МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп’ютерної інженерії

ВОВК Владислав Сергійович

Генерування художніх зображень за текстовим описом / Generating artistic images using text description

спеціальність: 123 – Комп’ютерна інженерія

освітньо–професійна програма – Комп’ютерна інженерія

Кваліфікаційна робота

Виконав: студент групи КІ–41

ВОВК Владислав Сергійович

Науковий керівник

к.т.н. доц. Піцун О.Й.

ТЕРНОПІЛЬ – 2026

АНОТАЦІЯ

Вовк В.С. Генерування художніх зображень за текстовим описом. – Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 123 «Комп'ютерна інженерія», освітньо-професійна програма. Західноукраїнський національний університет, Тернопіль, 2026.

Робота написана обсягом __ сторінок і містить __ рисунків, __ таблиць, __ додатки та __ джерел за переліком посилань.

Метою кваліфікаційної роботи є розробка настільного програмного засобу для автоматизованого синтезу художніх і фотореалістичних зображень за природною мовою користувача. У роботі узагальнено сучасні підходи до текст-у-зображення (text-to-image), зокрема дифузійні моделі та умовне моделювання за текстовим описом; проведено порівняльний аналіз локальних моделей на базі бібліотеки Hugging Face Diffusers і хмарних API (Google Gemini/Stability AI/OpenRouter). Запропоновано архітектуру застосунку з графічним інтерфейсом на CustomTkinter, модулем обробки промптів з підтримкою локального розширення та зовнішніх LLM-провайдерів (Google, Groq, Ollama), а також із збереженням шаблонів і журналу генерацій у базі даних SQLite. Реалізовано кілька альтернативних шляхів генерації для різних обчислювальних ресурсів і політик приватності; конфігурація ключів і тем інтерфейсу винесена у файл `config.env` з можливістю редагування з вбудованого вікна налаштувань.

Ключові слова: генеративний штучний інтелект; text-to-image; дифузійні моделі; Stable Diffusion; Python; настільний застосунок; SQLite; промпт-інжиніринг.

ANNOTATION

Vovk V.S. Generating artistic images using text description. – Manuscript.

Qualification work for the Bachelor degree in specialty 123 “Computer Engineering”, educational and professional program. Western Ukrainian National University, Ternopil, 2026.

The work is written in __ pages and contains __ figures, __ tables, __ appendices and __ sources in the reference list.

The purpose is to develop a desktop tool for automated synthesis of artistic or photorealistic images from user text. The thesis summarizes contemporary text-to-image approaches, focusing on diffusion models and text conditioning; it compares local Diffusers-based pipelines with cloud APIs (Google Gemini, Stability AI, OpenRouter). The designed architecture combines a CustomTkinter GUI, prompt post-processing with local heuristics and optional LLM providers (Google, Groq, Ollama via `PROMPT_ENHANCE_PROVIDER`), and SQLite persistence for templates and generation history. API keys and UI theme are centralized in `config.env` with an in-app settings editor. Multiple backends support different hardware and privacy constraints.

Keywords: generative AI; text-to-image; diffusion models; Stable Diffusion; Python; desktop application; SQLite; prompt engineering.

ЗМІСТ

					БР.КСМ.07165/17.00.00.000 ПЗ		
Змн.	Лист	№ докум.	Підпис	Дата			
Розробив		Іванко Л.В.			Генерування художніх зображень за текстовим описом	Літ.	Арк. 4 / Акрушів 34
Перевір.		Ігнатєв І.В.					
Консульт.		Піцун О.Й.				ТНЕУ,ФКІТ, КСМ–43/2	
Н. Контр.		Гураль І.В.					
Затвердив		Березький О.М					

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AI	–	Artificial Intelligence
API	–	Application Programming Interface
CTk	–	CustomTkinter
CLIP	–	Contrastive Language–Image Pre-training
CNN	–	Convolutional Neural Network
CUDA	–	Compute Unified Device Architecture
DDPM	–	Denoising Diffusion Probabilistic Model
GUI	–	Graphical User Interface
GPU	–	Graphics Processing Unit
HF	–	Hugging Face
HTTP	–	HyperText Transfer Protocol
JSON	–	JavaScript Object Notation
LLM	–	Large Language Model
LoRA	–	Low-Rank Adaptation
OOM	–	Out Of Memory
RGB	–	Red, Green, Blue
REST	–	Representational State Transfer
SD, SDXL, SD3	–	Stable Diffusion
SQL	–	Structured Query Language
SQLite	–	Structured Query Language Lite
T2I	–	Text-to-Image
UI	–	User Interface
URL	–	Uniform Resource Locator
VAE	–	Variational Autoencoder
VRAM	–	Video Random Access Memory

ВСТУП

У сучасному інформаційному суспільстві обсяги візуального контенту стрімко зростають: реклама, соціальні мережі, ігрова індустрія та навчальні матеріали потребують ілюстрацій у стислі терміни. Паралельно розвиваються генеративні моделі на основі глибинного навчання, які перетворюють текстовий опис на зображення, що візуально відповідає задуму користувача. Такі системи отримали назву text-to-image (T2I) і стали комерційно доступними як веб-сервіси, API й відкриті моделі для локального запуску.

Початковий етап досліджень у генерації зображень був пов'язаний із генеративно-змагальними мережами (GAN) та варіаційними автоенкодерами (VAE), однак останні роки принесли перевагу дифузійним моделям класу DDPM, які демонструють стабільнішу якість і кращу керованість [1–4]. Сучасні T2I-пайплайни спираються на багатомодальні енкодери на кшталт CLIP, які перетворюють природну мову на векторні представлення, узгоджені з процесом денойзингу [5, 6].

Практична реалізація T2I-інструменту стикається з інженерними компромісами. Локальний запуск великих моделей потребує потужної GPU та значного часу інференсу; хмарні API забезпечують швидкий результат без локального обладнання, але вимагають підключення до інтернету й породжують питання щодо конфіденційності промпта. Крім того, однаковий запит може по-різному інтерпретуватися різними моделями, тому важлива уніфікована обробка промпта і можливість зберігати успішні шаблони для повторного використання.

Об'єкт дослідження процес синтезу зображень за текстовим описом у настільному програмному застосунку під операційну систему Microsoft Windows.

Предмет дослідження програмні методи побудови T2I-пайплайнів на базі дифузійних моделей, обробки промптів із залученням локальних евристик та зовнішніх LLM, інтеграції локального інференсу через Hugging Face Diffusers із хмарними API, а також збереження історії генерацій у базі даних SQLite.

Мета кваліфікаційної роботи розробка настільного застосунку з графічним інтерфейсом, який забезпечує генерацію зображень із підтримкою локального та хмарного виконання залежно від обчислювальних ресурсів і вимог до конфіденційності даних.

Для досягнення поставленої мети необхідно вирішити такі основні задачі:

-провести порівняльний аналіз сучасних підходів до генерації зображень за текстом (GAN, VAE, дифузійні моделі) та програмних засобів їх реалізації;

-розробити узагальнений алгоритм дифузійного синтезу зображень з підтримкою локального та хмарного режимів виконання;

-розробити алгоритм формування та покращення текстових запитів із використанням шаблонів і зовнішніх LLM;

-спроектувати структуру бази даних SQLite для зберігання шаблонів промптів і журналу генерацій;

-програмно реалізувати настільний застосунок, провести його тестування та оцінити практичні результати роботи.

Для реалізації обрано настільний застосунок під Microsoft Windows на базі CustomTkinter поверх стандартного Tkinter. Таке рішення забезпечує локальний доступ до файлової системи, просте розгортання без веб-сервера й сучасний інтерфейс завдяки темам оформлення CTk. Веб-застосунок спростив би дистрибуцію, але вимагав би зберігання ключів API на серверній стороні та ускладнив би прямий локальний інференс PyTorch на GPU користувача; тому для роботи з вагами Diffusers на робочій станції доцільніший десктопний процес з окремим потоком для тривалих викликів моделі.

1 АНАЛІЗ ЗАСОБІВ ГЕНЕРУВАННЯ ЗОБРАЖЕНЬ

1.1 Підходи та методи синтезу зображень за текстом

З формальної точки зору, синтез зображення за текстовим описом є задачею відображення з простору умов рядка тексту, необов'язкового негативного промпту та параметрів семплера у простір яскравості пікселів високої розмірності. На відміну від задач класифікації, де для кожного входу існує єдина правильна відповідь, у генерації зображень модель відтворює розподіл правдоподібних варіантів. Це означає, що однаковий текстовий запит при повторному виконанні дасть різні результати навіть за схожих налаштувань семплера обставина, яку необхідно враховувати під час лабораторних порівнянь і відтворення експериментів.

Генеративно-змагальні мережі (GAN) тривалий час залишалися еталоном швидкої генерації зображень високої роздільності. Принцип їхньої роботи полягає у протистоянні двох нейронних мереж: генератор формує зображення, підлаштовуючись під статистику навчальних даних, тоді як дискримінатор відсіює неправдоподібні результати. Головною перевагою GAN є швидкий інференс лише один прямий прохід через мережу. Водночас суттєвими недоліками залишаються нестабільність навчання, ризик зменшення різноманітності генерованих зображень (колапс режимів) та труднощі з утриманням текстової умови протягом усього синтезу [7].

Варіаційні автоенкодери (VAE) пропонують альтернативний підхід через явне моделювання латентного простору. Енкодер стискає зображення до компактного векторного представлення, а декодер відновлює пікселі з цього коду. Попри те, що окремий VAE за суб'єктивною якістю поступається найкращим GAN, саме принцип латентного стиснення ліг в основу сучасних

дифузійних архітектур: денойзинг у малому латентному просторі значно дешевший за обчисленнями, ніж у повній розмірності пікселів [8].

Дифузійні моделі навчаються поступово відновлювати зображення з шуму, виконуючи цей процес покроково. На етапі інференсу кількість кроків та вибір конкретного семплера безпосередньо впливають як на швидкість генерації, так і на плавність траєкторії у просторі. Latent Diffusion Model переносить цей цикл денойзингу у латентний простір VAE, тому типовий Stable Diffusion є не “магією одного проходу”, а послідовністю чисельних оновлень, які нагадують ітеративне наближення.

Архітектура лінійки Stable Diffusion 1.5 складається з трьох основних компонентів: CLIP-текстового енкодера, що перетворює текстовий запит на умовні ознаки; UNet, який виконує денойзинг у латентному просторі; та VAE-декодера, що відображає латент назад у піксельне зображення. Новіші архітектури вносять зміни до окремих вузлів, проте загальний ланцюжок “текст → умовні ознаки → цикл денойзингу → декодування” залишається зрозумілим інженерним каркасом як для локального розгортання, так і для хмарних API [1, 3]. Три основні парадигми генерації зображень та узагальнений шлях перетворення текстового опису на вихідне зображення наведено на рисунку 1.1

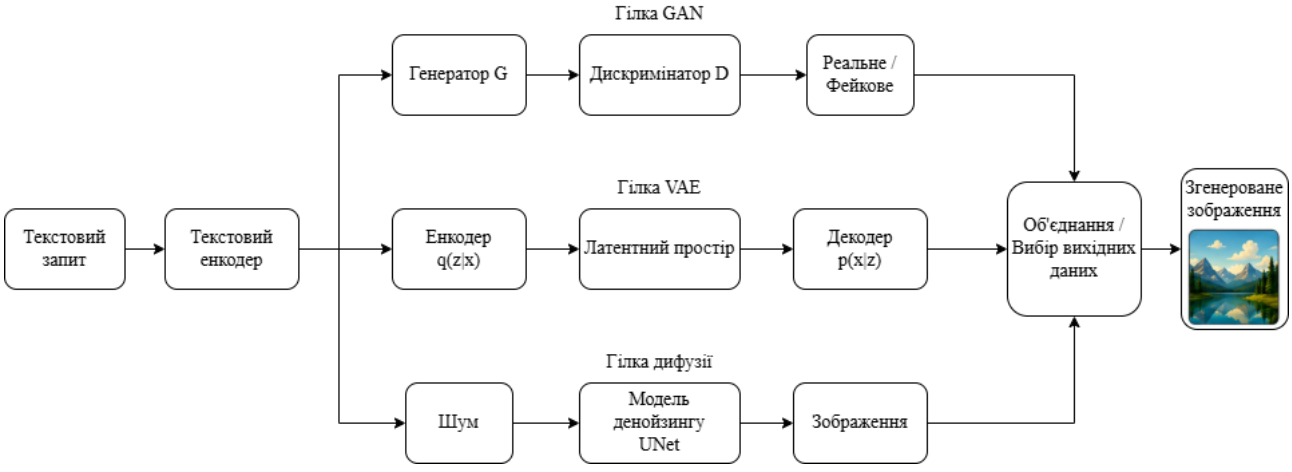

Рисунок 1.1 – Три парадигми генерації (GAN, VAE, дифузія) та узагальнений шлях text-to-image

Практика складання промптів у спільнотах розробників зазвичай зводиться до набору коротких англомовних тегів, що дозволяє точніше відповідати розподілу навчальних даних. Водночас питання ліцензування ваг моделей, дотримання умов використання сервісів та академічної чесності щодо запозичення стилів залишаються обов’язковим контекстом будь-якого дослідження: перенесення обчислень на власний комп’ютер не скасовує правових та етичних вимог до роботи з генеративними системами.

Окремої уваги заслуговує механізм крос-уваги (cross-attention), що є ключовим інструментом поєднання текстових та візуальних ознак у сучасних дифузійних архітектурах. На відміну від простого конкатенування умовного вектора з вхідними даними, механізм крос-уваги дозволяє моделі вибірково зосереджуватися на різних частинах текстового опису під час генерації кожної ділянки зображення. Це означає, що іменник "кіт" впливатиме переважно на пікселі, де модель очікує присутність тварини, а прикметник "синій" на колір відповідного об'єкта, а не на весь фон рівномірно.

Важливим концептуальним кроком у розвитку архітектур стало введення так званого guidance distillation технології компресії знань із повного дифузійного ланцюжка в модель, здатну генерувати якісні результати за значно меншої кількості кроків семплювання. Підходи типу LCM (Latent Consistency Model) або Turbo-дифузія дозволяють отримувати придатне зображення вже за чотири-вісім кроків замість стандартних двадцяти-п'ятдесяти, що суттєво знижує вимоги до часу очікування в інтерактивних застосунках. Разом із тим слід зазначити, що прискорені варіанти можуть демонструвати дещо нижчу деталізацію при складних сценах із великою кількістю об'єктів.

З точки зору практичного впровадження, дифузійні моделі також різняться між собою за форматом збереження ваг. Формат safetensors забезпечує безпечніше та швидше завантаження порівняно зі застарілим форматом pickle-

based .pt, оскільки виключає можливість виконання довільного коду під час десеріалізації. Це особливо актуально при роботі з моделями, завантаженими з відкритих репозиторіїв, де автентичність кожного файлу не завжди може бути перевірена через офіційні канали.

1.2 Програмні засоби та технологічний стек

Для розробки настільного застосунку під операційну систему Windows найбільш обґрунтованим вибором є мова програмування Python. Вона дозволяє без зайвих архітектурних витрат інтегрувати PyTorch, мережеві клієнти та бібліотеки для побудови графічного інтерфейсу в єдину кодову базу. За наявності відеокарти з підтримкою CUDA застосунок забезпечує прийнятний час генерації для моделей рівня SD 1.5; у разі її відсутності програма повинна чітко інформувати користувача про збільшену тривалість сеансу, а не зависати без пояснень.

Бібліотека Hugging Face Diffusers приховує деталі реалізації кроків семплера за стабільним інтерфейсом пайплайна, що суттєво спрощує підключення різних моделей [11]. Пакет Transformers забезпечує токенізатори для роботи з текстовими умовами, тоді як safetensors та Accelerate допомагають ефективно керувати вагами та відеопам’яттю в умовах обмежених ресурсів VRAM [12]. Використання цього стеку продиктоване не лише зручністю: він дозволяє уникнути повторної реалізації складної математики при переході між різними версіями моделей.

На рівні графічного інтерфейсу бібліотека CustomTkinter надає сучасніший вигляд елементів управління порівняно зі стандартним Tkinter та підтримує винесення тривалих обчислень у окремі потоки для збереження відгуку

інтерфейсу [14]. Бібліотека Pillow відповідає за формування попереднього перегляду та гнучкий експорт результатів у формати PNG, JPEG або WebP через стандартний діалог збереження файлів [15].

Хмарні сервіси Gemini, Stability AI, OpenRouter та інші підключаються як окремі модулі. Щоб не дублювати логіку нормалізації тексту, промпт, підготовлений для Diffusers, проходить через єдиний спільний шар і лише після цього формується REST-запит до зовнішнього API. Агрегатори на кшталт OpenRouter додають проміжний шлюз і потребують окремої обробки формату повідомлень для цього реалізовано спеціалізовані гілки пакування запиту та розпакування відповіді. Локальна база даних SQLite не вимагає встановлення окремого сервера і повністю задовольняє потреби даної роботи [18].

Типовий сценарій роботи користувача у застосунку охоплює такі кроки: формулювання сцени, підбір тегів, за потреби розширення промпту за допомогою мовної моделі, перевірка лімітів обраного провайдера, запуск генерації та збереження результату. Слід зазначити, що деякі моделі на Hugging Face Hub є закритими (gated) та вимагають автентифікаційного токена. Порівняно з MATLAB, Python залишається практичним і обґрунтованим вибором для роботи з відкритими вагами Stable Diffusion та REST-інтеграціями; MATLAB доцільний як допоміжний інструмент для окремих аналітичних обчислень, однак не є основою клієнтської частини застосунку.

Бібліотека Accelerate від Hugging Face відіграє роль прошарку між кодом PyTorch і різними апаратними конфігураціями, автоматично вирішуючи питання розподілу ваг між відеопам'яттю GPU та оперативною пам'яттю системи.

Функції model.enable_model_cpu_offload() та model.enable_sequential_cpu_offload() дозволяють запускати дифузійні пайплайни навіть на картах з обмеженим обсягом VRAM, переміщуючи неактивні модулі в оперативну пам'ять між кроками денойзингу. Вибір між двома режимами offload визначається компромісом між швидкістю та обсягом

відеопам'яті, що використовується одночасно. Порівняння трьох основних підходів до генерації зображень за ключовими критеріями наведено в таблиці 1.1

Таблиця 1.1 – Порівняння основних підходів генерації зображень

Критерій	GAN	VAE	Дифузія / латентна дифузія
Стабільність навчання %	Низька (~35%)	Середня (~60%)	Висока (~85%)
Якість високої роздільності %	~78%	~54%	~88%
Швидкість інференсу	Зазвичай висока	Висока	Нижча через багатокроковий семплер
Зрілість практик розгортання %	~52%	~73%	~91%

Слід враховувати також особливості версійності самого Python при побудові середовища розробки. Починаючи з Python 3.11, з'явились оптимізації інтерпретатора, що покращують загальну швидкість виконання, проте сумісність PyTorch та деяких розширень C/C++ з конкретними мінорними версіями може відрізнятися. Тому зафіксована версія у файлі .python-version або у налаштуваннях pyenv є частиною відтворюваного середовища нарівні з файлом requirements.txt. Управління залежностями через uv замість pip суттєво пришвидшує встановлення великих пакетів, таких як torch, завдяки паралельному завантаженню та кешуванню колес.

Важливим аспектом побудови графічного інтерфейсу є коректна обробка DPI-масштабування на сучасних моніторах із високою щільністю пікселів. CustomTkinter автоматично адаптує розміри елементів до системного масштабування Windows, що усуває проблему розмитих елементів управління, характерну для стандартного Tkinter. Водночас при ручному встановленні розмірів вікна та вбудованих зображень необхідно враховувати системний DPI

через ctypes або tkinter.winfo_fpixels(), щоб прев'ю згенерованих зображень відображалося з правильними пропорціями на різних конфігураціях моніторів.

1.3 Порівняльний аналіз інструментів і сценаріїв розгортання

Локальний інференс є доцільним у тих випадках, коли на робочій станції наявна відповідна відеопам’ять або коли внутрішня безпекова політика організації забороняє передачу текстових запитів і зображень за межі периметра мережі. Витрати при цьому виражаються у капітальних вкладеннях у апаратне забезпечення та поточному споживанні електроенергії. Натомість конфіденційність даних гарантована: усі матеріали залишаються на пристрої користувача і не передаються стороннім сервісам.

Хмарний підхід знімає залежність від характеристик клієнтського обладнання та нерідко є єдиним практичним рішенням для машин без дискретної відеокарти. Разом із тим він привносить нові обмеження: необхідність підписки або оплати за запитами, квоти на використання, мережеву затримку і, що особливо важливо, юридичні умови обробки та зберігання даних провайдером. Для державних установ і організацій, що працюють з чутливою інформацією, ці умови нерідко є вирішальним аргументом на користь локального режиму навіть за рахунок значно більшого часу генерації.

Оцінювання застосунку не повинно обмежуватися лише суб’єктивною якістю зображень. Не менш важливою є стійкість системи: помилки типу OOM (нестача відеопам’яті) на GPU усуваються зниженням роздільності або розміру батчу, тоді як HTTP-помилки 401 та 429 від хмарного API потребують обробки через повтор із затримкою та відображення зрозумілого статусу у вікні

застосунку — без аварійного завершення процесу. Порівняльний аналіз локального інференсу та хмарних API наведено в таблиці 1.2

Таблиця 1.2 – Локальний інференс і хмарні API

Критерій	Локально (Diffusers)	Хмарний API
Обчислювальні ресурси	Дискретна GPU потрібна для прийнятного часу; на CPU інференс можливий із кратним збільшенням тривалості сеансу	Обчислення виконує постачальник; на клієнтській машині GPU не обов’язкова
Конфіденційність промпта та зображення	Дані обробляються на ПК користувача	Передача на сервер згідно з умовами використання й політикою провайдера
Економіка	Капітальні витрати на апаратуру та експлуатаційне споживання електроенергії	Тарифікація за підпискою або обсягом запитів відповідно до обраного сервісу
Доступний набір моделей	Каталог Hugging Face Hub за умовами ліцензії кожної моделі	Фіксований або розширений каталог API; агрегатори на кшталт OpenRouter додають проміжний шлюз

Супровід локального стеку передбачає відстеження сумісності версій драйвера та збірки PyTorch між різними середовищами запуску. Для хмарних рішень актуальним завданням є ротація API-ключів і моніторинг лімітів. З практичної точки зору для дипломної роботи оптимальним виглядає комбінований підхід: локальна перевірка архітектурних рішень і хмарна демонстрація результатів. Такий режим є цілком реалістичним і відображає типовий інженерний компроміс між якістю, швидкістю та безпекою даних.

Порівняльне тестування локального та хмарного режимів на однакових промптах дає корисні практичні результати, навіть якщо вибірка є невеликою і не дозволяє робити статистично значущих висновків. Фіксація часу від надсилання запиту до отримання кінцевого файлу включає мережеву затримку, час генерації на стороні провайдера та час передачі зображення назад клієнту. У локальному режимі цей ланцюжок скорочується до чистого часу виконання пайплайна на доступному апаратному забезпеченні. Обидва показники мають бути задокументовані разом зі специфікацією апаратної конфігурації.

Ще одним чинником вибору між підходами є стабільність якості результатів у часі. Хмарні провайдери можуть оновлювати базові моделі без зміни назви ендпоінта, що призводить до непомітної зміни характеристик генерації. Локальний інференс із зафіксованим ідентифікатором ревізії моделі на Hugging Face Hub, навпаки, гарантує відтворюваність результатів за однакових seed-значень семплера. Для академічних цілей ця відтворюваність є критичною вимогою під час повторного проведення тестів або незалежної перевірки результатів роботи.

З операційної точки зору, хмарні провайдери нерідко запроваджують або змінюють квоти на використання без завчасного повідомлення у вільних тарифних планах. Це створює ризик раптового переривання роботи застосунку під час демонстрації або лабораторного тестування, якщо ліміти не відстежуються програмно. Реалізація простого лічильника виконаних запитів із порівнянням із відомою квотою провайдера дозволяє завчасно сповіщати користувача і уникати ситуацій, коли сеанс генерації переривається в середині роботи через код відповіді 429.

Серед стандартних кількісних метрик оцінювання генеративних моделей у науковій літературі найширшого поширення набули FID (Fréchet Inception Distance) та CLIPScore. Перша вимірює відстань між розподілами ознак реальних і синтезованих зображень у просторі ознак нейронної мережі [19], друга —

ступінь узгодженості між текстовим запитом та отриманим зображенням [32]. Однак для оцінювання настільної програми не менш важливими критеріями є суб'єктивна відповідність результату брифу, відсутність помітних артефактів та відтворюваність результатів за однакових параметрів семплера. Латентний пайплайн Stable Diffusion із текстовим енкодером CLIP, денойзингом засобами UNet та декодуванням через VAE наведено на рисунку 1.2

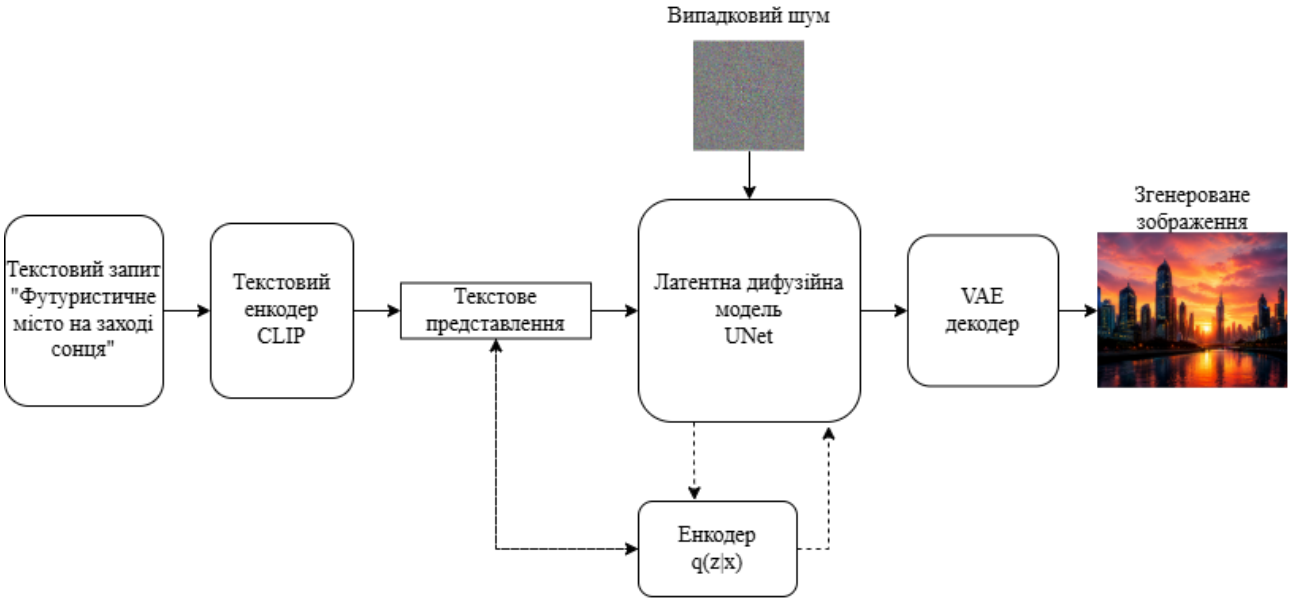

Рисунок 1.2 – Латентний пайплайн Stable Diffusion: CLIP, UNet у латенті, обмеження токенів і VAE-декодер

Для моделей рівня SD 1.5 необхідно враховувати обмеження токенізатора CLIP: надто довгий промпт буде мовчки обрізаний, якщо інтерфейс не надає явного попередження [5, 13]. З цієї причини рекомендується реалізувати у застосунку діалог скорочення промпту з відображенням його реальної довжини у токенах це запобігає несподіваним результатам під час демонстрації та захисту роботи.

До типових артефактів, що виникають при генерації зображень, належать неправдоподібно відтворені пальці рук, "пластиковий" вигляд шкіри, поява зайвих об'єктів та нечитабельні написи. Використання негативного промпту

дозволяє частково придушити найбільш поширені з них, проте не замінює більш точних інструментів контролю таких як ControlNet чи подальшої ретуші засобами постобробки.

Важливо розуміти, що генеративні моделі відображають упередження навчальних даних. Це не помилка конкретного рядка коду, а системна властивість статистичного навчання на великих наборах зображень. Крім того, слід враховувати можливість зловживань: генерація діпфейків та відтворення чужих художніх стилів без дозволу авторів є реальними ризиками. Локально розгорнуті ваги моделей зазвичай не мають вбудованих фільтрів контенту, тому відповідальність за дотримання етичних і правових норм покладається на користувача. Типові метрики оцінювання генеративних моделей та їх інтерпретацію наведено в таблиці 1.3

Таблиця 1.3 – Типові метрики та їхня інтерпретація

Метрика	Що відображає	Застосування у T2I
FID	Відстань розподілів ознак	Порівняння моделей на фіксованій вибірці
IS	Різноманітність і якість класів	Допоміжний індикатор для класифікаційних ознак
CLIPScore	Узгодженість текст–зображення	Швидка перевірка відповідності промпту

Наведені метрики відповідають загальноприйнятій практиці оцінювання генеративних моделей. Окремої уваги також заслуговує дотримання обмежень токенізатора, оскільки перевищення ліміту призводить до автоматичного скорочення промпту без явного попередження користувача.

Окрім стандартних кількісних метрик, в індустрії активно застосовується метрика LPIPS (Learned Perceptual Image Patch Similarity), яка оцінює

перцептивну подібність зображень на рівні ознак навченої нейронної мережі, а не попіксельну різницю. На відміну від PSNR та SSIM, LPIPS краще корелює із суб'єктивними оцінками якості людиною, що робить її корисним доповненням при порівнянні різних налаштувань семплера на фіксованому промпті. Разом із тим слід пам'ятати, що жодна автоматична метрика не замінює безпосередню перевірку відповідності зображення заявленому текстовому завданню з урахуванням специфіки конкретного застосування. Узагальнену багаторівневу архітектуру застосунку та цикл взаємодії користувача з підсистемами наведено на рисунку 1.3

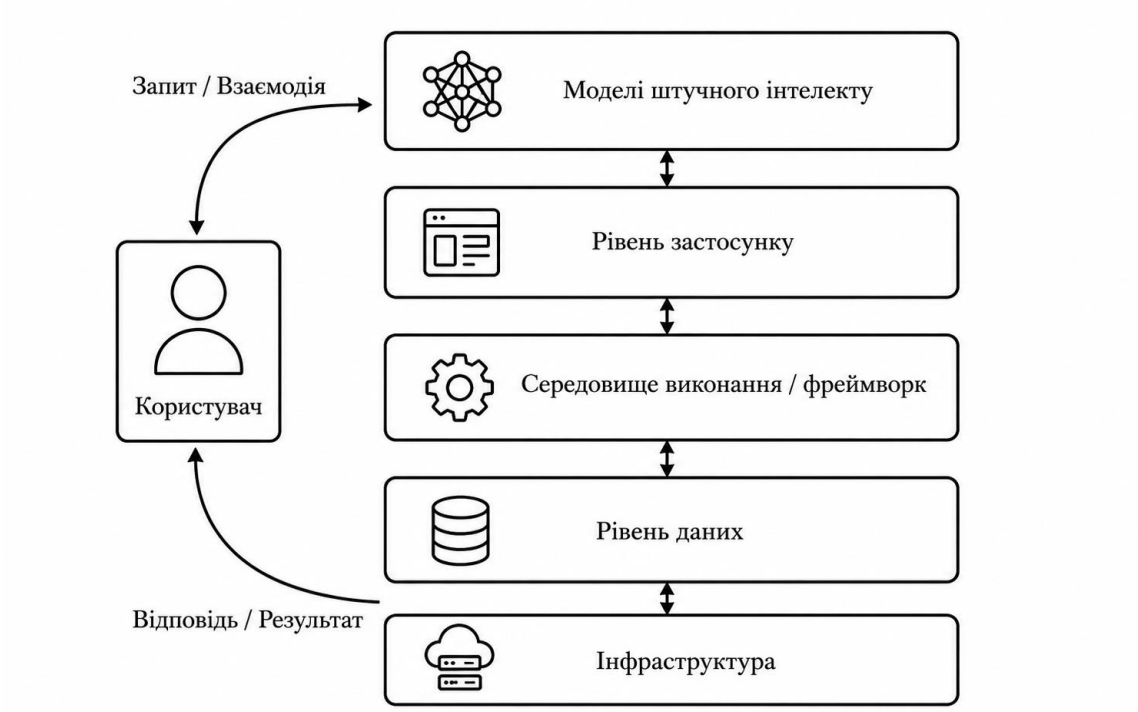

Рисунок 1.3 – Узагальнена багаторівнева архітектура застосунку і цикл взаємодії з користувачем

Явище галюцинацій у генерації зображень проявляється інакше, ніж у мовних моделях: замість вигаданих фактів модель може додавати зайві анатомічні елементи, спотворювати текстові написи на зображенні або неправильно відтворювати просторові відношення між об'єктами. Використання

негативного промпту для виключення типових артефактів є стандартною практикою, проте потребує дослідницького налаштування для кожного класу сцен. Надто агресивний негативний промпт може ненавмисно придушити бажані характеристики зображення, тому доцільно використовувати мінімально необхідний набір заборонених токенів.

Важливо також враховувати, що самі метрики FID і CLIPScore розраховуються на великих вибірках і не є показовими для оцінювання якості одиночного зображення.

2 АЛГОРИТМИ ГЕНЕРУВАННЯ ЗОБРАЖЕНЬ І ОБРОБКИ ЗАПИТІВ

2.1 Узагальнений алгоритм дифузійного перетвоерння «текст-у-зображення»

З інженерної точки зору процес інференсу латентної дифузії описується послідовним ланцюжком перетворень: текстовий запит перетворюється на умовні ознаки, на їх основі формується початковий шум у латентному просторі, після чого виконується багатокроковий денойзинг, а декодер VAE відновлює фінальне піксельне зображення. На практиці реалізація цього процесу потребує додаткових рішень: вибору семплера, налаштування guidance scale та застосування технік економії відеопам’яті (VRAM) саме ці чинники здебільшого визначають реальний час відгуку на конкретному обладнанні.

Текстовий енкодер формує векторне представлення промпту, яке на кожному кроці денойзингу змішується з активаціями UNet через механізм крос-уваги. Початковий шум генерується псевдовипадково з контрольованим seed-значенням, що дає змогу балансувати між детермінованістю результатів та їх варіативністю при повторних запусках.

На кожній ітерації денойзингу модель оновлює оцінку шуму або еквівалентну поправку стану конкретна формула залежить від обраного семплера. Метод Classifier-free Guidance (CFG) змішує умовне та безумовне передбачення з вагою $w > 1$: збільшення цього коефіцієнта посилює відповідність результату текстовому запиту, однак одночасно підвищує ймовірність появи артефактів пересичення [22].

Після завершення траєкторії денойзингу декодер VAE перетворює латентне представлення у піксельне зображення; далі виконується нормалізація діапазону значень і запис файлу на диск. Важливо зазначити, що процес інференсу не змінює ваги моделі, проте версія бібліотеки CUDA, версія драйвера

відеокарти та фрагментація відеопам’яті здатні призвести до збою навіть на тій самій моделі та обладнанні [27, 28]. Інференс text-to-image з розвилкою локального й хмарного виконання та спільним латентним циклом наведено на рисунку 2.1

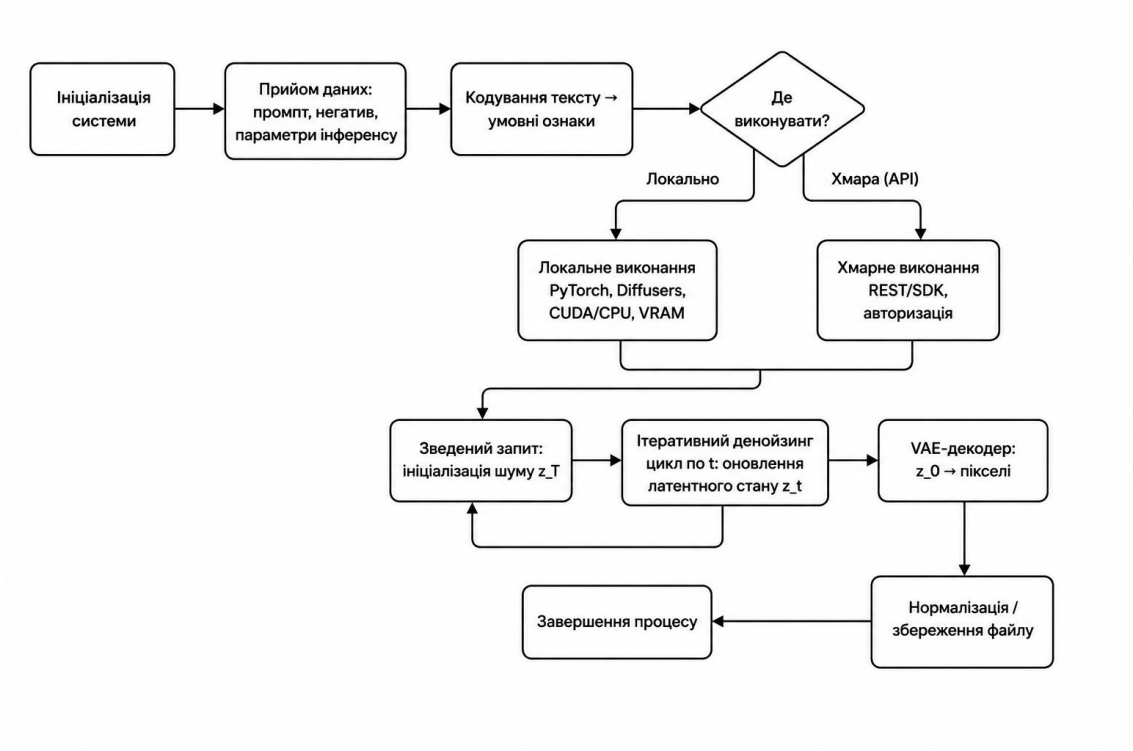

Рисунок 2.1 – Інференс text-to-image з розвилкою локального й хмарного виконання та спільним латентним циклом

Блок-схема узгоджує локальну й хмарну гілки зі спільним етапом формування латенту та декодування; далі параметри інференсу конкретизуються у табличному вигляді для зв’язку з налаштуваннями користувача.

У разі використання хмарного API внутрішні кроки денойзингу приховані за інтерфейсом сервісу, проте розуміння наведеного логічного ланцюжка залишається практично корисним: воно дозволяє користувачу чітко усвідомлювати, які параметри він реально контролює через інтерфейс застосунку, а які делегуються провайдеру. До найбільш поширених причин збоїв

належать: нестача відеопам’яті (OOM), відсутність підтримки CUDA у поточній збірці PyTorch, а також прострочені або недійсні API-ключі. Інтерфейс застосунку повинен надавати зрозуміле пояснення кожної з цих ситуацій і в жодному разі не знищувати незбережену роботу користувача. Вплив основних параметрів інференсу на якість і час генерації наведено в таблиці 2.1

Таблиця 2.1 – Вплив параметрів інференсу

Параметр	Зміна в бік збільшення	Типовий ефект
Кількість кроків	↑	Якість краща, час довший
Guidance scale	↑	Сильніша відповідність тексту; ризик артефактів
Роздільність	↑	Більші вимоги до VRAM і часу
Негативний промпт	Деталізація	Придушення небажаних артефактів

Залежність якості й часу від гіперпараметрів проявляється найвиразніше на етапі ітерацій семплера.

Вибір конкретного планувальника (scheduler) є одним із найбільш практично значущих рішень при налаштуванні пайплайна дифузійного інференсу. Бібліотека Diffusers надає уніфікований інтерфейс для взаємозамінного використання різних планувальників через атрибут pipeline.scheduler, що дозволяє протестувати кілька варіантів без зміни решти коду пайплайна. Euler Ancestral зберігає стохастичність між кроками, що збільшує варіативність результатів при однаковому промпті; DPM++ 2M Karras із застосуванням schedule Karras забезпечує добру якість вже за 15-20 кроків, що є типовим компромісом для інтерактивних застосунків.

Під семплером у програмній реалізації розуміють конкретну схему дискретизації процесу денойзингу: на кожному кроці модель оновлює оцінку

шуму за правилами обраного алгоритму. Крок ітерації семплера з паралельними передбаченнями для CFG та циклом до завершення наведено на рисунку 2.2

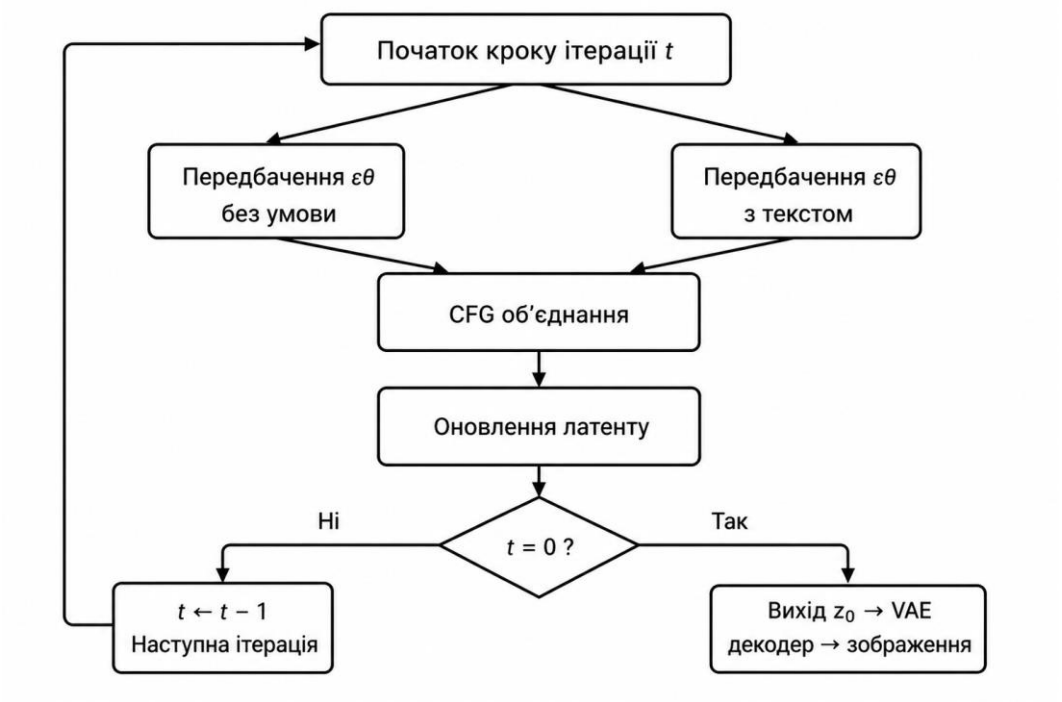

Рисунок 2.2 – Крок ітерації семплера з паралельними передбаченнями для CFG та циклом до завершення

Параметр eta (η) контролює рівень стохастики у методах DDIM та ряді похідних планувальників: при η = 0 траєкторія є повністю детермінованою і відтворюваною за фіксованого seed, при η = 1 еквівалентна повністю стохастичному процесу, як у DDPM. У більшості практичних сценаріїв рекомендується залишати η = 0 для відтворюваних демонстрацій і зберігати seed у метаданих зображення або запису журналу генерації. Ця інформація є частиною документації умов відтворення результатів, обов'язкової для академічних звітів.

Механізм VAE-encode/decode є вузьким місцем з точки зору якості для зображень із дрібними деталями, такими як текстові написи та регулярні текстури. Деякі спільнотні проєкти пропонують альтернативні VAE з

покращеними характеристиками для специфічних доменів, сумісні із стандартними архітектурами SD. При заміні VAE необхідно перевіряти масштабний коефіцієнт (scaling factor) і відповідність очікуваного розподілу латентних значень, оскільки неправильно підібраний VAE дасть розмиті або кольорово спотворені результати навіть при ідеальній роботі UNet.

2.2 Алгоритм формування рекомендацій для текстових запитів

У розробленому застосунку текстовий промпт проходить чітко визначений конвеєр підготовки. На першому етапі виконується нормалізація введеного тексту та дедуплікація тегів, розділених комами. Далі, за наявності відповідних записів у базі даних, до промпту додаються опційні стилістичні шаблони з SQLite. На заключному етапі якщо така можливість активована в налаштуваннях виконується звернення до мовної моделі (LLM), яка повертає компактний рядок англомовних тегів без зайвих пояснень у відповіді.

Вибір постачальника LLM здійснюється через змінні середовища: для автономного режиму роботи використовується Ollama, для онлайн-режиму комерційні API за наявності відповідного бюджету [16, 29]. Незалежно від обраного маршруту, результат роботи LLM має бути зведений до єдиного стандартизованого формату перед передачею в генератор зображень це забезпечує сумісність з усіма підтримуваними модулями генерації.

Окремі хмарні API очікують отримати єдиний об’єднаний текстовий опис із явним блоком “уникати” замість класичного параметра negative prompt. Це є технічною особливістю конкретного контракту API, а не зміною наміру користувача, тому відповідна адаптація формату виконується автоматично в модулі підготовки запиту.

Гілка обробки промпта для Stable Diffusion 1.5 потребує обов’язкової перевірки довжини у токенах CLIP; для моделей SD 3.x встановлені інші ліміти, тому у коді передбачено явне розгалуження залежно від версії моделі [25]. Це необхідно для того, щоб уникнути зайвого скорочення промпту там, де токенний ліміт є більшим. Запис до журналу виконується після фіксації фінальних рядків, що реально були передані в модель це дозволяє точно відтворити вдалий сеанс під час перевірки або захисту роботи. З міркувань безпеки, конфіденційні дані та API-ключі не виводяться у відкриті лог-файли; файл config.env зберігається поза репозиторієм git, а у консоль виводиться лише технічний traceback без повних тіл HTTP-відповідей. Режими покращення промпта та їх характеристики наведено в таблиці 2.2.

Таблиця 2.2 – Режими покращення промпта

Режим	Джерело	Результат
local_only	Евристики й пресети	Теги без мережі
google / groq / ollama	LLM API	Один рядок англомовних тегів
Шаблони SQLite	БД користувача	Суфікси позитиву / негативу
SD 1.5	CLIP tokenizer	Укорочення до безпечної довжини

Дедуплікація тегів є технічно простим, але практично важливим кроком конвеєра підготовки промпта. Реалізація через множину (set) Python із подальшим з'єднанням через кому не зберігає оригінального порядку токенів, що може впливати на результат генерації, оскільки CLIP-токенізатор надає певний пріоритет токенам, що з'являються раніше у послідовності. Альтернативним підходом є використання словника зі збереженням порядку вставки (dict.fromkeys()), що дозволяє усунути дублікати, зберігаючи при цьому послідовність першої появи кожного тегу. Таке рішення є більш передбачуваним

з точки зору відтворюваності результатів. Перелічені режими задають альтернативні гілки одного й того самого конвеєра підготовки тексту; набір аргументів на вході алгоритму узгоджується з таблицею 2.3.

Таблиця 2.3 – Вхідні дані алгоритму підготовки промпта

Елемент	Призначення
Текст позитивного промпта	Базовий опис сцени
Текст негативного промпта	Обмеження й заборони
Обрані шаблони	Додаткові стилістичні блоки
Модель зображення	Визначає постобробку рядків

Важливою деталлю реалізації LLM-розширення є формулювання системного промпту для допоміжної мовної моделі. Якщо системний промпт занадто вільний, LLM може повернути пояснення, поради або нумеровані списки замість очікуваного компактного рядка тегів. Надійнішим підходом є включення до системного промпту явного прикладу форматів введення та виведення (few-shot приклади), а також перевірка відповіді на відповідність очікуваному шаблону регулярним виразом перед її використанням. У разі невідповідності система повинна або повторити запит із більш суворим форматуванням, або повернутися до базового промпту без LLM-розширення.

Журналювання фінальних рядків промптів, що реально передавалися в модель, є критично важливим для забезпечення відтворюваності результатів. Зокрема, якщо LLM-розширення або дедуплікація суттєво змінили вхідний текст, запис у журналі повинен відображати саме ці фінальні рядки, а не оригінальний ввід користувача. Опціонально можна зберігати обидва варіанти оригінальний ввід у полі raw_prompt і фінальний рядок у полі prompt що

дозволить у майбутньому проаналізувати, наскільки суттєво LLM-обробка змінила намір користувача.

2.3 Структура бази даних шаблонів і історії генерацій

Файл бази даних data/app.db створюється автоматично при першому зверненні до відповідного модуля. Початкова версія застосунку обмежувалася двома таблицями шаблонів і журналу генерацій без використання зовнішніх ключів.

Таблиця models зберігає унікальний ідентифікатор model_key та тип підключення (локальний або конкретний хмарний API). Таблиця generation_presets містить збережені пресети параметрів інференсу: розміри зображення, кількість кроків, значення guidance scale та назву семплера. Поле preset_id у таблиці журналу є необов'язковим воно залишається порожнім, якщо користувач не обирав пресет. Таблиця prompt_templates виконує роль довідника текстових фрагментів суфіксів позитивного та негативного промптів.

Таблиця generation_history посилається на models через поле model_id, а також опційно на generation_presets і prompt_templates. У полях prompt та negative зберігаються підсумкові рядки, що реально передавалися в модель після застосування шаблонів і LLM-обробки. Поле saved_path вказує на збережений файл зображення, а поле status фіксує результат спроби генерації.

Зв'язок "багато тегів багато записів журналу" реалізовано через таблицю tags та проміжну таблицю generation_tags це класичне вирішення відношення M:N без дублювання назв тегів у кожному записі. Таблиця image_variants дозволяє зберігати кілька файлів для одного запису журналу (наприклад, повне PNG-зображення та мініатюра), не змішуючи їх шляхи в одному полі. ER-модель

бази даних SQLite із сімома таблицями, зовнішніми ключами та зв'язком M:N через generation_tags наведено на рисунку 2.3.

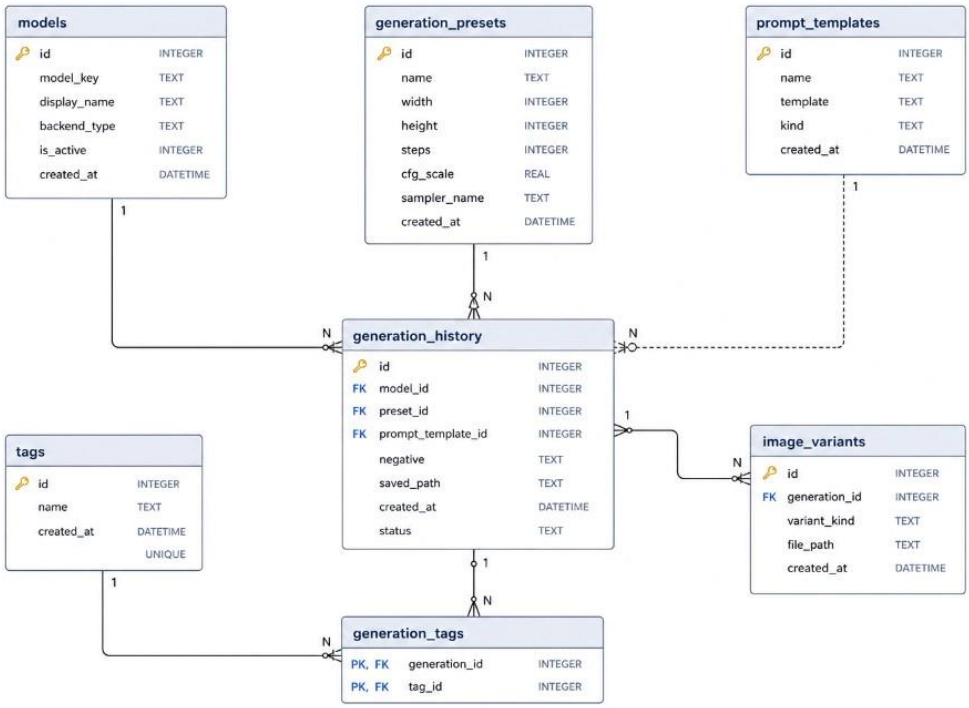

Рисунок 2.3 – ER-модель бази даних SQLite: сім таблиць, зовнішні ключі та зв’язок M:N через generation_tags

Міграція зі старішої версії схеми виконується автоматично: якщо в таблиці generation_history виявлено застаріле поле model_key, таблиця перейменовується, створюється нова структура, а наявні рядки переносяться з автоматичним створенням відповідних записів у таблиці models та записів у image_variants з типом kind=full за наявності шляху до файлу. Директива PRAGMA foreign_keys=ON активується при кожному підключенні до бази, щоб SQLite виконував перевірку цілісності посилань [18]. Перед оновленням версії програми на спільному комп’ютері рекомендується створювати резервну копію файлу app.db. Оскільки промпти можуть містити персональні дані, після

публічної демонстрації журнал слід очищати або зберігати базу на зашифрованому носії.

Діаграма відповідає програмній схемі: зовнішні ключі вмикаються через PRAGMA foreign_keys=ON, проміжна таблиця generation_tags реалізує зв’язок багато-до-багатьох між журналом і тегами. Поле prompt_template_id в журналі опційне воно дозволяє зафіксувати шаблон як підказку навіть після ручного редагування промпта. Основні таблиці бази даних SQLite із ключовими полями та призначенням наведено в таблиці 2.4.

Таблиця 2.4 – Основні таблиці SQLite прикладної бази даних

Таблиця	Ключові поля	Призначення
models	id, model_key, backend_type, is_active	Довідник моделей і типів підключення
generation_presets	id, name, width, height, steps, cfg_scale, sampler_name	Збережені пресети інференсу
prompt_templates	id, name, template, kind	Шаблони суфіксів і негативних блоків
generation_history	model_id, preset_id, prompt_template_id, prompt, negative, saved_path, status	Журнал успішних генерацій
tags	id, name	Словник міток
generation_tags	generation_id, tag_id	Зв’язок M:N між журналом і тегами
image_variants	generation_id, variant_kind, file_path	Файли виходу (повний, прев’ю тощо)

Проєктування схеми бази даних із явними зовнішніми ключами потребує розуміння поведінки SQLite при каскадному видаленні. За замовчуванням SQLite не виконує каскадне видалення навіть за наявності відповідних

REFERENCES-обмежень, якщо не ввімкнено PRAGMA foreign_keys=ON. Це означає, що видалення запису моделі без попереднього видалення пов'язаних записів у generation_history може порушити цілісність даних. Рекомендується або реалізувати явне каскадне видалення через ON DELETE CASCADE у визначенні FK, або виконувати видалення пов'язаних записів у межах однієї транзакції через код застосунку.

Набір полів покриває довідники, журнал і похідні файли; при збереженні зображення транзакція додає рядок історії, а таблиця image_variants фіксує фактичні шляхи (типово variant_kind=full). Подальші кроки журналювання зведені в таблиці 2.5.

Таблиця 2.5 – Етапи запису історії генерації

Крок	Дія
1	Успішне завершення пайплайна зображення
2	Формування підсумкових рядків промптів
3	Збереження файлу у каталог користувача
4	INSERT у generation_history та image_variants

Для індексування великого журналу генерацій корисно визначити складний індекс за полями model_id та часовою міткою created_at, що пришвидшить типовий запит фільтрування "останні N генерацій для конкретної моделі". Хоча для однокористувацького застосунку розмір журналу рідко досягає сотень тисяч записів, добра звичка визначення індексів з самого початку спрощує можливе масштабування та не несе суттєвих витрат при малому обсязі даних. Фіксація в журналі виконується лише після підтвердженого збереження файлу зображення, що узгоджує файлову систему з транзакцією SQLite. Вибір числової схеми семплера визначає компроміс між числом кроків і стійкістю траєкторії денойзингу; узагальнена характеристика наведена в таблиці 2.6.

Таблиця 2.6 – Орієнтовна поведінка типів семплерів у Diffusers

Семплер / родина	Типові кроки T	Характеристика й типові умови застосування
Euler / Euler a	20–40	Швидкі чернетки; ancestral додає стохастику до траєкторії
Heun / midpoint	Більше ніж у Euler	Гладші траєкторії при вищій обчислювальній вартості
DPM++ 2M Karras	15–30	Частий компроміс якості й числа кроків для SD
UniPC / DEIS	Низькі T можливі	Схеми залежать від версії бібліотеки diffusers

Рекомендується також реалізувати простий механізм версіонування схеми через допоміжну таблицю schema_version із єдиним рядком, що містить номер поточної версії схеми. При запуску застосунку процедура ініціалізації порівнює збережену версію з очікуваною і виконує лише необхідні міграційні кроки, не намагаючись повторно застосувати вже виконані зміни. Такий підхід є стандартом у промисловій розробці і робить підтримку схеми значно прозорішою при подальших модифікаціях застосунку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЮ ГЕНЕРУВАННЯ ХУДОЖНІХ ЗОБРАЖЕНЬ ЗА ТЕКСТОВИМ ОПИСОМ

3.1 Середовище розробки і конфігурація

Цільовою платформою застосунку є 64-бітна операційна система Windows. Вибір Python версії 3.10 і вище обумовлений сумісністю з методичними вимогами кафедри та наявністю актуальних бінарних пакетів PyTorch для цієї версії інтерпретатора. Перед встановленням залежностей у PowerShell активується ізольоване віртуальне середовище venv із каталогу проєкту командою Scripts\Activate.ps1.

Файл requirements.txt фіксує мінімально необхідні версії бібліотек для графічного інтерфейсу, пакетів diffusers і transformers, а також HTTP-клієнтів для хмарних API [11, 12]. За наявності GPU від NVIDIA збірку PyTorch підбирають відповідно до встановленого драйвера і перевіряють підтримку через виклик torch.cuda.is_available() на цільовій машині. Якщо відеокарта відсутня або не підтримується, застосунок коректно переходить у режим CPU з попереджувальним повідомленням для користувача [10, 27, 28].

Конфіденційні дані та API-ключі зберігаються у файлі config.env, який завантажується автоматично при запуску застосунку. Зміна налаштувань доступна через вбудований діалог без необхідності відкривати файл вручну. Файл конфігурації виключено з відстеження системою контролю версій git для запобігання випадковому розкриттю секретів.

Змінна середовища APP_THEME перемикає між світлою та темною темою CustomTkinter суто на рівні відображення, не впливаючи на логіку генерації зображень. Це забезпечує зручну можливість підготовки скріншотів у різних темах для звіту без жодних змін у коді. З міркувань інформаційної безпеки на навчальних комп’ютерах повні тіла відповідей API не виводяться у текстові лог-

файли: в інтерфейсі відображається лише короткий код помилки та зрозуміле повідомлення, тоді як деталі залишаються доступними у консолі розробника. Компонування головного вікна застосунку наведено на рисунку 3.1.

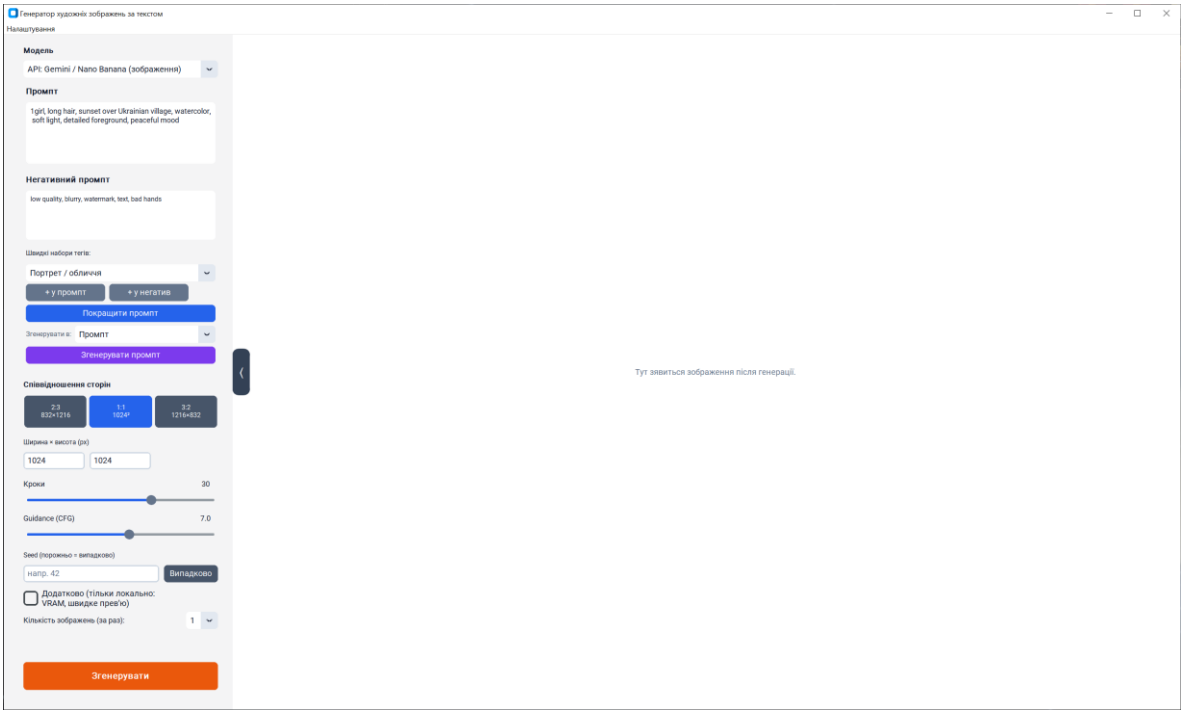

Рисунок 3.1 – Компонування головного вікна

Практика ізоляції середовища через venv має бути доповнена явним фіксуванням версій пакетів у requirements.txt з точними номерами, а не діапазонами (наприклад, diffusers==0.27.2 замість diffusers>=0.25.0). Використання розмитих обмежень версій призводить до ситуацій, коли застосунок, що коректно працював у момент розробки, перестає запускатися після оновлення залежностей на іншому комп'ютері. Генерація requirements.txt через pip freeze після повного тестування середовища фіксує точне середовище і є стандартною практикою для відтворюваних розробницьких середовищ.

Конфігураційний файл config.env у поєднанні з пакетом python-dotenv дозволяє централізовано керувати всіма чутливими параметрами. Важливо, щоб застосунок коректно запускався і при відсутності деяких опціональних ключів,

виводячи зрозуміле попередження замість виключення KeyError. Наприклад, якщо ключ STABILITY_API_KEY відсутній, відповідний хмарний модуль генерації повинен бути автоматично відключений в інтерфейсі, а не призводити до збою при спробі ініціалізації. Такий підхід до конфігурації відповідає принципу fail-safe за замовчуванням і суттєво спрощує першу настройку застосунку.

Підтримка змінної APP_THEME для перемикання між темами CustomTkinter є прикладом clean separation between presentation and logic розділення рівня відображення та рівня бізнес-логіки. Логіка генерації зображень, підготовки промптів і взаємодії з базою даних не повинна містити жодних посилань на поточну тему, а рівень інтерфейсу на деталі конкретних пайплайнів. Дотримання цього принципу спрощує написання автоматизованих тестів для бізнес-логіки, оскільки вони можуть виконуватися без ініціалізації графічного інтерфейсу.

3.2 Архітектура системи та основні модулі

Архітектура застосунку побудована на принципі розділення відповідальності. Модуль src/ui/app.py відповідає за компонування головного вікна, обробку подій інтерфейсу та виконання тривалих операцій поза головним потоком Tkinter. Модулі prompt_enhance.py та text_enhance_providers.py реалізують всю логіку підготовки та покращення текстових запитів. Каталог src/generators містить окремі реалізації для локального виклику дифузійних моделей і звернення до хмарних API.

Модуль database.py відповідає за створення та міграцію схеми SQLite, надає CRUD-операції для роботи з шаблонами і реалізує функцію log_generation,

яка в межах однієї транзакції створює запис журналу, фіксує варіанти вихідних файлів і за потреби додає теги [18]. Такий підхід ізолює інтерфейсний рівень від деталей визначення схеми бази даних.

Обробка помилок реалізована з їх класифікацією за типом: мережеві помилки, нестача відеопам’яті та некоректний ввід користувача. У кожному випадку інтерфейс відображає зрозуміле пояснення українською мовою, тоді як повний текст виключення залишається доступним у консолі для налагодження. Виклики PyTorch і HTTP-запити не виконуються безпосередньо в обробниках подій кнопок результати повертаються до головного потоку через черги або механізм after(), що запобігає заморожуванню інтерфейсу під час тривалих операцій.

У документації до проєкту рекомендується фіксувати перевірену сумісну пару: версію бібліотеки diffusers та ідентифікатор моделі на Hugging Face Hub. Оновлення залежностей іноді змінює назви аргументів пайплайна без відповідного попередження в офіційних прикладах, що може призвести до непередбаченої несумісності при відтворенні результатів [11]. Основні модулі та файли проєкту із зазначенням їх призначення наведено в таблиці 3.1.

Таблиця 3.1 – Основні модулі та файли проєкту

Компонент	Файл / призначення
Запуск	main.py
Інтерфейс	src/ui/app.py
Локальна дифузія	src/generators/local_sd.py
Сталість даних	src/database.py
Промпти	src/prompt_enhance.py
LLM-провайдери	src/text_enhance_providers.py
Хмарні сервіси	stability_api.py, gemini_backend.py, openrouter_image.py

Узгодженість імен файлів і функціональних ролей спрощує супровід при оновленні залежностей і заміні моделей на Hugging Face Hub. Вимоги до апаратного забезпечення для локального інференсу зведені в таблиці 3.2 з акцентом на обмеження пам’яті.

Таблиця 3.2 – Орієнтовні конфігурації ПК для локального інференсу

Клас ПК	VRAM / RAM	Очікувана модель	Обмеження середовища
Без дискретної GPU	— / ≥16 ГБ	Хмара або демонстрація на CPU	Локальна SD на CPU: час інференсу на порядки вищий
Entry GPU	6–8 ГБ VRAM	SD 1.5 знижена роздільність	VRAM: типові OOM без зменшення розміру або offload
Mid-range	10–12 ГБ VRAM	SD 1.5; SDXL обмежено	SDXL потребує нижчої роздільності або економії пам’яті
Upper mid	≥16 ГБ VRAM	SD3 з техніками offload	Пік споживання VRAM у диспетчері задач

Фрагмент перевірки обмеження CLIP для локальної SD 1.5 у модулі prompt_enhance:

```
def clip_token_safe_text(text: str, *, max_length: int = 77) -
> tuple[str, bool]:
    ...
    tokenizer = CLIPTokenizer.from_pretrained("openai/clip-
vit-large-patch14")
    ids = tokenizer(t, add_special_tokens=True,
truncation=False)["input_ids"]
    if len(ids) <= max_length:
        return t, False
    enc = tokenizer(t, add_special_tokens=True,
```

```
truncation=True, max_length=max_length)
    shortened = tokenizer.decode(enc["input_ids"],
skip_special_tokens=True).strip()
    return shortened, True
```

Взаємодія між потоком інтерфейсу та фоновим потоком генерації реалізується через стандартний механізм черг Python queue.Queue або через метод after() об'єкта Tkinter для планування відновлення стану кнопок і відображення результату в головному потоці. Виклики методів tkinter-віджетів безпосередньо з фонового потоку є небезпечними і можуть призводити до непередбачуваних помилок або зависання інтерфейсу, оскільки Tkinter не є потокобезпечним. Правильна реалізація цього шаблону є однією з найбільш поширених технічних складнощів при розробці GUI-застосунків на Python. Вікно налаштувань застосунку наведено на рисунку 3.2.

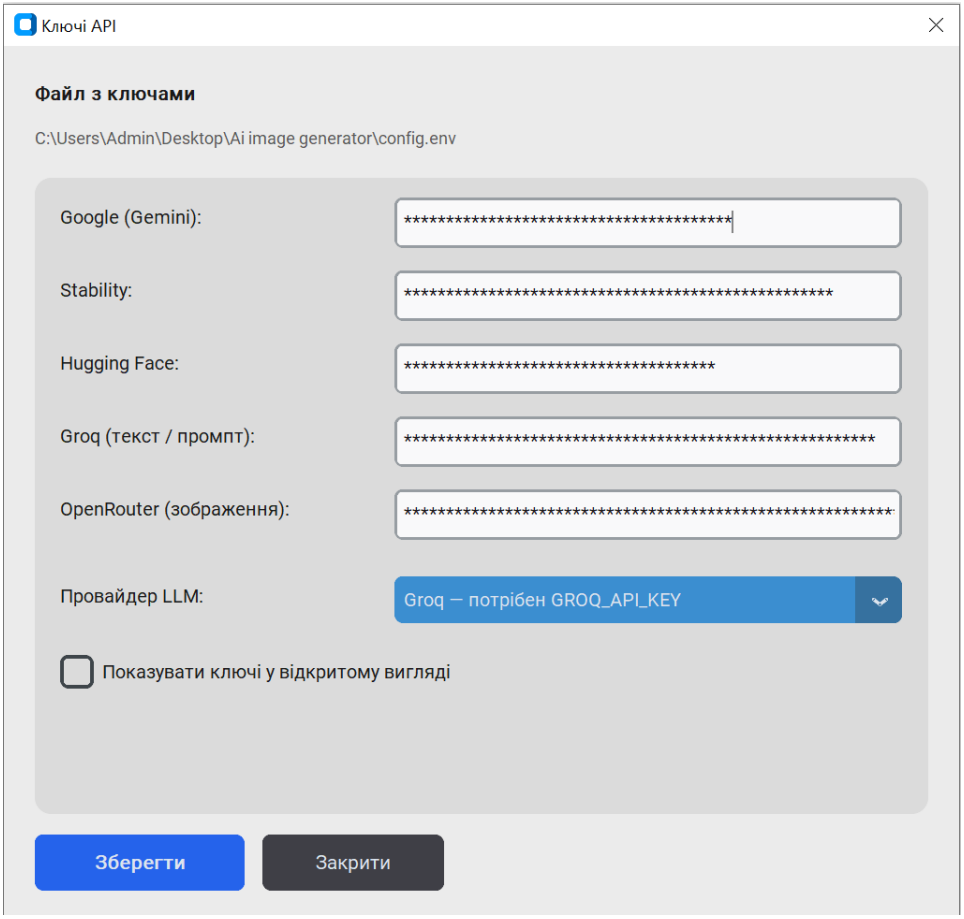

Рисунок 3.2 – Вікно налаштувань

Модульна структура кодової бази суттєво впливає на зручність майбутніх змін. Зокрема, введення абстрактного базового класу BaseGenerator із єдиним методом generate(prompt, negative_prompt, params) дозволяє уніфікувати інтерфейс між локальним і хмарними генераторами. Інтерфейсний код тоді не містить розгалужень за типом підключення, а лише отримує поточний екземпляр генератора з реєстру та викликає generate(). Це робить додавання нового модуля генерації, наприклад нового хмарного API, зводить лише до реалізації нового підкласу без змін в інтерфейсному рівні.

Логування на рівні модулів із використанням стандартного пакету logging Python є кращою практикою порівняно з прямими викликами print(). Ієрархічна система логерів дозволяє налаштувати різні рівні деталізації для різних підсистем: наприклад, увімкнути DEBUG-логи лише для модуля database.py під час відлагодження проблем із транзакціями, зберігаючи лаконічний вивід для решти модулів. Конфігурація логування через dictConfig у момент запуску застосунку дозволяє легко перемикатися між режимами розробки і продакшн без зміни коду самих модулів.

3.3 Сценарії використання й тестування

Тестування застосунку проводилося за сценарним підходом без навантажувального стрес-тестування, що відповідає масштабу однокористувацького клієнта. Для гілки локальної генерації на GPU перевірявся повний цикл роботи SD 1.5, а також реакція системи при штучно знижених параметрах роздільності метою було отримати коректне інформаційне повідомлення про нестачу пам’яті замість аварійного завершення без пояснень.

Гілку Stable Diffusion 3 за наявності токена HF_TOKEN тестували щонайменше на одній машині з обмеженим обсягом VRAM та активованими механізмами економії пам’яті. Хмарні режими роботи перевірялися на коректність формування HTTP-запитів, а також на належну поведінку при отриманні кодів відповіді 401 (помилка автентифікації) та 429 (перевищення ліміту запитів). Приклад протоколу тестування застосунку із зазначенням очікуваних результатів наведено в таблиці 3.3.

Таблиця 3.3 – Приклад протоколу тестів

ID	Сценарій	Очікуваний результат	Об’єкт перевірки
T1	Порожній промпт	Блокування генерації або явне попередження	Модуль валідації полів інтерфейсу
T2	Довгий текст для SD 1.5	Укорочення до ліміту токенів і прапорець у UI	Функція clip_token_safe_text
T3	Два шаблони типу suffix	Об’єднаний рядок без повторів тегів	merge_comma_dedup, SQLite шаблони
T4	Успішна генерація зі збереженням	Рядок у generation_history і запис у image_variants	log_generation після підтвердженого шляху файлу

Функціонал шаблонів перевірявся на коректність дедуплікації тегів: застосування двох суфіксів із однаковими тегами, що перетинаються, повинно давати кожен тег рівно один раз у результуючому рядку. Гілка розширення промпту через LLM за наявності API-ключа перевірялась на виконання вимоги “один рядок тегів без додаткових пояснень”. Протокол кожного тесту рекомендується оформлювати у форматі “очікуваний результат / фактичний

результат” із зазначенням версії Python, CUDA та моделі відеокарти це є стандартною вимогою до відтворюваності результатів на кафедрі

Тестування обробки помилок є не менш важливим, ніж тестування "щасливих" шляхів виконання. Симуляцію OOM-помилки на GPU можна виконати через явне виділення великого тензора перед запуском пайплайна або через налаштування штучно зниженого ліміту пам'яті. Перевірка реакції на HTTP-код 401 від хмарного API виконується через підстановку навмисно некоректного API-ключа в конфігурації тестового середовища. У кожному випадку очікуваним результатом є зрозуміле повідомлення для користувача та збереження цілісності стану застосунку, а не аварійне завершення процесу.

Регресійні тести, що перевіряють коректність дедуплікації тегів та роботу конвеєра промпта без мережевого з'єднання (у режимі local_only), можуть бути виконані у будь-якому середовищі без GPU і без реальних API-ключів. Такі тести є найшвидшим і найдешевшим способом впевнитися, що модифікація конвеєра підготовки промпта не порушила існуючу логіку. Рекомендується виконувати їх автоматично перед кожним коміт-ом через простий скрипт або за допомогою pytest без додаткових залежностей від GPU-бібліотек.

3.4 Результати роботи застосунку

Після завершення генерації користувач отримує зображення через стандартний діалог збереження файлу або у типовий каталог output. Бібліотека Pillow забезпечує підтримку кількох форматів експорту [15]. Попередній перегляд у головному вікні масштабується із використанням методу letterbox, що запобігає спотворенню пропорцій зображення і дозволяє одразу оцінити результат без відкриття зовнішнього переглядача.

Підтримка світлої та темної теми CustomTkinter демонструє гнучкість інтерфейсного рівня [14]. Порівняльний аналіз “локально проти хмари” на одному й тому самому промпті з фіксацією часу виконання секундоміром забезпечує наочні дані без потреби у статистично значущій вибірці.

Серед напрямків подальшого розвитку застосунку варто виділити: реалізацію черги завдань для пакетної генерації, інтеграцію ControlNet та LoRA-адаптерів, а також агресивніше кешування завантажених ваг моделей для зменшення залежності від якості інтернет-з’єднання при першому запуску. Вікно перегляду шаблонів і журналу генерацій наведено на рисунку 3.3.

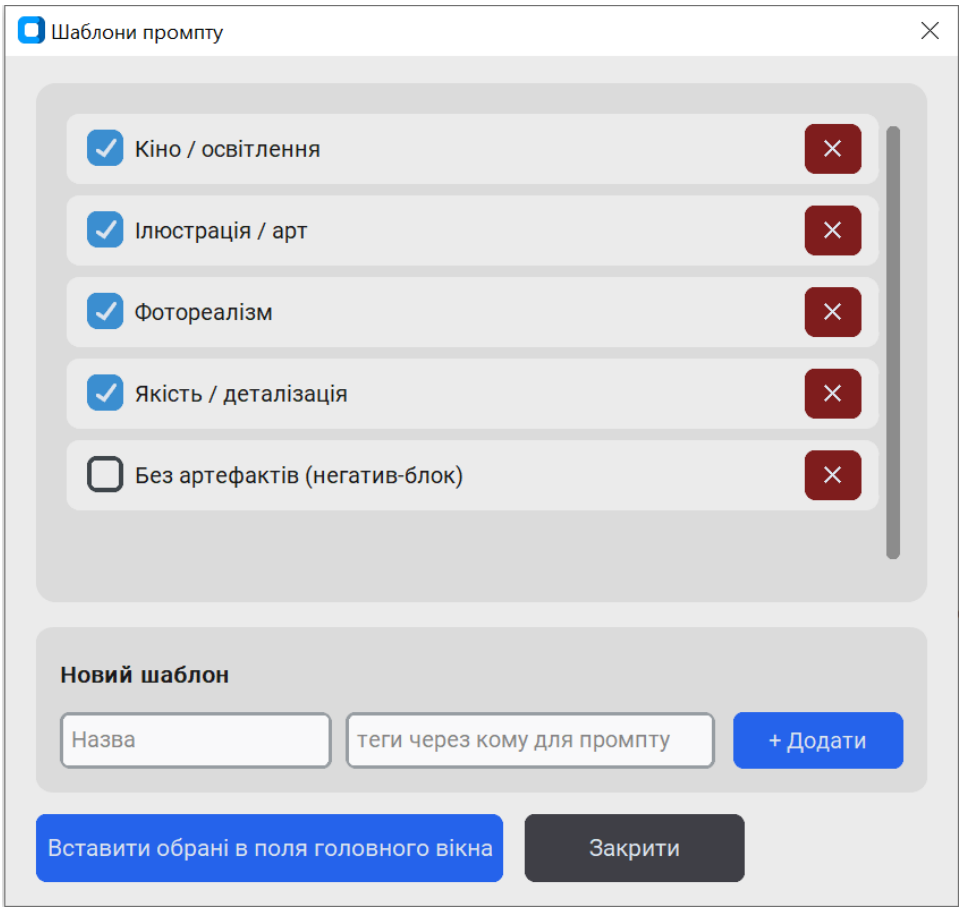

Рисунок 3.3 – Шаблони та журнал

Ініціалізація схеми бази даних зводиться до виконання DDL при старті застосунку:

```
def init_db(db_path: Optional[Path] = None) -> None:
    conn = get_connection(db_path)  # PRAGMA foreign_keys=ON
    try:
        conn.executescript(
            """
            CREATE TABLE IF NOT EXISTS models (...);
            CREATE TABLE IF NOT EXISTS generation_presets
(...);
            CREATE TABLE IF NOT EXISTS prompt_templates (...);
            CREATE TABLE IF NOT EXISTS generation_history (...
FK ...);
            CREATE TABLE IF NOT EXISTS tags (...);
            CREATE TABLE IF NOT EXISTS generation_tags (...);
            CREATE TABLE IF NOT EXISTS image_variants (...);
            """
        )
        # міграція зі старої generation_history(model_key) за
потреби
        ...
        conn.commit()
    finally:
        conn.close()
```

У локальному генераторі перед викликом пайплайна SD 1.5 підставляються безпечні для CLIP рядки:

```
if not is_sd3:
        p2, trunc_p = clip_token_safe_text(prompt)
        n2, trunc_n = clip_token_safe_text(negative or "")
        call_kw["prompt"] = p2
        call_kw["negative_prompt"] = n2 or None
```

Форматування попереднього перегляду через метод letterbox, що зберігає пропорції зображення, є важливим UX-рішенням, оскільки моделі типу SD 1.5 підтримують нестандартні співвідношення сторін (наприклад, 768×512 або 512×768 для портретних зображень). Спотворення пропорцій у вікні перегляду створює хибне враження про результат генерації і може змусити користувача повторити генерацію вважаючи результат невдалим. Коректне letterbox-

масштабування з нейтральним заливом фону та центруванням зображення є стандартним рішенням, що реалізується кількома рядками коду Pillow.

Підтримка різних форматів збереження через стандартний діалог Windows дозволяє користувачеві вибирати між PNG (без втрат, великий розмір), JPEG (з втратами, компактний) та WebP (без втрат або з втратами, компактний з кращою якістю). Доцільно зберігати у метаданих файлу або у паралельному текстовому файлі параметри генерації: промпт, seed, назву моделі, кількість кроків та guidance scale. Для PNG це можна реалізувати через вбудовані текстові метадані (tEXt або iTXt чанки), що підтримуються Pillow через атрибут pnginfo. Такий підхід дозволяє відновити точні умови генерації при перегляді зображення навіть без звернення до журналу бази даних.

Наступним практичним кроком розвитку застосунку після базової функціональності є реалізація підтримки ControlNet для уточнення просторового розташування об'єктів і збереження загальної композиції при варіаціях промпту. Інтеграція ControlNet вимагає додаткового кроку попередньої обробки вхідного зображення для отримання карти умов (pose, depth, canny тощо) і передачі її в пайплайн разом із текстовим промптом. Це суттєво розширює можливості застосунку для творчих завдань без значних змін у загальній архітектурі, оскільки бібліотека Diffusers надає уніфікований ControlNetPipeline-інтерфейс, сумісний із існуючим кодом локального генератора.

ВИСНОВКИ

У дипломній роботі вирішено актуальну науково–практичну задачу розроблено настільний програмний засіб генерації зображень за текстовим описом (text-to-image), що об’єднує локальний інференс і хмарні API, уніфіковану обробку промптів та збереження шаблонів і журналу генерацій під Microsoft Windows. За результатами виконаної роботи можна зробити такі висновки.

1. На основі аналізу предметної галузі встановлено, що комерційні T2I-сервіси та окремі відкриті моделі не забезпечують єдиного інструменту з одночасною підтримкою локального запуску, хмарних API і керування промптами з урахуванням конфіденційності та обмежень обладнання. Визначено, що власний настільний застосунок на Python і CustomTkinter дає баланс між функціональністю, простотою розгортання та використанням GPU користувача.

2. Здійснено порівняльний аналіз інструментального стеку: обґрунтовано вибір Python 3.10+, PyTorch і Hugging Face Diffusers для локального інференсу, CustomTkinter для інтерфейсу, SQLite для збереження даних та HTTP/SDK-клієнтів для інтеграції з Google Gemini, Stability AI і OpenRouter.

3. Спроєктовано та реалізовано модульну архітектуру системи: рівень графічного інтерфейсу з фоновим виконанням генерації, рівень обробки промптів і конфігурації, модулі локальної та хмарної генерації, локальна СУБД SQLite для шаблонів, пресетів і журналу генерацій.

4. Розроблено структуру бази даних SQLite із сімома таблицями: generation_history, models, prompt_templates, generation_presets, tags,

generation_tags та image_variants. Реалізовано автоматичну ініціалізацію схеми, міграцію та запис у журнал після збереження зображення.

5. Розроблено графічний інтерфейс застосунку для введення промптів, вибору бекенда, керування шаблонами, історією та налаштуваннями. Функціональне тестування підтвердило коректну роботу штатних сценаріїв, обробку помилок OOM і HTTP 401/429 та збереження цілісності стану програми.

6. Реалізовано повний цикл роботи користувача: підготовка промпта (евристики, шаблони, LLM), генерація локально або в хмарі, попередній перегляд і експорт у PNG, JPEG та WebP. Світла та темна теми CustomTkinter забезпечують зручну роботу на різних конфігураціях моніторів.

7. Розрахунки техніко–економічної частини показали, що собівартість розробки становить 15 489 грн, прогнозована ціна – 19 361 грн, а індекс прибутковості PI = 24,86 підтверджує економічну доцільність впровадження розробленого рішення.

Практична цінність роботи полягає у створенні готового настільного text-to-image клієнта для дизайнерів і контент-мейкерів, який поєднує локальний Stable Diffusion із хмарними альтернативами в одному інтерфейсі. Напрямком подальших досліджень є пакетна генерація, інтеграція LoRA і ControlNet, профілювання під різні GPU та шифрування config.env для машин спільного користування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ho J., Jain A., Abbeel P. Denoising Diffusion Probabilistic Models // Advances in Neural Information Processing Systems (NeurIPS). – 2020. – Vol. 33.

2. Sohl-Dickstein J. et al. Deep Unsupervised Learning using Nonequilibrium Thermodynamics // ICML. – 2015.

3. Rombach R. et al. High-Resolution Image Synthesis with Latent Diffusion Models // CVPR. – 2022.

4. Dhariwal P., Nichol A. Diffusion Models Beat GAN on Image Synthesis // NeurIPS. – 2021.

5. Radford A. et al. Learning Transferable Visual Models From Natural Language Supervision // arXiv:2103.00020. – 2021.

6. Saharia C. et al. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding // arXiv:2205.11487. – 2022.

7. Goodfellow I. et al. Generative Adversarial Networks // NeurIPS. – 2014.

8. Kingma D. P., Welling M. Auto-Encoding Variational Bayes // ICLR. – 2014.

9. Dosovitskiy A. et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale // ICLR. – 2021.

10. Paszke A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library // NeurIPS. – 2019.

11. Hugging Face. Diffusers documentation [Електронний ресурс]. – Режим доступу: https://huggingface.co/docs/diffusers

12. Hugging Face. Transformers documentation [Електронний ресурс]. – Режим доступу: https://huggingface.co/docs/transformers

13. OpenAI. CLIP model card / implementation notes (ViT-L/14) [Електронний ресурс]. – Режим доступу: https://github.com/openai/CLIP

14. CustomTkinter. Documentation [Електронний ресурс]. – Режим доступу: https://customtkinter.tomschimansky.com/

15. Pillow (PIL Fork). Documentation [Електронний ресурс]. – Режим доступу: https://pillow.readthedocs.io/

16. Google AI for Developers. Gemini API [Електронний ресурс]. – Режим доступу: https://ai.google.dev/

17. Stability AI. Platform documentation [Електронний ресурс]. – Режим доступу: https://platform.stability.ai/

18. SQLite Consortium. SQLite Documentation [Електронний ресурс]. – Режим доступу: https://www.sqlite.org/docs.html

19. Heusel M. et al. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium // NeurIPS. – 2017.

20. Salimans T. et al. Improved Techniques for Training GANs // NeurIPS. – 2016.

21. Hertz A. et al. Prompt-to-Prompt Image Editing with Cross-Attention Control // arXiv:2208.01626. – 2022.

22. Ho J., Salimans T. Classifier-Free Diffusion Guidance // NeurIPS 2021 Workshop on Deep Generative Models.

23. Karras T. et al. Analyzing and Improving the Image Quality of StyleGAN // CVPR. – 2020.

24. Esser P. et al. Taming Transformers for High-Resolution Image Synthesis // CVPR. – 2021.

25. Podell D. et al. SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis // arXiv:2307.01952. – 2023.

26. Peebles W., Xie S. Scalable Diffusion Models with Transformers (DiT) // ICCV. – 2023.

27. NVIDIA CUDA Toolkit Documentation [Електронний ресурс]. – Режим доступу: https://docs.nvidia.com/cuda/

28. PyTorch. Get Started / CUDA builds [Електронний ресурс]. – Режим доступу: https://pytorch.org/get-started/locally/

29. OpenRouter. Models / API reference [Електронний ресурс]. – Режим доступу: https://openrouter.ai/docs

30. Requests: HTTP for Humans documentation [Електронний ресурс]. – Режим доступу: https://requests.readthedocs.io/

31. MathWorks. Deep Learning Toolbox documentation [Електронний ресурс]. – Режим доступу: https://www.mathworks.com/help/deeplearning/

32. Brookes D., Martin-Ivie N. CLIPScore and Multimodal Image-Text Retrieval // Tutorial surveys on metric design. – 2022–2023. (оглядові матеріали)

33. Lin T.-Y. et al. Microsoft COCO: Common Objects in Context // ECCV. – 2014. (контекст датасетів візуально-мовного навчання)

34. Betker J. et al. Improving Image Generation with Better Captions (DALL·E 3 technical report) // OpenAI. – 2023.

35. Stability AI. SD3 Technical Report materials / announcements [Електронний ресурс]. – Режим доступу: офіційні публікації Stability AI (за наявности).