейсу, приклади типових сценаріїв роботи, а також рекомендації з усунення можливих помилок. Документація покликана забезпечити ефективне впровадження програмного продукту, його супровід та подальший розвиток.

На рисунку 3.4 представлено головне вікно програмного забезпечення AI Image Editor – настільного додатку для аналізу та редагування зображень з інтеграцією штучного інтелекту. Інтерфейс реалізовано на основі бібліотеки PyQt5, що забезпечує інтуїтивну взаємодію користувача з функціоналом системи. Основні компоненти включають область відображення зображення, список виявлених об’єктів, історію змін, а також панель керування з кнопками для завантаження, аналізу, генерації та скидання зображень. Для реалізації генеративного інпейнтінгу використано модель Stable Diffusion Inpainting, що дозволяє автоматично редагувати виділені об’єкти на основі текстового опису користувача. Система підтримує обробку зображень у фонових потоках, зберігаючи чутливість інтерфейсу під час виконання обчислювально інтенсивних завдань.

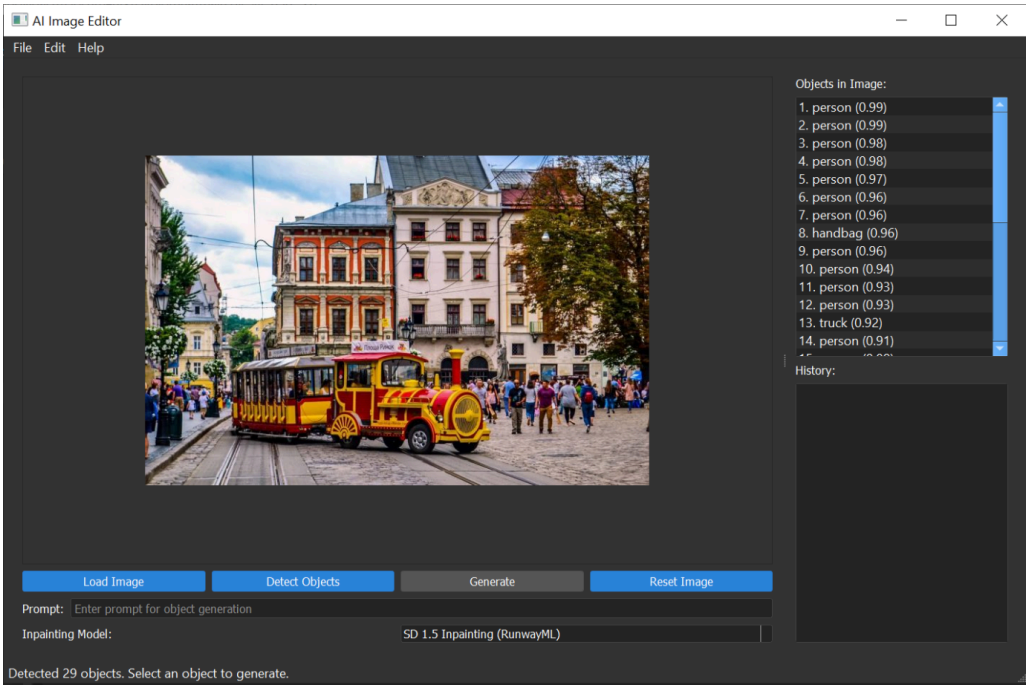


Рисунок 3.4 – Основний додаток

У пункті меню File зібрано базові операції роботи з файлами, необхідні для зручної взаємодії з зображеннями у додатку AI Image Editor. За допомогою підпункту Open Image користувач може завантажити потрібне зображення з локального пристрою для подальшої обробки. Комбінація клавіш Ctrl+O дозволяє виконати цю дію швидко, без потреби переходити в меню вручну.

Після виконання необхідних змін та редагування зображення, користувач може зберегти результат, використовуючи підпункт Save Image або комбінацію клавіш Ctrl+S. При цьому доступні стандартні формати збереження, такі як PNG або JPG.

Для завершення роботи з додатком передбачено підпункт Exit, що закриває програму. Його можна активувати також через клавішу Ctrl+Q, що дозволяє швидко вийти з програми, зберігши час при щоденному використанні. Таким чином, меню File забезпечує повний цикл базової взаємодії користувача із зображеннями – від відкриття до збереження та завершення сеансу (рис. 3.5).

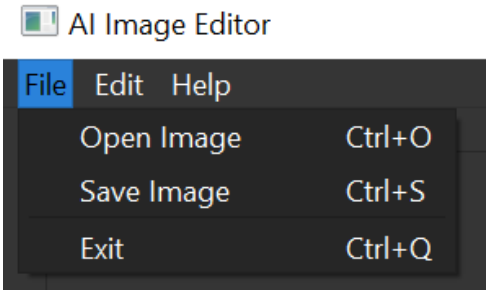


Рисунок 3.5 – Пункт File

В пункті Edit реалізовано можливість прибрати «Скинути» зображення – Reset Image (рис. 3.6).

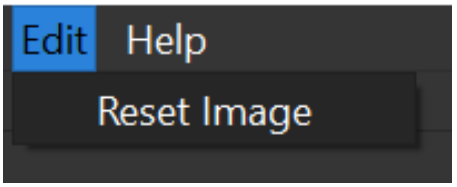


Рисунок 3.6 – Пункт Edit

У пункті Help реалізовано функціонал довідки, який містить основну інформацію про програму (рис. 3.7).

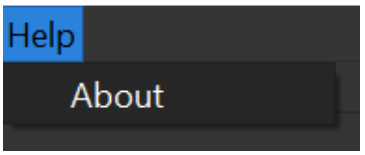


Рисунок 3.7 – Пункт Help

Найбільший простір займає саме нами вибране зображення, яке в програму можна завантажити через кнопку Load Image, яка відкриє провідник через який і здійснюється вибір зображення. Кнопка Detect Object – запускає визначення об’єктів на фото. Generate – Відповідає за старт генерації, але перед початком необхідно прописати Prompt. Reset Image – Скидає наше зображення.

Пункт Prompt – пункт в якому ми прописуємо текстовий опис того, що хочемо згенерувати. Пункт Inpainting Model – дозволяє вибрати модель для генерації з випадаючого списку (рис. 3.8).

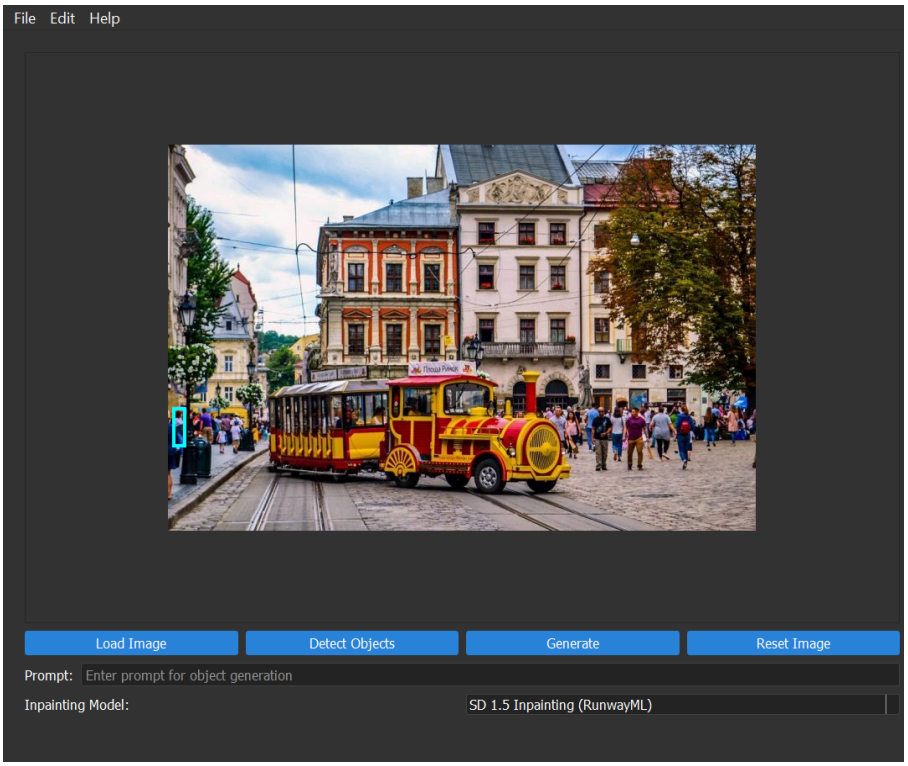


Рисунок 3.8 – Ліва частина інтерфейсу додатку

Пункт Objects in Image – демонструє нам які є об’єкти на вибраному фото. History – історія наших промптів.

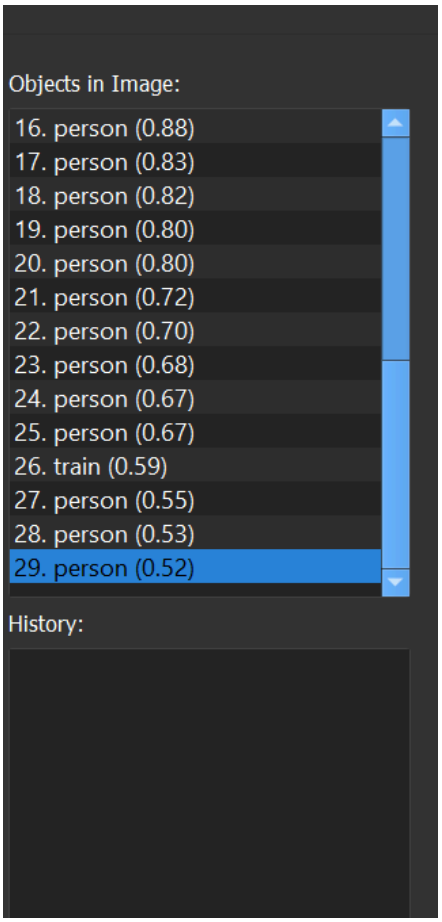


Рисунок 3.9 – Права частина інтерфейсу додатку.

**Висновки до розділу 3**

У рамках 3 розділу було всебічно розглянуто та детально описано ключові аспекти, пов’язані з реалізацією, тестуванням, налагодженням, документуванням та розгортанням програмного забезпечення «AI Image Editor».

Процес розробки інформаційно-комп’ютерної системи базувався на архітектурних рішеннях, деталізованих у попередніх розділах, та був сфокусований на створенні високопродуктивного та стабільного додатку. Інтеграція передових моделей глибокого навчання, таких як Faster R-CNN для

детекції об’єктів та Stable Diffusion Inpaint для генеративного інпейнтінгу, вимагала ретельного підходу до керування ресурсами, особливо відеопам’яттю.

Це було досягнуто шляхом вибору оптимальних типів даних для моделей та реалізації ефективних механізмів їх завантаження. Особливу увагу було приділено забезпеченню графічного інтерфейсу користувача: всі ресурсоємні операції, що включають інференс моделей ШІ, були перенесені у окремі фонові потоки, а взаємодія між ними та інтерфейсом реалізована виключно через безпечний механізм сигналів та слотів фреймворку PyQt5, що повністю усунуло проблему блокування інтерфейсу.

Ретельне тестування та налагодження стали невід’ємною частиною циклу розробки. Хоча формальні юніт-тести не були основним фокусом, кожна критична функція та компонент тестувалися ізольовано, зокрема, перевірялася коректність генерації масок та обробки зображень. Інтеграційне тестування підтвердило безперебійну взаємодію між графічним інтерфейсом, фоновими потоками та ШІ-моделями, гарантуючи коректну передачу даних та відображення результатів. Функціональне тестування, що охопило всі заявлені функціональні прецеденти від завантаження зображення до його збереження після інпейнтінгу, підтвердило повну відповідність поведінки системи визначеним вимогам. Тестування інтерфейсу користувача забезпечило його інтуїтивність, чутливість та коректне відображення на різних конфігураціях.

Процес налагодження активно використовував як базові методи логування та друку, так і потужні інтегровані налагоджувачі, а також широке застосування механізмів обробки винятків (try...except), що значно підвищило відмовостійкість програми, дозволяючи коректно реагувати на непередбачені ситуації, такі як некоректні вхідні дані або відсутність ресурсів, без аварійного завершення.

У рамках цього розділу було також детально визначено системні вимоги до апаратного та програмного забезпечення, що є критично важливим для користувачів. Було підкреслено значну перевагу наявності GPU з підтримкою

CUDA для досягнення оптимальної продуктивності, що прямо впливає на швидкість обробки зображень та користувацький досвід.

Одним з ключових практичних досягнень розділу 3 є розробка та реалізація власного графічного інсталятора програмного забезпечення. Це рішення дозволило вирішити типові проблеми розгортання складних ШІ-додатків, пов’язані з великим розміром дистрибутивів та необхідністю динамічного завантаження моделей. Розроблений інсталятор автоматизує процес встановлення необхідних Python-залежностей та завантаження моделей безпосередньо з Hugging Face, надаючи користувачеві інтуїтивно зрозумілий графічний інтерфейс з прогрес-баром та детальним логом.

# ВИСНОВКИ

У результаті виконання кваліфікаційного проєкту було успішно створено додаток для обробки та редагування зображень з використанням мови програмування Python та технологій штучного інтелекту. Проєкт охопив усі етапи розробки – від аналітичного до практичного, що дозволило реалізувати повноцінне та функціональне програмне рішення відповідно до визначених вимог.

Проведено аналіз сучасних методів обробки та редагування зображень, що дозволило обрати найактуальніші підходи – зокрема використання бібліотек OpenCV, PIL та моделей глибокого навчання, що відкриває широкі можливості для автоматизації та інтелектуального покращення зображень.

Визначено як функціональні, так і нефункціональні вимоги до додатку. Сформульовано основні сценарії взаємодії користувача з інтерфейсом та системою в цілому. Для візуалізації архітектури системи розроблено UML-діаграми, що дозволило впорядкувати процес розробки та структурувати компоненти.

Обрано оптимальний стек технологій: мова програмування Python та бібліотек PyQt5, PyTorch, torchvision, diffusers та Pillow для роботи з зображеннями, а також застосовано моделі машинного навчання для реалізації AI-функціоналу. Алгоритми роботи зображень були детально описані, що забезпечило надійність та розширюваність системи.

Здійснено практичну реалізацію додатку відповідно до поставлених вимог. Інтерфейс реалізовано з урахуванням принципів зручності та інтуїтивності, функціонал покриває потреби редагування, збереження та аналізу зображень, включаючи інтелектуальні підказки.

Проведено всебічне тестування додатку, що охоплює функціональні можливості, перевірку точності роботи інтегрованих моделей та оцінку продуктивності на різних конфігураціях. Тестування засвідчило стабільність роботи системи та відповідність заявленим вимогам.

Описано процес компіляції виконавчого файлу для встановлення додатку на кінцеві пристрої. Надано інструкції зі створення інсталяційного пакету, що забезпечує зручність розгортання та використання розробленого програмного забезпечення.

Написано технічну документацію до додатку, що містить опис основного функціоналу, інструкції користувача та специфікацію мінімальних і рекомендованих системних вимог для запуску програмного забезпечення.

Розроблений додаток повністю відповідає поставленим цілям та демонструє ефективне поєднання класичних методів обробки зображень і сучасних підходів на основі штучного інтелекту. Реалізація проєкту підтвердила доцільність обраного технічного рішення, а також надала цінний практичний досвід у розробці програмного забезпечення, тестуванні, документуванні та підготовці інсталяційних пакетів. Отримані результати можуть бути використані як основа для подальшого розширення функціоналу або інтеграції в більш складні системи.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. COCO – Common Objects in Context. URL: https://cocodataset.org/ (date of access: 01.02.2025).

2. Gad A. F. Faster R-CNN Explained for Object Detection Tasks. DigitalOcean – Cloud Infrastructure for Developers. URL: https://www.digitalocean.com/community/tutorials/faster-r-cnn-explained-object-detection (date of access: 01.05.2025).

3. Generate Images from Text in Python – Stable Diffusion – GeeksforGeeks. URL: https://www.geeksforgeeks.org/generate-images-from-text-in-python-stable-diffusion/ (date of access: 12.03.2025).

4. GIMP. URL: https://www.gimp.org/ (date of access: 18.04.2025).

5. GitHub. Build and ship software on a single, collaborative platform. GitHub. URL: https://github.com/ (date of access: 07.02.2025).

6. Getting started – skimage 0.25.2 documentation. URL: https://scikit-image.org/docs/stable/u ser_guide/getting_started.html (date of access: 27.03.2025).

7. Hugging Face – The AI community building the future. Hugging Face – The AI community building the future. URL: https://huggingface.co/ (date of access: 16.04.2025).

8. Microsoft. Visual Studio Code – Code Editing. URL: https://code.visualstudio.com/ (date of access: 09.05.2025).

9. OpenCV documentation index. URL: https://docs.opencv.org/ (date of access: 28.04.2025).

10. Pillow (PIL Fork). URL: https://pillow.readthedocs.io/en/stable/ (date of access: 17.03.2025).

11. PyTorch. URL: https://pytorch.org/ (date of access: 19.04.2025).

12. Thonny, Python IDE for beginners. URL: https://thonny.org/ (date of access: 24.02.2025).

13. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/python/python_virtualenv.asp (date of access: 03.02.2025).

14. Welcome to Python.org. URL: https://www.python.org/ (date of access: 15.03.2025).

15. What is Python? Opensource.com. URL: https://opensource.com /resources/python (date of access: 29.04.2025).