Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та безпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

ДОСЛІДЖЕННЯ ДИФУЗІЙНИХ МОДЕЛЕЙ У ЗАДАЧІ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ ЗА ТЕКСТОВИМ ОПИСОМ

RESEARCH OF DIFFUSION MODELS IN THE TASK OF IMAGE GENERATION FROM TEXT DESCRIPTIONS

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІм-21
Фролов Олександр Денисович

(підпис)

Керівник:
к.т.н., доцент
Пех Петро Антонович

(підпис)

Кваліфікаційну роботу
допущено до захисту
« ___ » грудня 2025 р.
Гарант освітньої програми:
к.т.н., доцент Гринюк Сергій Васильович

(підпис)

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та безпеки*
Ступінь вищої освіти: *магістр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ доц. Т.ТЕРЛЕЦЬКИЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Фролову Олександру Денисовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Дослідження дифузійних моделей у задачі генерації* зображень за текстовим описом

Керівник роботи *к.т.н., доцент Пех Петро Антонович*

затверджені наказом закладу вищої освіти від «17» червня 2025 року № 290/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 09.12.2025р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області, різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Аналіз проблеми за темою та постановка завдань дослідження; Актуальність текстово-візуальної генерації; п роблематика сучасних генеративних моделей; п*остановка завдань дослідження; об'єкт, предмет, мета й методи роботи; проєктування системи генерації зображень; вибір мов, бібліотек і структур; проєктування апаратної частини; реалізація та експериментальні дослідження. Висновки*

5. Перелік графічного (ілюстративного) матеріалу:
Актуальність, мета, завдання (1 сл.); Загальна схема роботи дифузійної моделі (1 сл.); Архітектура Stable Diffusion (1 сл.); Структура U-Net та схема денойзингу (1 сл.); Прямий та зворотний процеси дифузії (1 сл.); формалізація дифузійного процесу (1 сл.); Графік β-параметрів (1 сл.); Порівняльна характеристика моделей (1 сл.); Структурна схема програмно-апаратного комплексу (1 сл.); Порівняльний аналіз якості (1 сл.);

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Теоретичні основи дослідження дифузійних моделей	*Пех П.А., доцент*		
Проєктування системи генерації зображень	*Пех П.А., доцент*		
Реалізація та експериментальні дослідження	*Пех П.А., доцент*		
Нормоконтроль	*Багнюк Н.В., доцент*		
Гарант ОП	*Гринюк С.В., доцент*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Міскевич О.І., ст.викладач*		

7. Дата видачі завдання 18.06.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд літератури із досліджуваної проблеми*	До 01.08.2025 р.	
2.	*Теоретичні основи дослідження дифузійних моделей*	До 20.08.2025 р.	
3.	*Проєктування системи генерації зображень*	До 25.09.2025 р.	
4.	*Реалізація та експериментальні дослідження*	До 20.10.2025 р.	
5.	*Висновки та пропозиції*	До 25.10.2025 р	
6.	*Формування списку використаних джерел*	До 27.10.2025 р.	
7.	*Формування додатків*	До 30.10.2025 р.	
8.	*Оформлення ілюстративного матеріалу*	До 05.11.2025 р.	
9.	*Представлення остаточного варіанту кваліфікаційної роботи керівникові*	До 11.11.2025 р.	
10.	*Нормоконтроль*	До 29.11.2025 р.	
11.	*Інструментальна перевірка на академічний плагіат*	До 02.12.2025 р.	
12.	*Здача кваліфікаційної роботи та всіх супровідних документів на кафедру*	До 09.12.2025 р.	

Здобувач вищої освіти _____ (підпис) Фролов О.Д. (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (підпис) Пех П.А. (прізвище, ініціали)

АНОТАЦІЯ

Фролов О. Д. Дослідження дифузійних моделей у задачі генерації зображень за текстовим описом. Рукопис.

Кваліфікаційна робота магістра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота містить вступ, три розділи, висновки, список використаних джерел та додатки. Об'єктом дослідження є процес генерації зображень у системах штучного інтелекту. Предметом дослідження є дифузійні моделі та методи їх застосування в програмно-апаратних засобах. Метою роботи є дослідження принципів роботи дифузійних моделей та створення системи генерації зображень за текстовим описом із використанням сучасних бібліотек та апаратних компонентів.

У першому розділі подано огляд сучасних методів генерації зображень (GAN, VAE, трансформери, дифузійні моделі), описано архітектуру Stable Diffusion та особливості процесів зашумлення й відновлення зображень.

Другий розділ присвячено проєктуванню системи. Обґрунтовано вибір технологій Python, PyTorch та бібліотеки HuggingFace Diffusers для реалізації моделі, а також застосування протоколу MQTT та мікроконтролера ESP32 для організації взаємодії між програмною та апаратною частинами.

У третьому розділі наведено реалізацію системи та результати експериментів. Реалізовано сервер генерації зображень, створено прошивку ESP32 та виконано тестування планувальників дифузійної моделі. Отримано показники швидкодії та якості, що дали змогу визначити оптимальні режими роботи та продемонструвати практичну придатність розробленої системи.

Ключові слова: дифузійна модель, генерація зображень, Stable Diffusion, штучний інтелект, MQTT, ESP32.

ANNOTATION

Frolov O. Research of Diffusion Models in the Task of Image Generation from Text Descriptions. Manuscript.

Qualifying work of a Master's of EP «Computer Engineering» specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2025.

The qualification work consists of an introduction, three chapters, conclusions, a list of references, and appendices. The object of the study is the process of image generation in artificial intelligence systems. The subject of the study is diffusion models and methods of their application in software and hardware solutions. The aim of the work is to investigate the principles of diffusion-based image generation and to develop a system capable of producing images from textual descriptions using modern libraries and embedded components.

The first chapter provides an overview of contemporary image generation methods, including GANs, VAEs, transformer-based architectures, and diffusion models. It also describes the architecture of Stable Diffusion and the principles of forward and reverse diffusion processes.

The second chapter focuses on system design. It justifies the use of Python, PyTorch, and the HuggingFace Diffusers library for implementing the model, as well as the application of the MQTT protocol and the ESP32 microcontroller for communication between software and hardware components.

The third chapter presents the implementation of the system and the results of experimental evaluation. A server for image generation was developed, ESP32 firmware was implemented, and testing of different diffusion schedulers was carried out. The obtained performance indicators made it possible to identify optimal operation modes and demonstrate the practical applicability of the developed system.

Keywords: diffusion model, image generation, Stable Diffusion, artificial intelligence, MQTT, ESP32.

ЗМІСТ

ВСТУП

Швидкий розвиток методів штучного інтелекту, зокрема генеративних моделей, суттєво розширив можливості автоматичного створення зображень за текстовим описом. Дифузійні моделі, що стали новим етапом еволюції генеративних нейронних мереж, демонструють високу точність реконструкції деталей, гнучкість параметричного керування та стабільність результатів. Водночас їх інтеграція у програмно-апаратні системи залишається актуальним завданням, оскільки поєднання потужних моделей із мікроконтролерами, сенсорами та інтерфейсами взаємодії відкриває нові можливості для створення інтелектуальних пристроїв та кіберфізичних систем. Саме тому дослідження дифузійних моделей та побудова практичної системи генерації зображень за текстом є важливим як у науковому, так і в прикладному аспектах.

Метою роботи є дослідження дифузійних моделей для генерації зображень за текстовим описом та розроблення програмно-апаратної системи, що забезпечує створення, обробку та візуалізацію результатів на основі сучасних алгоритмів штучного інтелекту.

Об’єкт дослідження – процес генерації зображень за текстовим описом у системах штучного інтелекту.

Предмет дослідження – дифузійні моделі та методи їх застосування у програмно-апаратних засобах для генерації зображень.

Завдання, які необхідно виконати:

– реалізувати програмну частину системи генерації зображень на основі моделі Stable Diffusion;

– розробити серверний модуль взаємодії між моделлю та апаратною платформою ESP32;

– рослідити ефективність різних планувальників (samplers) дифузійної моделі та порівняти їх за швидкістю та якістю роботи;

– дослідити ефективність різних планувальників (samplers) дифузійної моделі та порівняти їх за швидкістю та якістю роботи;

– візуалізувати процеси генерації та результати роботи системи шляхом отримання зображень і відображення статусу на апаратному індикаторі;

– спроектувати архітектуру програмно-апаратної системи, що включає модель, сервер, протокол обміну та мікроконтролер;

– запропонувати рекомендації щодо вибору оптимальних методів генерації та можливих напрямів подальшого вдосконалення системи.

Апробація результатів. Результати роботи представлені на IV міжнародній науково-практичній Інтернет-конференції (ІПШРІТ 2025), яка проводилася 25 листопада 2025 р., м . Черкаси [1].

Публікації. Результати роботи були представленні у статті, опублікованій у науковому журналі «Комп’ютерно-інтегровані технології: освіта, наука, виробництво», 2025 року [2].

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ ДИФУЗІЙНИХ МОДЕЛЕЙ

1.1 Аналіз сучасних методів генерації зображень з текстових описів

Одним із ключових напрямів розвитку сучасного штучного інтелекту є створення моделей, здатних перетворювати текстові описи у візуальні образи. Такі системи мають широке застосування у дизайні, мистецтві, ігровій індустрії, рекламі та автоматизованому контент-генеруванні. Протягом останніх років методи генерації зображень еволюціонували від класичних нейронних мереж до складних дифузійних процесів, що забезпечують високу реалістичність та контрольованість результатів.

1.1.1 Огляд GAN, VAE, трансформерів та дифузійних моделей

Першими ефективними моделями генерації зображень стали генеративно-змагальні мережі (Generative Adversarial Networks, GAN), запропоновані І. Гудфеллоу у 2014 році. Архітектура GAN складається з двох частин – генератора та дискримінатора (рис. 1.1), які змагаються між собою. Генератор створює зображення, намагаючись «обманути» дискримінатор, тоді як дискримінатор вчиться розпізнавати, які зображення є реальними, а які – штучно створеними.

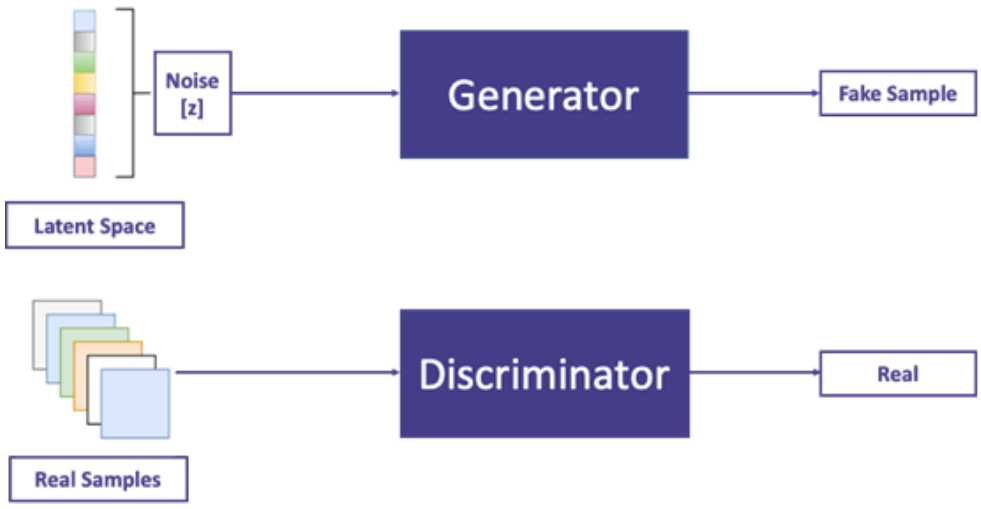

Рисунок 1.1 – Загальна архітектура GAN: взаємодія генератора і дискримінатора [3]

Основною перевагою GAN є здатність формувати високоякісні, фотореалістичні зображення. Проте навчання таких моделей є складним і часто супроводжується нестабільністю – модель може «застрягати» у стані, коли генерує лише обмежену кількість варіацій (ефект mode collapse).

Згодом з'явилися вдосконалені версії, такі як DCGAN (Deep Convolutional GAN), яка використовує згорткові шари, StyleGAN і StyleGAN2, що забезпечили реалістичну генерацію людських облич, а також AttnGAN – модель, яка вперше інтегрувала механізм уваги для врахування текстового опису під час генерації.

Таким чином, GAN стали фундаментом для подальших досліджень у сфері генерації зображень, проте залишалися недостатньо контрольованими при роботі з текстовими описами.

Варіаційні автокодери (Variational Autoencoders, VAE), запропоновані у 2013 році, стали ще одним напрямом генеративного моделювання. Їхня основна ідея полягає у кодуванні зображення у латентний простір, з якого потім відновлюється вихідне зображення через декодер. Під час навчання модель не просто запам'ятовує зображення, а навчається представляти їх у вигляді ймовірнісного розподілу (зазвичай нормального). Це дозволяє плавно інтерполювати між зображеннями в латентному просторі.

Перевагами VAE є стабільність навчання і здатність контролювати властивості зображень. Однак їхнім недоліком є розмитість результатів через спрощену апостеріорну апроксимацію. Покращені варіанти – β-VAE (для disentanglement) і CVAE (conditional VAE), який враховує додаткову інформацію, зокрема текстові описи [4].

VAE стали важливою проміжною ланкою, оскільки саме вони ввели концепцію латентного простору, яка згодом була використана в латентних дифузійних моделях (LDM).

З появою трансформерів (Transformers) у 2017 році в галузі обробки природної мови (NLP) відкрився новий підхід і до генерації зображень. Основна ідея полягає у використанні механізму самоуваги (self-attention), який дозволяє

моделі знаходити взаємозв’язки між словами в тексті та елементами зображення [5].

Першою масштабною системою, що поєднала текст і зображення, стала CLIP (Contrastive Language–Image Pre-training) від OpenAI (2021 рік). Вона навчає текстовий і візуальний енкодери у спільному семантичному просторі, що дозволяє обчислювати схожість між текстом і зображенням [6]. На основі цього принципу було створено DALL·E, який перетворює текст у послідовність візуальних токенів, і Imagen від Google, що використовує трансформер T5 для тексту та дифузійний декодер для створення високоякісних зображень.

Трансформерні моделі забезпечили глибоке розуміння контексту тексту і семантичну відповідність між словами та візуальними елементами, але самостійно не досягали якості сучасних дифузійних підходів.

Останнім етапом еволюції генеративних систем стали дифузійні моделі (Diffusion Models), які моделюють процес поступового додавання шуму до зображення та його подальшого відновлення (рис. 1.2).

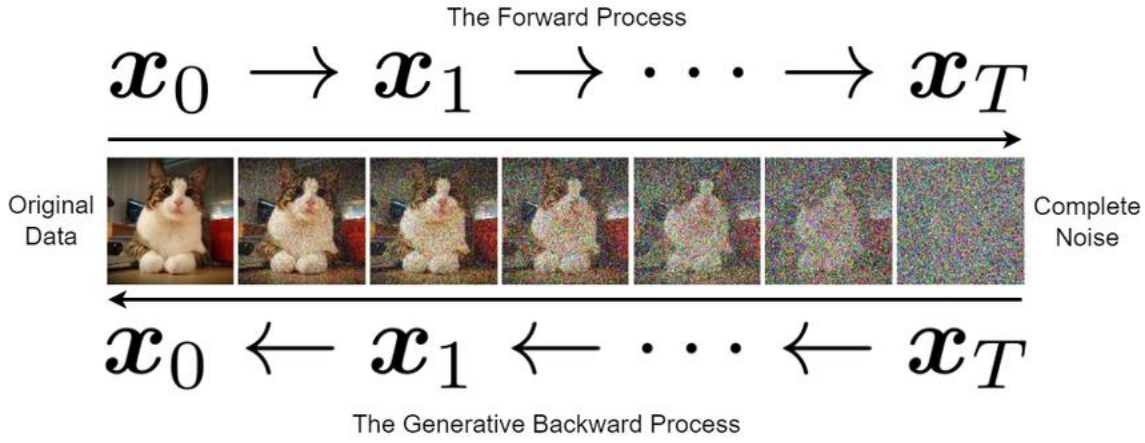

Рисунок. 1.2 – Схематичне представлення прямого (forward) і зворотного (reverse) процесів дифузії [7]

У forward-процесі зображення поступово зашумлюється на кілька сотень кроків, доки не перетвориться на випадковий шум. У зворотному процесі навчена нейромережа крок за кроком видаляє шум, відновлюючи структуру об’єкта.

На практиці сучасні реалізації – DDPM, DDIM, Stable Diffusion, Imagen – використовують оптимізовані варіанти цього принципу. Зокрема, Stable Diffusion працює у латентному просторі, що суттєво зменшує обчислювальні витрати, а текстовий опис кодується за допомогою CLIP або T5 [8].

Дифузійні моделі забезпечують високу фотореалістичність, контрольованість процесу та стійкість навчання, що робить їх найефективнішим підходом серед сучасних генеративних методів.

1.1.2 Порівняльна характеристика сучасних моделей (Stable Diffusion, DALL·E, Imagen, Midjourney)

Після появи перших генеративних архітектур, таких як GAN та VAE, розвиток технологій обробки тексту й візуальної інформації призвів до створення систем, здатних поєднувати природну мову з високоякісною візуальною генерацією. Найбільш відомими представниками цього напрямку є моделі Stable Diffusion, DALL·E, Imagen та Midjourney. Незважаючи на спільну мету – синтез зображень за текстовим описом, – вони реалізують різні підходи до архітектури, навчання та контролю якості.

Stable Diffusion – це латентна дифузійна модель, представлена у 2022 році компанією Stability AI. Її ключова особливість полягає у використанні латентного простору, де процес дифузії відбувається не на рівні пікселів, а у стислому поданні, отриманому за допомогою автокодера VAE.

Текстовий опис спочатку перетворюється у векторне подання через CLIP Text Encoder, після чого це подання впливає на процес дифузії, керуючи тим, які деталі зображення повинні бути збережені або підкреслені. Генерація завершується декодуванням результату назад у піксельний простір.

Переваги**:**

– працює на звичайних графічних процесорах (GPU середнього рівня);

– відкрита архітектура, що дозволяє створювати власні чекпоїнти;

– підтримує fine-tuning (DreamBooth, LoRA) для персоналізованого навчання.

Недоліки**:**

– вимагає налаштування гіперпараметрів для оптимальної якості;

– генерація складних сцен може потребувати доопрацювання текстових промптів.

DALL·E, розроблена компанією OpenAI, стала однією з перших систем, що продемонструвала реалістичну синтезу зображень із довільних текстових описів. Перша версія (2021 рік) використовувала трансформерну архітектуру, де і текст, і зображення розглядалися як послідовність токенів. Друга версія DALL·E (2022) – поєднала CLIP та дифузійну модель для поліпшення якості.

CLIP тут використовується для обчислення спільного простору текст-зображення, а дифузійний декодер – для генерації фінального зображення, яке найкраще відповідає цьому простору. Користувачі можуть змінювати частини зображень за допомогою inpainting, що дозволяє «домальовувати» відсутні області або модифікувати композицію [9].

Переваги:

– висока відповідність між текстом і зображенням;

– можливість редагування частин сцени.

Недоліки:

– модель є закритою (немає відкритого доступу до архітектури;

– обмеження на зміст, контрольовані політикою OpenAI.

Imagen – це розробка компанії Google Research (Brain Team), що поєднує потужну мовну модель T5 із багаторівневою дифузійною мережею. Основна ідея полягає в тому, що точність розуміння тексту значно впливає на якість зображення, тому Google зосередилася на покращенні семантичного узгодження між мовою і візуальною частиною.

Imagen використовує каскадну структуру дифузії: спочатку створюється зображення низької роздільності (64×64), яке потім послідовно покращується до 1024×1024 за допомогою super-resolution diffusion models. За рахунок цього метод забезпечує виняткову деталізацію та узгодженість із текстом, хоча залишається закритим для публічного використання [10].

Переваги:

– висока реалістичність і точність передачі змісту;

– ефективна багаторівнева обробка.

Недоліки:

– недоступна для відкритого використання;

– потребує великих обчислювальних ресурсів (кластер TPU).

Midjourney – це комерційна генеративна система, яка активно використовується в мистецьких і дизайнерських спільнотах. Її архітектура офіційно не розкривається, проте за результатами можна зробити висновок, що вона базується на дифузійному підході, подібному до Stable Diffusion, але має власні механізми стилізації та підсилення візуальних текстур.

Midjourney вирізняється тим, що орієнтована не на технічну точність, а на естетичну виразність. Її алгоритм надає перевагу композиційним рішенням, гармонії кольорів та художнім ефектам. Це зробило модель популярною серед дизайнерів, ілюстраторів і митців [11].

Переваги:

– висока художня якість і стильова гнучкість;

– інтерактивність і простота використання (через Discord-інтерфейс).

Недоліки:

– закрита архітектура;

– відсутність прозорого контролю над точністю текстового опису;

Порівняння сучасних моделей показує, що всі вони базуються на ідеї дифузійного процесу, але відрізняються за рівнем відкритості, точністю текстового аналізу, якістю зображення та спрямованістю:

– Stable Diffusion став еталоном відкритої реалізації та основою для численних модифікацій;

– DALL·E і Imagen демонструють найвищу точність передачі змісту, але є комерційними продуктами;

– Midjourney робить акцент на художньому вираженні, відсуваючи технічну точність на другий план.

Попри те, що всі розглянуті моделі належать до одного класу генеративних систем, їх архітектура, ступінь відкритості та підхід до роботи з текстовим описом суттєво відрізняються. Stable Diffusion є найбільш гнучким та доступним рішенням, що дозволяє повний локальний контроль над процесом генерації й підтримує додаткове навчання за допомогою LoRA та DreamBooth. DALL·E та Imagen, навпаки, орієнтовані на досягнення максимальної семантичної відповідності тексту, але залишаються закритими та недоступними для модифікації користувачем. Midjourney демонструє найсильнішу художню виразність і естетичну якість, проте практично не дає змоги впливати на внутрішні параметри моделі й залишається повністю хмарним сервісом.

Таким чином, сучасні системи генерації зображень за текстовим описом формують два основні напрями розвитку: моделі з відкритою архітектурою, орієнтовані на дослідників і розробників (Stable Diffusion), та комерційні закриті рішення, оптимізовані під максимальну якість або стилізацію (DALL·E, Imagen, Midjourney). У межах цієї дипломної роботи особлива увага приділяється саме відкритим дифузійним моделям, оскільки вони забезпечують необхідну відтворюваність, контрольованість і можливість інтеграції в програмно-апаратні системи.

1.1.3 Особливості архітектур: текстовий енкодер, U-Net, шумова модель, денойзер

Сучасні дифузійні моделі базуються на модульній архітектурі, у якій кожен компонент виконує окрему функцію – аналіз тексту, представлення латентного простору, моделювання шуму та покрокове відновлення зображення. Злагоджена взаємодія цих модулів дозволяє досягати високої фотореалістичності та точності семантичної відповідності між текстом і результатом.

Першим етапом роботи системи є обробка текстового опису. Для цього використовується текстовий енкодер (Text Encoder), який перетворює вхідний текст у векторне представлення – послідовність чисел, що описує семантичний зміст фрази (рис. 1.3).

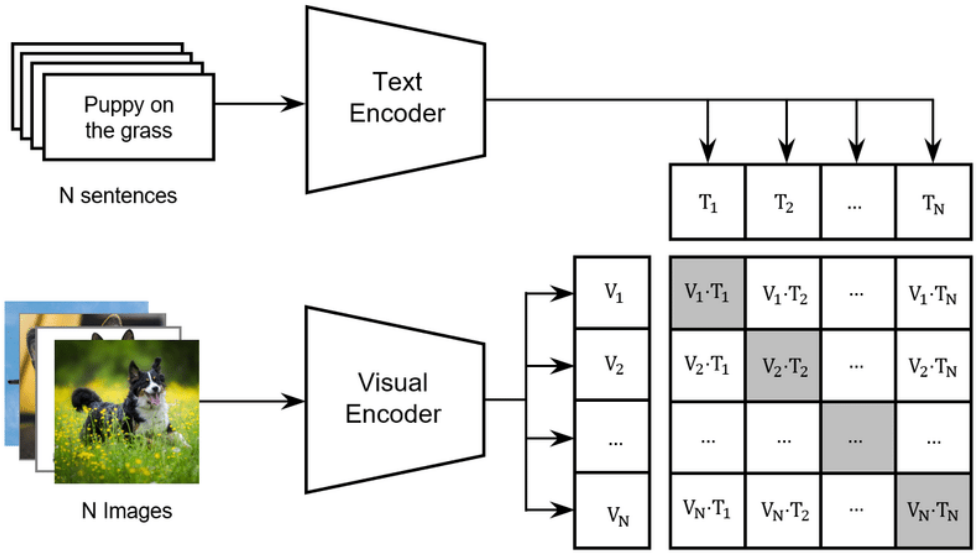

Рисунок 1.3 – Схема роботи текстового енкодера: перетворення тексту у векторне представлення [12]

У більшості сучасних моделей використовується один із двох типів енкодерів:

– T5 або BERT – мовні трансформери, які краще розуміють синтаксис і контекст;

– CLIP Text Encoder – модель, навчена на мільйонах пар «текст-зображення», що дозволяє зіставляти текстові та візуальні простори.

Енкодер формує набір текстових ембедингів, які потім передаються до основного генеративного блоку, впливаючи на кожен етап процесу дифузії через механізм cross-attention. Таким чином, саме енкодер визначає, наскільки точно згенероване зображення відповідатиме опису користувача.

Основу дифузійної мережі складає U-Net – згорткова нейромережа, що має характерну форму літери «U» завдяки наявності енкодера, декодера та skip-зв’язків між ними.

Енкодер U-Net поступово зменшує розмірність зображення, витягуючи з нього абстрактні ознаки, тоді як декодер відновлює деталізацію, комбінуючи її з високорівневими характеристиками. Skip-зв’язки дозволяють зберегти інформацію про локальні структури, що особливо важливо для відтворення дрібних деталей.

У дифузійних моделях U-Net приймає на вхід зашумлене зображення та часовий параметр, після чого прогнозує шум, який необхідно видалити на поточному кроці. Це прогнозоване значення позначається як $\epsilon_\theta(x_t, t)$ і використовується для реконструкції чистого зображення. За формулою (1.1) зашумлення має вигляд:

$$x_t = \sqrt{\alpha_t}\, x_0 + \sqrt{1 - \alpha_t}\, \epsilon, \tag{1.1}$$

де x_0– чисте зображення;

x_t– зашумлене зображення на кроці t;

ϵ– випадковий шум;

α_t– коефіцієнт з β-розкладу;

ϵ_θ– прогноз шуму від моделі U-Net.

Завдання мережі полягає у тому, щоб навчитися відтворювати шум ϵ, який було додано. Саме точність цього прогнозу визначає якість зворотного процесу відновлення.

У процесі навчання модель оптимізує похибку між реальним шумом та тим, який вона передбачає, поступово навчаючись точніше реконструювати структуру латентного представлення. На практиці це означає, що навіть невелика помилка прогнозу на ранніх кроках може суттєво вплинути на фінальний результат, оскільки зворотний процес є послідовним і кумулятивним. Тому під час тренування особливу увагу приділяють вибору функції втрат, масштабуванню шуму та коректності часових ембедингів, які визначають, який саме рівень зашумлення наразі обробляє модель.

Дифузійні системи працюють за принципом послідовного додавання та видалення шуму. Кількість та інтенсивність шуму на кожному кроці визначається шумовим планувальником (scheduler), який задає параметри β-послідовності (рис. 1.4).

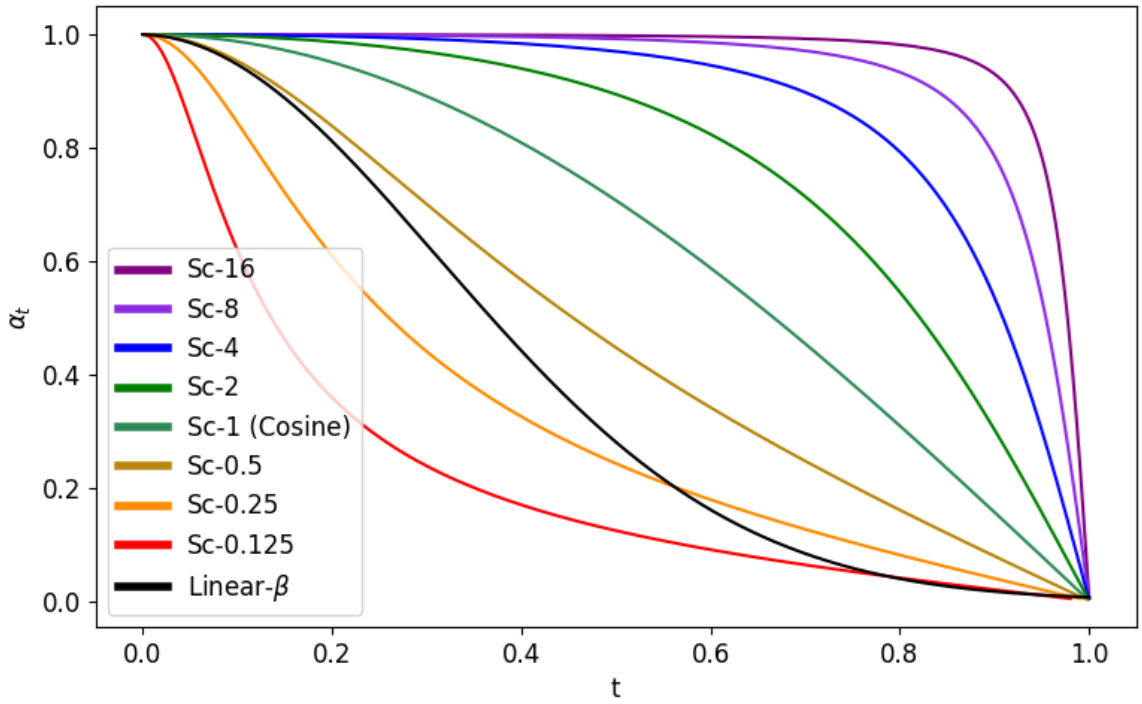

Рисунок 1.4 – Приклад графіка β-послідовності (noise schedule) [13]

Різні типи планувальників – linear, cosine, quadratic – по-різному розподіляють шум, що впливає на швидкість і стабільність навчання. У forward-процесі модель додає шум згідно з обраною β-послідовністю, тоді як у зворотному – видаляє його у зворотному порядку. Вибір планувальника є критично важливим, адже від нього залежить якість реконструкції зображення та швидкість генерації.

У практичних реалізаціях вибір конкретного планувальника безпосередньо впливає на кількість необхідних кроків дифузії та можливість компромісу між швидкістю обчислень і візуальною точністю результату. Оптимальне поєднання параметрів планувальника дозволяє значно скоротити час генерації без суттєвої втрати якості зображення.

Ключовим елементом дифузійної моделі є денойзер – нейронна мережа, що виконує функцію зворотного процесу дифузії (рис. 1.5). Вона приймає зашумлене зображення та прогнозує, який шум потрібно відняти, щоб отримати чистішу версію.

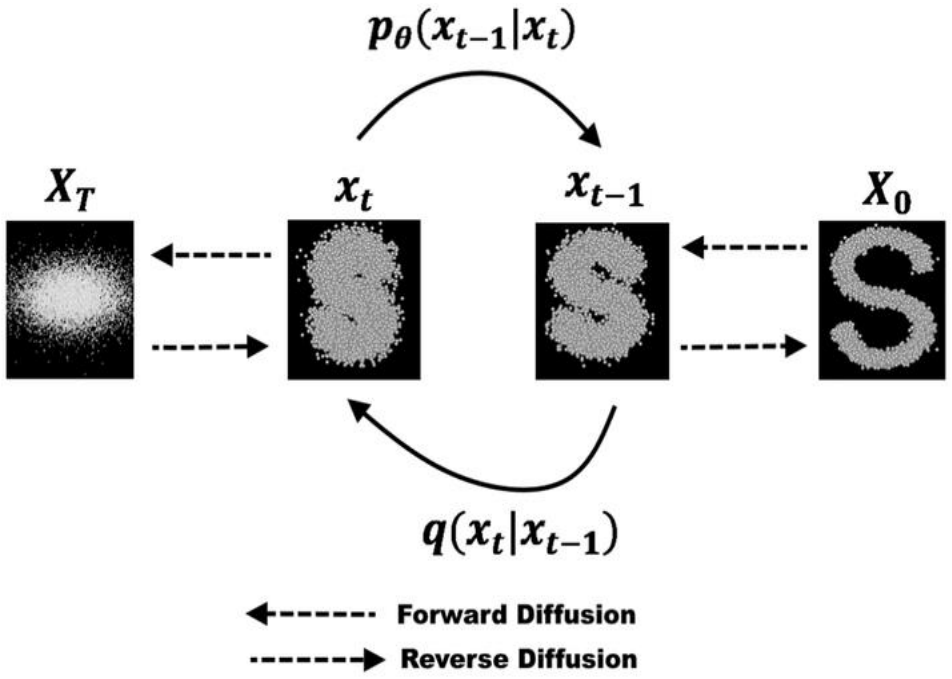

Рисунок 1.5 – Схематичне представлення процесу денойзингу [14]

На практиці денойзер і є тією самою U-Net-мережею, збагаченою часовими ембедингами та текстовим контекстом.

У процесі навчання модель оптимізує помилку між передбаченим шумом та справжнім шумом , завдяки чому з часом навчається ефективно «очищати» зображення навіть із сильного шуму.

Кожен з описаних компонентів працює у тісній взаємодії, формуючи повний цикл генерації:

– текстовий енкодер обробляє вхідний опис і створює семантичні ембединги;

– U-Net приймає латентне зашумлене зображення та ці ембединги, здійснюючи кероване відновлення;

– шумовий планувальник визначає, скільки шуму додати або прибрати на кожному кроці;

– денойзер поступово очищує дані, наближаючи результат до кінцевого зображення.

Завдяки такій архітектурі сучасні системи можуть генерувати не лише реалістичні, але й концептуально точні зображення, що відповідають контексту текстового опису.

Архітектура дифузійних моделей побудована на поєднанні семантичного аналізу тексту та покрокової реконструкції зображення. Текстовий енкодер формує змістовний опис, U-Net – обробляє візуальні структури, шумова модель – задає динаміку процесу, а денойзер – виконує поступове очищення. Така взаємодія забезпечує високу якість і контрольованість генерації, роблячи дифузійні моделі найефективнішим підходом у задачі синтезу зображень за текстовими описами.

1.2 Математичні основи дифузійних процесів

1.2.1 Формалізація прямого та зворотного процесів

Дифузійні моделі ґрунтуються на ідеї поступового зашумлення зображення (прямий процес) та навчання моделі відновлювати його (зворотний процес). Це можна розглядати як стохастичну симуляцію фізичного явища – дифузії, коли впорядкована структура поступово руйнується під дією шуму, а потім реконструюється в оберненому напрямку.

У прямому процесі, як показано на рисунку 1.5, вихідне зображення багаторазово перетворюється додаванням випадкового шуму, доки не стане майже повністю випадковим. На кожному кроці модель додає шум за нормальним розподілом із параметром, який визначає його інтенсивність.

За формулою (1.2) цей процес можна описати як:

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}\, x_{t-1}, \beta_t I), \tag{1.2}$$

де x_t – зашумлене зображення на кроці t;

β_t – коефіцієнт шуму;

I – одинична матриця, яка визначає ізотропний розподіл шуму.

Іншими словами, кожен новий стан – це попереднє зображення із трохи більшою кількістю шуму. Якщо повторити цей процес достатню кількість разів (зазвичай 1000 кроків), початкове зображення перетворюється на повністю випадковий шум, що не містить помітних візуальних ознак (рис. 1.6).

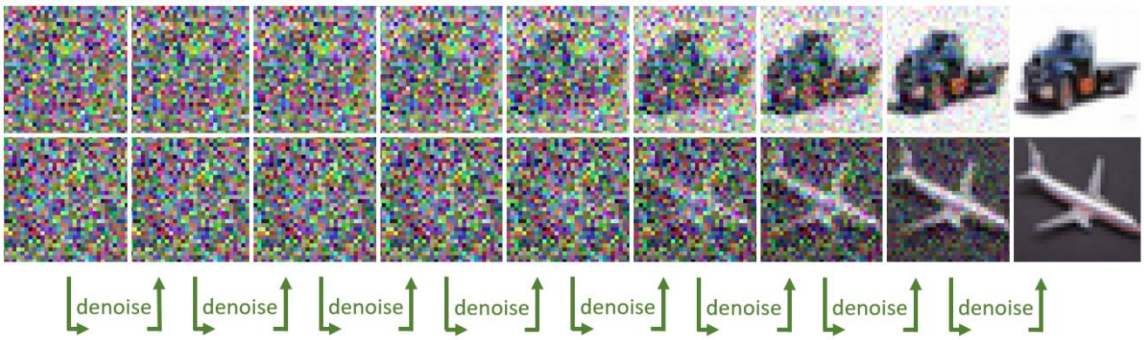

Рисунок 1.6 – Приклад поступового зашумлення [15]

Завдяки математичній властивості нормального розподілу цей процес можна описати не лише послідовно, а й аналітично за формулою (1.3):

$$q(x_t \mid x_0) = \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t}\, x_0, (1 - \bar{\alpha}_t) I\right), \tag{1.3}$$

де $\bar{\alpha}_t = \prod_{i=1}^{t} \quad (1 - \beta_i)$.

Ця формула дає змогу згенерувати будь-який проміжний стан без необхідності проходити всі попередні кроки – що зручно при навчанні моделі.

Зворотний процес – це основа генерації зображення. Його мета – навчити нейромережу обертати дифузію, тобто відновлювати зображення із шуму крок за кроком. Якщо у forward-процесі ми додаємо шум, то тут – навпаки, видаляємо його поступово, доки не отримаємо відновлене зображення.

Цей процес визначається умовним розподілом наведений у формулі (1.4):

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)), \tag{1.4}$$

де $\mu_\theta(x_t, t)$– передбачене середнє значення (тобто «очищене» зображення);

$\Sigma_\theta(x_t, t)$– коваріаційна матриця, яка описує залишкову невизначеність.

Оскільки точний аналітичний вираз для $p(x_{t-1} \mid x_t)$ невідомий, модель навчається його апроксимувати. Для цього використовується нейронна мережа – денойзер (зазвичай U-Net), параметризована, яка на кожному кроці прогнозує шум, що потрібно видалити.

Таким чином, зворотний процес є послідовним застосуванням цієї нейромережі, доки випадковий шум не перетвориться у структуроване зображення, яке відповідає заданому текстовому опису.

1.2.2 Стохастичне диференціальне рівняння (SDE)

Стохастичні диференціальні рівняння (SDE, Stochastic Differential Equations) є математичною основою, що описує, як змінюється випадковий процес у часі під впливом шуму.

У контексті дифузійних моделей вони дозволяють перейти від дискретного процесу з фіксованими кроками (як у DDPM) до неперервного стохастичного опису. Це робить моделювання більш гнучким і дозволяє використовувати різні чисельні методи інтеграції для симуляції зворотного процесу.

У найпростішій формі наведеній у формулі (1.5), SDE описує зміну стану деякої змінної x_tз плином часу як суму детермінованої складової (дрейфу) та випадкової складової (шуму):

$$dx = f(x, t)\, dt + g(t)\, dw, \tag{1.5}$$

де $f(x, t)$ – функція дрейфу (визначає середній напрям зміни даних);

$g(t)$ – коефіцієнт інтенсивності шуму;

dw – броунівський (випадковий) рух, який вносить стохастичність.

У forward-процесі SDE відповідає поступовому додаванню шуму, який розмиває структуру даних, тоді як у reverse-процесі використовується обернене

SDE, де функції мають протилежний напрям – таким чином шум поступово зменшується, а зображення відновлюється.

Дискретні дифузійні моделі (як DDPM) використовують послідовність кроків $t = 1,2,\dots,T$, кожен із яких додає невеликий шум згідно з параметром β_t. Проте якщо розглянути границю, коли кількість кроків прямує до нескінченності, отримаємо неперервний процес, що описується саме стохастичним рівнянням наведений у формулі (1.6).

$$dx = -\frac{1}{2}\beta(t)x\,dt + \sqrt{\beta(t)}\,dw, \tag{1.6}$$

Така форма відповідає forward SDE, яка описує процес поступового «розчинення» зображення у шумі. Їй відповідає обернене рівняння наведене у формулі (1.7):

$$dx = [-f(x,t) + g(t)^2\nabla_x \log p_t(x)]dt + g(t)\,d\bar{w}, \tag{1.7}$$

де x– вектор стану даних у момент часу t;

t – «час» дифузійного процесу, що визначає рівень зашумлення даних;

$p_t(x)$ – щільність ймовірності даних у точці x на момент часу t;

$\nabla_x \log p_t(x)$ – градієнт логарифма щільності (score-функція), що задає напрямок найбільш імовірної зміни стану;

f(x, t) – детермінована функція дрейфу, яка описує регулярну (середню) динаміку зміни даних;

g(t) – коефіцієнт інтенсивності шуму, який масштабує випадкову складову процесу;

dẆ – диференціал броунівського (вінерівського) процесу, що моделює випадкові флуктуації;

dx – нескінченно мала зміна стану системи за проміжок часу dt.

Цей вираз формально описує зворотний процес, який реалізує сама нейромережа під час генерації (рис. 1.7).

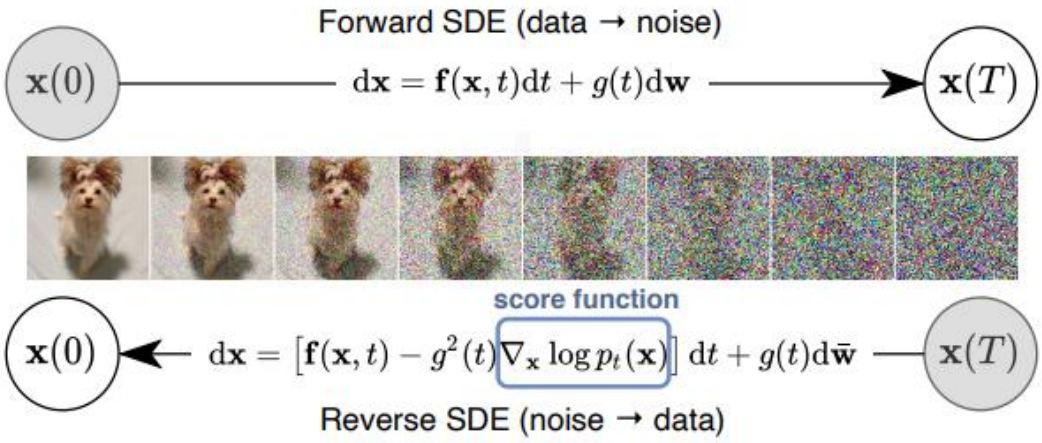

Рисунок 1.7 – Forward і Reverse SDE [16]

Якщо пояснити це простими словами, стохастичне диференціальне рівняння діє як математичний «міст» між хаосом і порядком. Forward SDE додає шум – тобто вводить у систему невизначеність, імітуючи природне розсіювання сигналу, а Reverse SDE навчає модель рухатися у зворотному напрямку, відновлюючи структуру, колір і контури зображення.

Завдяки такому опису можна застосовувати різні математичні інструменти для розв'язання SDE: чисельні інтегратори (Euler-Maruyama), апроксимації Фоккера-Планка, або їхні спрощення, як у DDIM (Denoising Diffusion Implicit Models), які скорочують кількість кроків генерації без втрати якості.

Стохастичне диференціальне рівняння – це не просто теоретичний концепт, а основа для побудови алгоритмів генерації. Більшість сучасних реалізацій (Stable Diffusion, Imagen) працюють саме на рівні апроксимації цього рівняння, де мережа навчається оцінювати локальний напрямок зміни (градієнт) у зворотному процесі.

Це дозволяє:

– контролювати кількість кроків генерації (чим менше кроків – тим швидше, але грубіше зображення);

– керувати типом шуму (лінійний, косинусний тощо);

– адаптувати процес до латентного простору (Latent Diffusion), де рівняння застосовується не до пікселів, а до стислих векторних подань.

Стохастичне диференціальне рівняння є фундаментом математичної формалізації дифузійних моделей.

Воно описує еволюцію даних у стохастичному середовищі, де forward-процес поступово додає шум, а reverse-процес – навпаки, усуває його. Завдяки SDE дифузійні моделі отримали глибоке фізико-математичне підґрунтя, що дозволило перейти від емпіричних методів до точного статистичного відтворення структури реальних зображень.

1.2.3 Пояснення ролі β-параметрів (β-schedule), вибір функції втрат

Параметри β_t є ключовими в процесі дифузії, оскільки саме вони визначають, яку кількість шуму додає або видаляє модель на кожному кроці. Від їхнього вибору залежить стабільність навчання, швидкість збіжності та якість згенерованих зображень.

Кожен крок процесу дифузії додає частку шуму, контрольовану коефіцієнтом β_t. Якщо шум зростає надто швидко – модель втрачає інформацію; якщо надто повільно – навчання стає неефективним. Тому використовуються різні графіки зміни β (β-schedule), як показано на рисунку 1.10.

У практиці Stable Diffusion використовується модифікований косинусний графік, який забезпечує рівномірний розподіл шуму та покращує відновлення дрібних деталей.

Під час навчання модель навчається передбачати шум, який було додано на кожному кроці. Для цього використовується середньоквадратична похибка (MSE) між справжнім і передбаченим шумом, яка наведена у формулі (1.8):

$$L = \mathbb{E}_{x,\epsilon,t}[\| \epsilon - \epsilon_\theta(x_t, t) \|^2], \tag{1.8}$$

де x – початкове (чисте) зображення з тренувального набору даних;

t – номер кроку дифузійного процесу, що визначає інтенсивність доданого шуму;

x_t – зображення, отримане після зашумлення xна кроці t;

ϵ – випадковий шум, доданий до зображення під час прямого процесу дифузії;

$\epsilon_\theta(x_t, t)$ – шум, який прогнозує модель U-Net для відновлення чистого зображення;

L – функція втрат, що вимірює різницю між справжнім шумом і шумом, передбаченим моделлю;

$\mathbb{E}$ – математичне сподівання, що усереднює значення втрат за всіма можливими вибірками x, ϵ, t.

Ця функція втрат стимулює модель мінімізувати різницю між тим, який шум реально було додано, і тим, який вона вважає за потрібне прибрати. В результаті модель вчиться точно «очищати» зображення на кожному етапі дифузії (рис. 1.8).

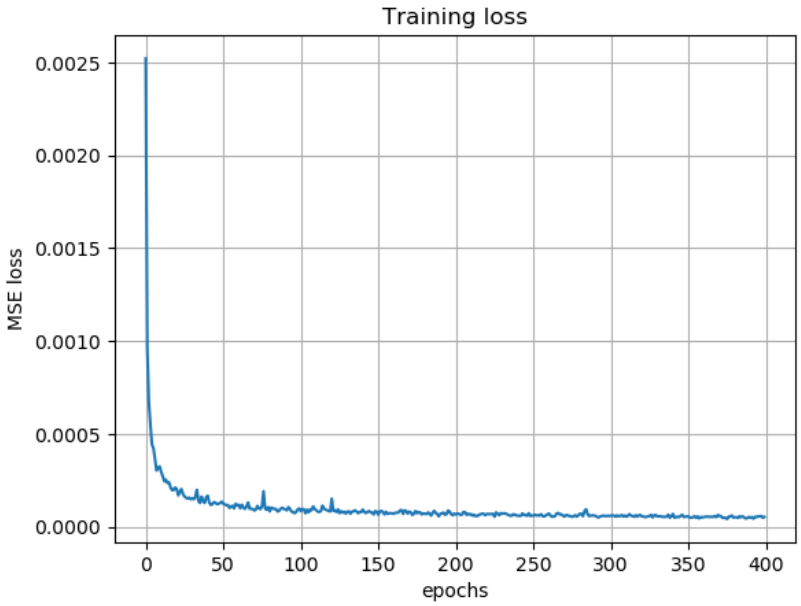

Рисунок 1.8 – Візуалізація процесу навчання на основі функції втрат MSE [17]

Графік β-параметрів визначає, як швидко інформація втрачається під час зашумлення, а функція втрат – як ефективно модель навчається її відновлювати. Оптимальне поєднання цих елементів забезпечує стійкість навчання і високу

точність реконструкції, що є основою якісної генерації зображень у дифузійних моделях.

1.3 Огляд бібліотек та середовищ для реалізації дифузійних моделей

1.3.1 PyTorch, TensorFlow, HuggingFace Diffusers

У процесі реалізації дифузійних моделей одним із ключових аспектів є вибір середовища програмної розробки та бібліотек машинного навчання. Сучасні фреймворки, такі як PyTorch, TensorFlow і HuggingFace Diffusers, забезпечують гнучкі засоби для побудови, навчання та тестування моделей глибокого навчання, включно з дифузійними архітектурами. Кожен із них має свої переваги та особливості, які визначаються структурою API, продуктивністю, підтримкою апаратного прискорення та зручністю інтеграції:

1) PyTorch – це один із найпоширеніших фреймворків для реалізації глибокого навчання, розроблений компанією Meta *(*Facebook*)*. Його основною перевагою є динамічне обчислювальне дерево (Dynamic Computational Graph), яке дозволяє виконувати обчислення «на льоту» [18]. Це означає, що структура моделі може змінюватися під час виконання, що особливо корисно при експериментах із новими архітектурами, як-от дифузійні моделі.

PyTorch має потужний набір бібліотек для роботи з нейромережами:

– torch.nn – модулі для побудови моделей (лінійні шари, згортки, нормалізації);

– torch.optim – реалізація оптимізаторів (Adam, RMSprop тощо);

– torch.utils.data – обробка та завантаження наборів даних;

– torch.cuda – інтеграція з графічними процесорами (CUDA, cuDNN).

Для роботи з дифузійними моделями PyTorch забезпечує найкращу підтримку серед наявних інструментів, оскільки більшість реалізацій, включно зі Stable Diffusion, DDPM і Imagen, створено саме на його основі. Крім того, PyTorch має високу сумісність із популярними бібліотеками:

– Transformers від HuggingFace – для інтеграції текстових енкодерів (CLIP, BERT, T5);

– TorchMetrics – для оцінки якості генерації;

– Accelerate – для розподіленого навчання на кількох GPU.

Переваги PyTorch:

– ефективна підтримка GPU через CUDA;

– висока сумісність з іншими бібліотеками ШІ;

– велика кількість прикладів, готових моделей та туторіалів;

– простота синтаксису, близька до Python.

Недоліки:

– вища споживаність пам’яті при великих моделях;

– швидкість нижча за TensorFlow у деяких задачах розподіленого навчання.

2) TensorFlow – це ще один потужний фреймворк, створений Google Research. Він використовує статичний обчислювальний граф (Static Graph), що забезпечує високу продуктивність у виробничих середовищах, але меншу гнучкість у дослідницьких експериментах [19]. TensorFlow має дві основні складові:

– Keras API – високорівневий інтерфейс для побудови нейромереж у кілька рядків коду;

– tf.data – оптимізована система обробки наборів даних, що дозволяє ефективно завантажувати зображення під час навчання.

TensorFlow активно використовується у промислових додатках, де важливі стабільність, масштабованість і інтеграція з Google Cloud. Однак для дослідницьких завдань, особливо при роботі з новими архітектурами, частіше віддають перевагу PyTorch через його гнучкість.

Переваги TensorFlow:

– Keras API – високорівневий інтерфейс для побудови нейромереж у кілька рядків коду;

– масштабованість і підтримка TPU (Tensor Processing Unit);

– сумісність із виробничими середовищами;

– розвинена візуалізація процесу навчання через TensorBoard;

– автоматична оптимізація графа обчислень.

Недоліки:

– складніша налагоджуваність у порівнянні з PyTorch;

– менше готових прикладів дифузійних моделей у відкритому доступі.

3) HuggingFace Diffusers – це спеціалізована бібліотека, розроблена компанією HuggingFace, для спрощеної роботи з дифузійними моделями. Вона надає високорівневий інтерфейс для запуску готових моделей, таких як Stable Diffusion, DDIM, Latent Diffusion Model (LDM), без необхідності глибоко занурюватися у код самої архітектури [20]. Бібліотека включає набір модулів:

– DiffusionPipeline – головний клас, що об’єднує модель, енкодер і планувальник шуму;

– Schedulers – реалізації різних β-графіків (DDIMScheduler, PNDMScheduler, EulerScheduler тощо);

– Models – U-Net архітектури, VAE-енкодери, текстові енкодери;

– Pipelines – попередньо налаштовані схеми для генерації зображень, аудіо або відео.

Перевага цієї бібліотеки полягає у тому, що вона інкапсулює всю складну логіку роботи з моделлю, залишаючи користувачу лише інтерфейс для текстового запиту. Крім того, вона інтегрується з HuggingFace Hub, що дозволяє миттєво завантажувати попередньо навчені чекпоїнти або власні моделі.

Переваги Diffusers:

– швидкий старт без необхідності глибокого програмування;

– сумісність з PyTorch;

– підтримка десятків архітектур дифузійних моделей;

– модульність та гнучкість (можна замінювати окремі компоненти).

Недоліки:

– вища вимога до оперативної пам’яті при генерації;

– залежність від GPU для продуктивної роботи.

PyTorch, TensorFlow та HuggingFace Diffusers утворюють основу екосистеми інструментів для розробки та дослідження дифузійних моделей. PyTorch виступає основною дослідницькою платформою завдяки своїй гнучкості та підтримці CUDA, TensorFlow – як масштабований інструмент для виробничих середовищ, а HuggingFace Diffusers – як високорівнева бібліотека, що дозволяє швидко реалізовувати готові рішення та інтегрувати нові моделі у практичні застосунки.

1.3.2 Моделі та датасети (COCO Captions, LAION-5B)

Для побудови дифузійних моделей надзвичайно важливу роль відіграє вибір навчальних даних. Модель не просто навчається створювати візуальні образи – вона повинна зрозуміти семантичний зв’язок між текстом і зображенням. Саме тому для тренування генераторів зазвичай використовують великі набори даних, у яких кожне зображення супроводжується коротким текстовим описом або підписом (caption).

Найпоширенішими серед таких наборів є COCO Captions і LAION-5B, які стали базою для навчання сучасних дифузійних систем, таких як Stable Diffusion, DALL·E, Imagen тощо.

Ці набори відрізняються як масштабом, так і якістю розмітки: COCO Captions містить ретельно перевірені вручну описи, тоді як LAION-5B забезпечує колосальне різноманіття тематик та стилів завдяки автоматичному збору даних із відкритих джерел. Їх спільне використання дозволяє поєднати точність семантичної відповідності з широким спектром візуального контенту під час навчання моделей.

COCO Captions (Common Objects in Context) – це один із найвідоміших відкритих наборів даних для комп’ютерного зору, створений під егідою Microsoft Research. Його головна особливість полягає у наявності текстових описів до кожного зображення, що дозволяє навчати моделі зв’язку між природною мовою та візуальними об’єктами (рис. 1.9).

Рисунок. 1.9 – Приклад пари «зображення + підписи» з набору COCO Captions [21]

Основні характеристики COCO Captions:

– приблизно 330 000 зображень високої якості;

– понад 1,5 мільйона текстових описів, по п’ять підписів до кожного зображення;

– опис об’єктів у природних сценах (люди, тварини, побутові предмети, транспорт, природа);

– деталізована розмітка – координати об’єктів, їх категорії, дії тощо.

Цей набір даних часто використовується для попереднього навчання або тонкого налаштування (fine-tuning) дифузійних моделей. Він добре підходить для початкового етапу, коли модель вчиться співвідносити короткі описи («a dog playing with a ball») із типовими сценами.

Переваги COCO Captions:

– висока якість розмітки;

– природна мова без штучних скорочень;

– підходить для початкового навчання зв’язку «текст-зображення».

Недоліки:

– обмежений обсяг даних для сучасних великих моделей;

– мала різноманітність тем і художніх стилів.

Для масштабного навчання дифузійних систем використовується набір LAION-5B (Large-scale Artificial Intelligence Open Network) – один із найбільших відкритих мультимодальних датасетів у світі. Він був створений на основі зображень і текстів, зібраних із відкритого Інтернету, з попередньою фільтрацією за змістом і мовною відповідністю.

Основні характеристики LAION-5B:

– понад 5 мільярдів пар «зображення + текст»;

– різноманітні джерела (сайти, фотостоки, енциклопедії, блоги);

– багатомовна структура (англійська, німецька, французька, японська тощо);

– очищення через модель CLIP, яка відбирає лише ті пари, де текст справді відповідає зображенню.

Цей набір став базовим для навчання моделей Stable Diffusion і OpenCLIP. Використання LAION-5B дозволило суттєво підвищити семантичну узгодженість між текстовим описом і результатом генерації – тобто модель почала краще розуміти контекст, сюжет і стиль запиту.

Переваги LAION-5B:

– величезний обсяг даних, достатній для навчання моделей із мільярдами параметрів;

– широка тематика, що включає як побутові сцени, так і мистецтво, науку, дизайн;

– підтримка багатомовності, що дозволяє тренувати універсальні енкодери тексту.

Недоліки:

– нерівномірна якість описів через автоматичне збирання даних;

– потребує фільтрації для усунення неякісних або небезпечних зображень;

– високі вимоги до обчислювальних ресурсів під час використання.

Датасети типу COCO Captions і LAION-5B використовуються для спільного навчання двох мереж:

– текстового енкодера, який перетворює опис у векторне представлення;

– візуального енкодера (або декодера), який відтворює візуальний образ.

Ці два простори з’єднуються у спільному семантичному просторі CLIP, де зображення та тексти з близьким змістом мають мінімальну відстань між векторами. Під час генерації дифузійна модель використовує цей простір, щоб узгодити «сенс» тексту із візуальною структурою майбутнього зображення.

Таким чином, якість навчального набору безпосередньо визначає здатність моделі точно відтворювати зміст опису, зберігати контекст та створювати логічні композиції. Набори COCO Captions забезпечують базове розуміння сцен, а LAION-5B – широкий семантичний і стилістичний діапазон, необхідний для повноцінного навчання великих дифузійних моделей.

У сучасних дифузійних моделях набори COCO Captions і LAION-5B виконують взаємодоповнювальні функції: перший забезпечує якісну ручну розмітку для навчання базових зв’язків, а другий – надає масштаб і різноманітність для узагальнення. Разом вони створюють основу, завдяки якій моделі типу Stable Diffusion, DALL·E та Imagen можуть перетворювати текстові описи у фотореалістичні та семантично точні візуальні образи.

1.3.3 Вимоги до апаратного забезпечення (GPU, пам’ять, обчислювальні ресурси)

Дифузійні моделі належать до найбільш обчислювально вимогливих систем глибокого навчання. Процес генерації зображень включає сотні послідовних кроків, кожен із яких виконує складні тензорні операції у високих розмірностях. Тому ефективна робота таких моделей можлива лише за наявності потужного апаратного забезпечення, зокрема графічних процесорів (GPU), оптимізованої пам’яті та прискореної системи обміну даними.

Найважливішим елементом у реалізації дифузійних моделей є графічний процесор, оскільки саме він забезпечує масово-паралельні обчислення, необхідні для згорткових і трансформерних операцій.

Основні характеристики GPU, що впливають на продуктивність:

– кількість CUDA-ядер (для NVIDIA) або потокових процесорів (для AMD);

– обсяг відеопам’яті (VRAM) – визначає, яку роздільність зображення можна генерувати;

– пропускна здатність пам’яті (memory bandwidth) – впливає на швидкість обробки батчів;

– підтримка бібліотек CUDA, cuDNN, TensorRT.

Для базового використання моделей типу Stable Diffusion достатньо GPU із 6-8 ГБ пам’яті (наприклад, RTX 3060 / 3070 / 4060 Ti). Для навчання або роботи з високою роздільністю рекомендується використовувати GPU з 12-24 ГБ VRAM (наприклад, RTX 3090 / 4090 / A6000 / Tesla V100 / A100).

Науково-дослідні інституції та лабораторії часто застосовують кластерні системи на базі NVIDIA Tesla або A100, які дозволяють проводити розподілене навчання дифузійних моделей на десятках відеокарт одночасно.

Окрім GPU, велике значення має оперативна пам’ять (RAM), особливо при роботі з великими датасетами або одночасною обробкою кількох процесів. Для комфортної роботи в середовищах розробки (Google Colab, Jupyter, Visual Studio Code) рекомендується:

– мінімум 16 ГБ RAM – для запуску готових моделей;

– 32-64 ГБ RAM – для навчання або fine-tuning моделей на власних даних.

Зберігання даних потребує швидкого SSD-диска, оскільки обсяг датасетів (наприклад, LAION) може сягати кількох терабайтів. SSD також пришвидшує завантаження моделей і доступ до чекпоїнтів (.ckpt, .safetensors).

Дифузійні моделі мають високу обчислювальну складність, оскільки кожне зображення створюється через десятки ітерацій. Наприклад, при генерації зображення розміром 512×512 пікселів модель може виконати від 50 до 200 кроків дифузії, кожен з яких вимагає кілька гігафлопів операцій.

Це призводить до високого енергоспоживання – сучасні GPU споживають до 350-450 Вт під час повного навантаження.

Для ефективного використання обчислювальних ресурсів застосовують:

– FP16 (half precision) – обчислення з половинною точністю, що знижують споживання пам’яті майже вдвічі;

– Batching і Gradient Checkpointing – методи зменшення використання пам’яті під час навчання;

– GPU Offloading / CPU-GPU Splitting – розподіл обчислень між процесором і відеокартою.

Оскільки локальні ресурси часто є недостатніми, у науковій практиці широко використовуються хмарні платформи:

– Google Colab / Kaggle Notebooks – безкоштовний доступ до GPU;

– Paperspace, RunPod, Lambda Labs – платні сервіси для навчання великих моделей;

– NVIDIA DGX Cloud – спеціалізовані кластери для обчислень зі штучним інтелектом.

Такі середовища дозволяють проводити експерименти з дифузійними моделями без потреби купувати дороге обладнання.

Для реалізації дифузійних моделей потрібні значні обчислювальні ресурси – у першу чергу потужний GPU, достатній обсяг оперативної пам’яті та швидке сховище. Оптимізація моделей за допомогою FP16, offloading і розподілених обчислень дозволяє зменшити навантаження без суттєвої втрати якості. Таким чином, навіть у межах навчального або дослідницького проєкту можливо реалізувати генерацію зображень за текстовими описами, використовуючи доступні GPU або хмарні середовища.

РОЗДІЛ 2
ПРОЄКТУВАННЯ СИСТЕМИ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ

2.1Архітектура програмно-апаратного комплексу

1) Архітектура комплексу складається з трьох основних рівнів; програмний рівень – включає дифузійну модель (Stable Diffusion / HuggingFace Diffusers), реалізовану на Python із використанням бібліотек torch, diffusers, transformers, flask або fastapi. Цей рівень виконує:

– обробку текстового запиту користувача;

– генерацію зображення за допомогою дифузійної моделі;

– передачу інформації про стан процесу (наприклад, «генерація почалась», «готово») до апаратної частини.

2) апаратний рівень – мікроконтролер ESP32, який виконує допоміжні функції:

– моніторинг стану системи (температура, навантаження, мережевий зв’язок);

– керування процесом запуску генерації (через кнопку або сенсор);

– виведення інформації на OLED-дисплей або світлодіоди;

– обмін даними з комп’ютером через Wi-Fi або MQTT-протокол.

3) комунікаційний рівень – забезпечує двосторонній обмін даними між Python-програмою і ESP32. Для цього використовується або HTTP-запит (REST API), або MQTT-брокер (наприклад, Mosquitto), що дозволяє передавати сигнали «запуск генерації», «готово», «помилка», «температура CPU» тощо.

У процесі роботи система виконує такі кроки:

1) користувач вводить текстовий запит;

2) Python-програма приймає цей запит і передає його до моделі Stable Diffusion, реалізованої через HuggingFace Diffusers;

3) після запуску процесу Python надсилає сигнал на ESP32 через REST або MQTT: «status = "generating"»;

4) ESP32 вмикає світлодіод або виводить повідомлення на OLED-дисплей («Генерація зображення…»);

5) коли процес завершується, Python надсилає сигнал «status = "done"» і, за потреби, відправляє попередній перегляд результату;

6) ESP32 оновлює індикацію («Готово»), а користувач може переглянути результат на екрані комп'ютера або у вебінтерфейсі.

Варіанти реалізації зв'язку між модулями:

– HTTP REST API: ESP32 надсилає запити типу GET /status або POST /control до Flask-сервера. Простий у реалізації, добре підходить для локальної мережі;

– MQTT: використовується брокер (наприклад, Mosquitto), через який обидві сторони обмінюються повідомленнями;

– топік ai/status – передача стану («генерує», «готово»);

– топік esp32/control – запуск або зупинка генерації. Цей підхід зручний при розгортанні в мережах IoT або в хмарних середовищах;

– WebSocket (альтернатива): реалізація двостороннього зв'язку в реальному часі через браузер або вебінтерфейс.

Запропонована архітектура дозволяє інтегрувати алгоритмічну частину дифузійної генерації з апаратною платформою ESP32, створюючи комплексну систему ШІ + IoT. Такий підхід відповідає сучасним тенденціям у комп'ютерній інженерії, де поєднуються глибинні моделі обробки даних та інтелектуальні периферійні пристрої.

2.2 Проєктування програмного забезпечення

Проєктування програмного забезпечення починається з визначення його основних функцій. Основне завдання – приймати від користувача текстовий запит, обробляти його за допомогою дифузійної моделі, отримувати зображення та відправляти повідомлення про статус виконання на апаратну частину. Для

цього програма реалізує три основні логічні шари: обробку запитів, генерацію та комунікацію.

У першому шарі система приймає текстовий опис сцени через простий веб-інтерфейс або API-запит. Цей запит проходить попередню перевірку – видаляються порожні рядки, зайві пробіли, неприпустимі символи. Також можна додатково вказати параметри генерації: роздільність зображення, кількість кроків дифузії, силу керування (guidance scale) або випадкове зерно (seed). Валідація цих параметрів здійснюється безпосередньо перед передачею запиту до моделі.

Другий шар відповідає за роботу самої дифузійної моделі. Програма використовує бібліотеку HuggingFace Diffusers, у якій реалізовано обгортку для моделей типу Stable Diffusion. У коді створюється пайплайн, який завантажує готову модель, ініціалізує планувальник кроків (наприклад, DDIM або Euler) та переносить обчислення на графічний процесор через бібліотеку PyTorch. У цьому блоці також визначено функцію, яка приймає текст і генерує зображення, після чого результат зберігається у каталозі результатів. Паралельно ведеться журналювання – у файл записуються параметри запиту, час виконання, використаний сід і шлях до збереженого зображення.

Третій шар – це комунікаційна частина, яка забезпечує зв’язок між Python-програмою та ESP32. Для цього застосовується або протокол MQTT, або звичайний REST API. У випадку MQTT кожен пристрій обмінюється короткими повідомленнями у визначених темах. Наприклад, сервер публікує повідомлення «генерація почалась», «завершено», або «помилка», тоді як ESP32 може відправляти команду «start», якщо натиснута фізична кнопка. У реалізації HTTP-зв’язку сервер надає декілька REST-ендпоїнтів: один для запуску генерації, інший для перевірки статусу, і ще один для отримання результату.

У коді комунікаційного модуля передбачено обробку кількох типових станів системи: очікування (idle), генерація (generating), успішне завершення (done) або помилка (error). Залежно від стану, Python-сервер надсилає відповідний статус ESP32, а той змінює повідомлення на дисплеї або вмикає

певний світлодіод. Такий підхід дозволяє відстежувати роботу навіть без постійного доступу до комп’ютера.

Крім основної логіки, у програмі передбачено допоміжні елементи – модуль збереження результатів, конфігураційні файли, логування та прості засоби безпеки. Усі параметри (тип моделі, планувальник, роздільність, ключі доступу до API або брокера MQTT) зберігаються у конфігураційному файлі, який легко редагувати без зміни основного коду. Модуль збереження автоматично створює нові папки за датою та веде таблицю метаданих у форматі JSON або SQLite, що полегшує пошук згенерованих зображень. Журналювання подій виконується стандартними засобами Python – фіксуються всі виклики, помилки та час обробки кожного запиту.

Додатково передбачено базовий механізм контролю помилок мережевої взаємодії та повторної відправки запитів у разі тимчасових збоїв зв’язку з сервером або брокером MQTT. Це дозволяє забезпечити безперервність роботи системи та підвищує надійність процесу генерації в умовах нестабільного з’єднання.

Структура програмного забезпечення побудована так, щоб воно могло працювати як локально, так і в хмарному середовищі (наприклад, у Google Colab або RunPod). Для прискорення обчислень використовується режим обчислень з половинною точністю (FP16) і оптимізовані бібліотеки для attention-блоків. Якщо пам’яті GPU недостатньо, частина обчислень переноситься на центральний процесор. Завдяки цьому система залишається стабільною навіть на відеокартах середнього класу.

Таким чином, спроєктоване програмне забезпечення є модульною системою, у якій текстовий запит користувача проходить послідовну обробку: від валідації і передачі у дифузійну модель – до генерації зображення, запису результату і надсилання сигналу про завершення на ESP32. Така структура забезпечує не лише правильну логіку роботи, а й гнучкість – програму можна масштабувати, додавати нові типи моделей або розширювати функціонал апаратної взаємодії без істотних змін у коді.

2.3 Проєктування апаратної частини на базі ESP32

Апаратна частина створеного комплексу виконує допоміжні функції управління, моніторингу та індикації процесу генерації зображень. В її основі використано мікроконтролер ESP32, який поєднує високу обчислювальну здатність, модуль Wi-Fi та Bluetooth, що робить його ідеальним варіантом для інтеграції з комп'ютерною системою штучного інтелекту. Основне завдання ESP32 – отримувати від Python-сервера сигнали про стан процесу, відображати їх користувачу та, за потреби, надсилати команди керування у зворотному напрямку.

Під час розроблення апаратної частини було визначено, що мікроконтролер повинен мати можливість працювати у двох режимах – пасивному (індикація станів) та активному (ініціація процесів). У пасивному режимі ESP32 приймає дані через бездротовий інтерфейс Wi-Fi. Коли сервер розпочинає генерацію, він надсилає повідомлення про початок роботи. Мікроконтролер, отримавши цей сигнал, відображає відповідне повідомлення на OLED-дисплеї або змінює стан світлодіодного індикатора. Коли процес завершується, надходить інше повідомлення, після чого індикатор переходить у режим «готово». Це забезпечує користувача візуальним зворотним зв'язком без необхідності стежити за консольною або веб-частиною програми.

У активному режимі пристрій може надсилати команди на сервер. Це реалізується за допомогою однієї або кількох кнопок, підключених до GPIO-портів. При натисканні кнопки ESP32 формує запит типу «start», який через MQTT або HTTP передається Python-програмі. Сервер приймає цю команду, додає нове завдання у чергу й запускає процес генерації. Таким чином, користувач може керувати всією системою безпосередньо через апаратну панель.

Для індикації використовується невеликий OLED-дисплей з інтерфейсом I^2C. На екрані відображаються короткі повідомлення про стан системи, наприклад: «Очікування», «Генерується…», «Готово», або «Помилка

зв’язку». Ці повідомлення змінюються автоматично після отримання відповідних даних від сервера. У коді ESP32 передбачено буферний механізм, який не дозволяє екрану мерехтіти при оновленні тексту, що робить індикацію плавною та зручною для сприйняття. Додатково використовується світлодіод, який блимає під час активного процесу та гасне після завершення.

Окрім базової індикації, апаратна частина може виконувати роль датчика середовища або контролера системного стану. Для цього до ESP32 можна підключити сенсор температури та вологості, наприклад DHT22, що дозволяє контролювати температуру навколишнього середовища або корпусу пристрою. Ці дані можуть надсилатися серверу в окремому потоці MQTT і зберігатися у файлі журналу, що корисно при тривалих тестуваннях або оцінюванні стабільності роботи GPU-системи. Таким чином, апаратна частина може виконувати не лише індикаційні, а й аналітичні функції.

Зв’язок між ESP32 та Python-програмою здійснюється через локальну Wi-Fi-мережу. У конфігурації мікроконтролера задається IP-адреса сервера або брокера MQTT. При запуску пристрій підключається до мережі, встановлює з’єднання з брокером і підписується на визначені теми повідомлень. У разі втрати з’єднання реалізовано автоматичне перепідключення, що забезпечує стабільність системи при тривалій роботі. Передбачено також обробку помилок: якщо протягом певного часу не надходять оновлення, ESP32 переходить у режим очікування і виводить попередження «Немає зв’язку».

Програмна логіка мікроконтролера побудована на стандартних бібліотеках Arduino-середовища. У коді описано ініціалізацію Wi-Fi-модуля, підключення до брокера, обробники отриманих повідомлень та оновлення інтерфейсу відображення. Основний цикл програми постійно перевіряє, чи надійшли нові дані про стан. Якщо отримано статус «generating», вмикається відповідна анімація або миготіння світлодіода; якщо «done» – дисплей показує повідомлення «Готово», а індикатор гасне. При отриманні команди «error» ESP32 виводить текст помилки, дозволяючи користувачу відразу помітити збій у роботі моделі.

Завдяки використанню ESP32 уся система стає кіберфізичною, тобто поєднує інтелектуальну обробку даних та фізичну взаємодію з користувачем. Мікроконтролер діє як посередник між складними обчислювальними процесами дифузійної моделі та людиною, забезпечуючи просту і наочну форму контролю. Таке рішення робить систему зручною для демонстрацій, лабораторних експериментів або досліджень, де важливо показати не лише результат роботи штучного інтелекту, а й сам процес у реальному часі.

У підсумку, апаратна частина комплексу реалізує кілька ключових функцій: прийом сигналів про стан від Python-сервера, індикацію цих станів на OLED-дисплеї, передачу команд керування, а також моніторинг зовнішніх параметрів середовища. Таке поєднання робить систему не просто програмним застосунком для генерації зображень, а повноцінною інтерактивною платформою, яка демонструє взаємодію штучного інтелекту з реальним фізичним пристроєм.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1 Реалізація програмної частини

Програмна частина реалізована мовою Python 3.9 із використанням бібліотек PyTorch і HuggingFace Diffusers, що забезпечують повну підтримку сучасних дифузійних моделей, включно зі Stable Diffusion. Розроблення виконувалось у середовищі Visual Studio Code. Використовувався локальний графічний процесор NVIDIA RTX 3050 Laptop GPU (4 ГБ VRAM), що дозволяє проводити генерацію зображень у роздільності 512×512 без перенесення частини обчислень на CPU.

Для початку потрібно налаштувати середовище, створивши віртуальне середовище (рис. 3.1).

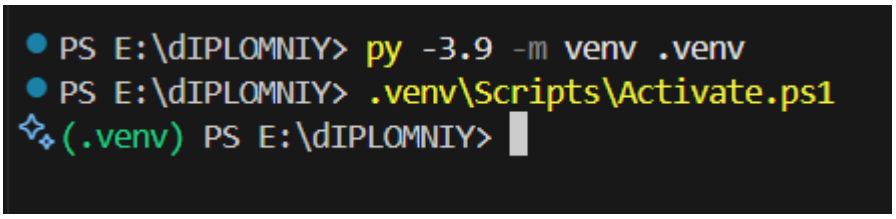

Рисунок 3.1 – Створення віртуального середовища в Visual Studio Code

Тоді встановити необхідні бібліотеки, використавши наступні команди:

– pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121;

– pip install diffusers==0.30.0 transformers accelerate scipy safetensors;

– pip install flask paho-mqtt pillow.

Після встановлення бібліотек перевіряється доступність Cuda ядер, для цього можна через термінал вивести їх статус за допомогою двох команд:

– import torch обробку;

– print(torch.cuda.is_available()).

Якщо після їх виконання виводиться «True», GPU готовий до роботи.

3.1.1 Імпорт і ініціалізація моделі

Для генерації зображень використовується попередньо натренована модель Stable Diffusion v1.5, розміщена у відкритому репозиторії HuggingFace Hub. Вона здатна перетворювати короткі текстові описи англійською мовою у фотореалістичні зображення (лістинг 3.1).

Лістинг 3.1 – Використання Stable Diffusion v1.5

```
from diffusers import StableDiffusionPipeline, DDIMScheduler
import torch
import cuda

model_id = "runwayml/stable-diffusion-v1-5"

pipe = StableDiffusionPipeline.from_pretrained(
    model_id,
    torch_dtype=torch.float16,
    safety_checker=None
).to("cuda")

pipe.scheduler = DDIMScheduler.from_config(pipe.scheduler.config)
```

кінець лістингу 3.1

Модель завантажується один раз при запуску серверу, після чого може багаторазово генерувати зображення. Параметр «torch_dtype=torch.float16» знижує споживання пам’яті приблизно на 40 %, що особливо корисно при роботі з відеокартами середнього рівня.

3.1.2 Функція генерації зображення

Основна функція приймає текстовий запит, кількість кроків дифузії, роздільність і коефіцієнт керування (guidance scale). Вона повертає готове зображення у форматі PNG і створює файл результату з метаданими (лістинг. 3.2).

Лістинг 3.2 – Функція «приймання текстових запитів, кроків дифузії, роздільності і коефіцієнта керування»

```
import os
from datetime import datetime
def generate_image(prompt, steps=30, width=512, height=512, guidance=7.5,
seed=None):
    generator = torch.Generator(device="cuda")
    if seed:
        generator.manual_seed(seed)
    result = pipe(
        prompt,
        num_inference_steps=steps,
        guidance_scale=guidance,
        width=width,
        height=height,
        generator=generator
    )
    image = result.images[0]
    folder = "results"
    os.makedirs(folder, exist_ok=True)
    filename =
f"{folder}/img_{datetime.now().strftime('%Y%m%d_%H%M%S')}.png"
    image.save(filename)
    print(f"Зображення збережено: {filename}")
    return image
pipe.scheduler = DDIMScheduler.from_config(pipe.scheduler.config)
```

кінець лістингу 3.2

Після запуску цього коду можна вводити будь-який текстовий запит англійською або українською мовою.

У цій роботі не проводилося додаткове навчання моделі, оскільки Stable Diffusion є вже повністю навченою системою на масштабному датасеті LAION-5B. Замість повторного тренування реалізовано тонке налаштування параметрів генерації, що впливають на якість і стиль вихідного зображення. Підбір значення guidance_scale дозволяє регулювати «слухняність» моделі: при низькому значенні зображення буде креативнішим, а при високому – точніше відображатиме текст.

Для відтворюваності результатів застосовується параметр seed, який фіксує генератор випадкових чисел. Це дозволяє отримати однакове зображення при повторному запуску з тим самим текстом.

3.1.3 Журналювання та метадані

Після кожної генерації програма записує у лог:

– текст запиту (prompt);

– кількість кроків і час виконання;

– обсяг використаної відеопам’яті;

– ім’я створеного файлу.

3.2 Реалізація апаратної частини

Апаратна частина системи реалізована на базі мікроконтролера ESP32, який виконує роль інтерфейсу між користувачем і програмним модулем генерації зображень. Завдяки наявності вбудованого Wi-Fi модуля ESP32 забезпечує обмін даними із сервером через протокол MQTT, а також індикацію поточного стану на OLED-дисплеї.

Усі обчислення, пов’язані з генерацією зображень, виконуються на комп’ютері, а мікроконтролер відповідає за прийом команд, передачу сигналів та візуальне відображення стану роботи системи.

Додатково ESP32 виконує роль «станового індикатора» роботи системи, оскільки отримує проміжні повідомлення від серверної програми про етапи генерації та відображає їх у реальному часі. Це дозволяє користувачу контролювати процес без необхідності відкривати інтерфейс комп’ютера. Така взаємодія забезпечує не лише зручність, а й підвищує надійність роботи, адже мікроконтролер реагує на помилки або затримки у мережевому з’єднанні та може повідомляти про них через OLED-дисплей або світлодіодний індикатор.

Для створення апаратної частини було використано такі основні компоненти: мікроконтролер ESP32 DevKitC, OLED-дисплей на контролері SSD1306 із роздільністю 128×64 пікселів, світлодіодний індикатор та кнопка запуску. Дисплея SSD1306 підключений до ESP32 через інтерфейс I^2C.

Живлення (VCC) подається на 3.3 V, сигнали SDA та SCL підключено до контактів GPIO21 і GPIO22 відповідно, що забезпечує стабільний обмін даними між пристроями (рис. 3.2).

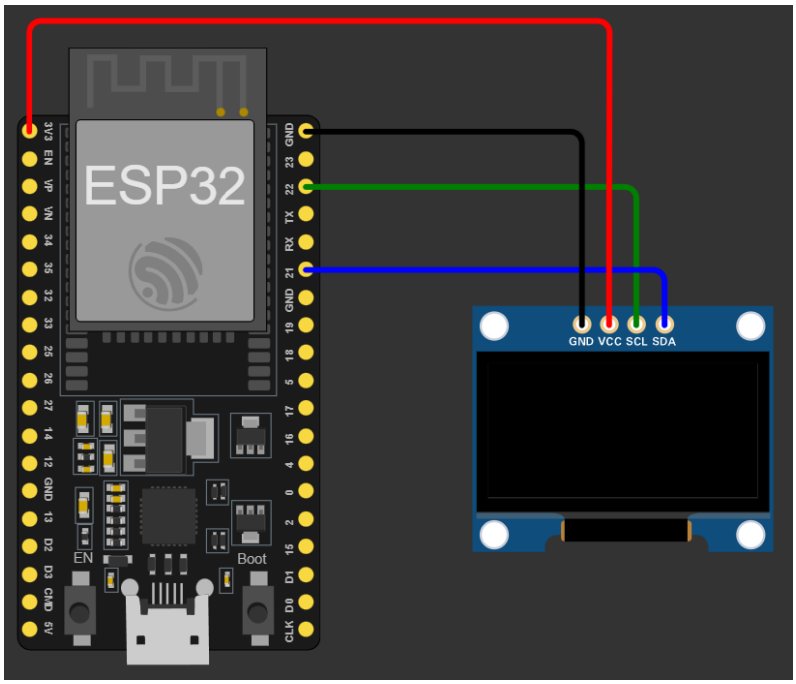

Рисунок 3.2 – Схема підключення компонентів апаратної частини системи [22]

Для забезпечення зв’язку між ESP32 та сервером у системі використовується протокол MQTT, що передбачає обмін короткими повідомленнями у форматі JSON. На стороні комп’ютера працює MQTT-брокер, який отримує команди та передає їх усім підписаним пристроям. Мікроконтролер підключається до брокера та підписується на тему ai/status, з якої отримує повідомлення про стан генерації наприклад, idle, generating, done, error (рис. 3.3).

У зворотному напрямку ESP32 публікує повідомлення у тему esp32/control, надсилаючи команди «start» або «stop». Таким чином, реалізовано двосторонній обмін між апаратною та програмною частинами без прямого дротового з’єднання.

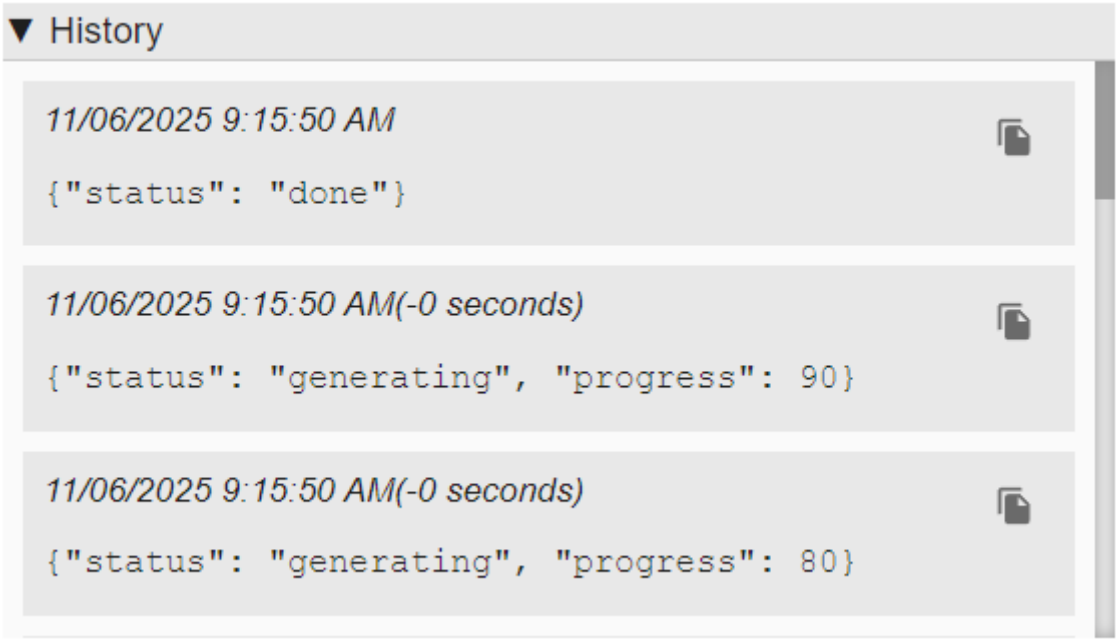

Рисунок 3.3 – Послідовність обміну даними між ESP32 та сервером через MQTT

Програмне забезпечення для ESP32 розроблено у середовищі Arduino IDE. Використано бібліотеки Network, MQTTClient (для MQTT), ssd1306 (для OLED-дисплея) та json (для роботи з JSON-повідомленнями). Код прошивки складається з трьох основних частин: ініціалізація з'єднань, обробка повідомлень та індикація станів (додаток А).

Після запуску пристрій виконує підключення до Wi-Fi, встановлює з'єднання з MQTT-брокером і переходить у стан очікування. OLED-дисплей у цей момент показує повідомлення «Готово. Натисни кнопку для старту», а світлодіод гасне. При натисканні кнопки на мікроконтролері формується команда «{"command": "start"}», яка надсилається на сервер. Як тільки Python-програма починає генерацію зображення, вона публікує повідомлення «{"status":"generating","progress":0}» (рис. 3.4). ESP32 отримує цей сигнал і оновлює дисплей, показуючи повідомлення «Генерація...» та шкалу прогресу, яка поступово заповнюється.

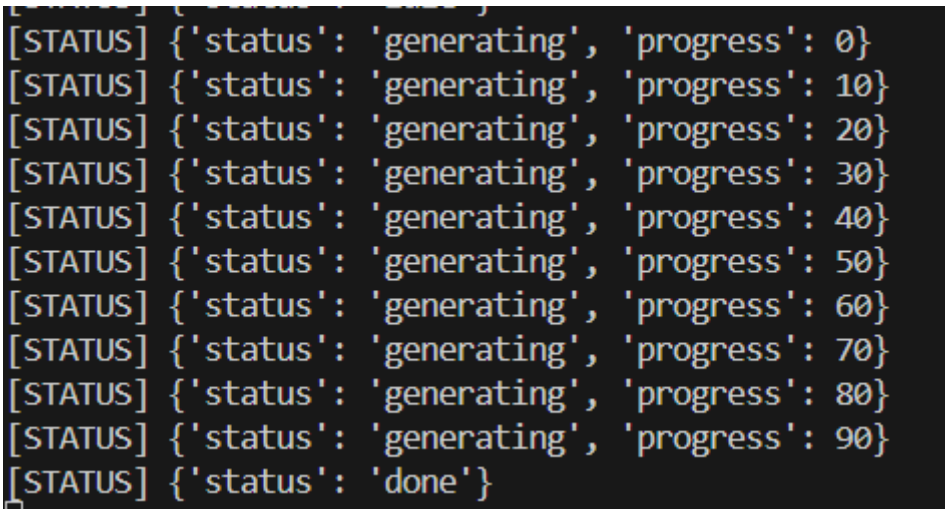

Рисунок 3.4 – Відображення стану «Генерація…» OLED-дисплея у терміналі

У процесі роботи мікроконтролер постійно перевіряє нові повідомлення від брокера. Коли сервер завершує генерацію, він надсилає статус «{"status":"done"}», як показано на рисунку 3.3. Отримавши його, ESP32 змінює напис на екрані на «Готово», а світлодіод гасне. У разі помилки з’єднання або збою генерації сервер відправляє «{"status":"error"}», після чого OLED виводить повідомлення «Помилка». Користувач таким чином отримує чіткий візуальний сигнал про стан системи в реальному часі.

У прошивці також передбачено періодичну відправку телеметрії – коротких повідомлень, які містять поточну температуру, вологість або інші параметри. Ці дані можуть бути корисними під час експериментів для контролю умов роботи пристрою або оцінки стабільності Wi-Fi-з’єднання. Додатково реалізовано автоматичне перепідключення до мережі у випадку розриву, що гарантує стабільну роботу під час тривалих сесій генерації.

У результаті реалізації апаратна частина забезпечує такі функції:

– отримання повідомлень від сервера про поточний стан генерації;

– індикацію цих станів на OLED-дисплеї та за допомогою LED-індикатора;

– надсилання команд запуску і зупинки процесу;

– передачу телеметричних даних (опціонально).

Практичні випробування підтвердили стабільність роботи системи та коректність взаємодії між компонентами. Мікроконтролер успішно відображав зміну станів у реальному часі, а сервер правильно реагував на команди, що надходили від ESP32. Таким чином, розроблена апаратна частина повністю забезпечує інтерактивну взаємодію з користувачем і підвищує наочність роботи генеративної системи.

3.3 Експериментальні дослідження

У попередніх підрозділах було представлено програмну реалізацію генеративного модуля на основі Stable Diffusion та апаратну інтеграцію з мікроконтролером ESP32, який забезпечує індикативний зв’язок між користувачем і сервером. На цьому етапі важливо оцінити не лише факт коректної роботи системи, але й вибрати оптимальний стиль генерації, тобто визначити, який тип планувальника (sampler) забезпечує найкращий баланс часу, стабільності та якості зображень.

Оскільки дифузійна модель реконструює зображення з шуму через послідовність кроків, саме планувальник визначає, яким шляхом ця реконструкція відбувається. Різні самплери (DDIM, Euler, DPM++, Heun тощо) по-різному поводяться в умовах обмежених кроків, різної роздільності та навантаження на GPU. Від цього залежить і практичність використання моделі у реальних умовах.

Мета експериментального дослідження – порівняти продуктивність різних планувальників Stable Diffusion для трьох профілів генерації:

– fast – мінімальна кількість кроків (20);

– balanced – середня кількість кроків (30);

– quality – розширені налаштування (40 кроків і підвищена роздільність 640×640).

Експерименти проводилися на GPU з підтримкою FP16, що забезпечує стабільні результати та типовий рівень продуктивності для подібних систем.

3.3.1 Методика проведення експериментів

Для тестування було використано спеціальний бенчмарк-скрипт, який автоматично генерував серії зображень різними планувальниками: ddpm, ddim, pndm, euler, euler_a, dpm++, heun.

Для кожного планувальника виконувались три набори умов:

– fast – 512×512, 20 кроків, guidance 7.0;

– balanced – 512×512, 30 кроків, guidance 7.5;

– quality – 640×640, 40 кроків, guidance 8.0.

Для всіх експериментів використовувався один і той самий prompt, що дозволило виключити вплив семантики на час генерації. Результати зберігались у JSON і CSV файлах, на основі яких виконано аналіз.

3.3.2 Аналіз продуктивності

1) Профіль fast (512×512, 20 кроків): у цьому найшвидшому режимі найкращі результати за часом показали:

– Euler – 5,568 с.;

– DPM++ – 5,582 с.;

– Euler A – 5,575 с.

Це майже ідентичні результати, що демонструє ефективність сімейства Euler та DPMSolver. Час DDPM та PNDM був трохи більшим – 5,65-5,88 с, але все ще прийнятним. Heun показав найповільніший результат – 10,589 с, що майже вдвічі більше за Euler. DDIM також виявився повільнішим за Euler – 8,652 с.

Висновок: для швидкої генерації оптимальні Euler/Euler A/DPM++.

2) Профіль balanced (512×512, 30 кроків): зі збільшенням кількості кроків співвідношення збереглося:

– DDIM – 8,181 с.;

– Euler – 8,194 с.;

– Euler A – 8,208 с.;

– DPM++ – 8,211 с.

Всі ці планувальники працюють майже однаково швидко на 30 кроках. PNDM і DDPM дають трішки повільніший, але стабільний результат (8,3-8,5 с). Heun знову працює в рази повільніше – 15,871 с.

Висновок: у профілі balanced різниця між DDIM, Euler та DPM++ мінімальна, але Heun залишається непридатним через надмірну тривалість.

3) Профіль quality (640×640, 40 кроків): цей профіль показує реальні можливості алгоритмів при підвищеному навантаженні:

– Euler – 20,463 с.;

– Euler A – 20,490 с.;

– DPM++ – 20,502 с.

Вони знову виявляються найстабільнішими за часом.

Суттєво повільнішим став DDIM – 27,484 с. Найповільніший – Heun (40,003 с), що у двічі повільніше за Euler.

Висновок: при високій роздільності та великій кількості кроків Euler/Euler A/DPM++ – беззаперечні лідери.

3.3.3 Загальний порівняльний висновок

За всіма проведеними тестами можна зробити такі висновки:

1) найоптимальніші для реальної системи: Euler, Euler A та DPM++:

– стабільна швидкість;

– хороша деталізація;

– без різких піків часу;

– добре працюють як на низьких, так і на високих параметрах.

2) найповільніший і найменш практичний – Heun:

– повільний у всіх профілях;

– не дає відповідного приросту якості, щоб виправдати час.

3) DDIM і PNDM – «золоті середняки». DDIM інколи програє Euler/DPM++, але забезпечує передбачуваний стиль картинки. PNDM – стабільний, трохи повільніший, але технічно точний.

4) DDPM – референсний метод, який працює очікувано, але повільніший без практичних переваг.

3.3.4 Візуальні результати генерації

Для того щоб підтвердити результати з попереднього підрозділу та показати, як різні планувальники впливають на вигляд вихідного зображення, було відібрано кілька прикладів згенерованих зображень, отриманих у процесі тестування. Усі приклади створені за однаковим текстовим описом, що дозволяє коректно порівнювати саме роботу алгоритмів.

Для Euler видно характерні чіткі контури, добре визначені грані об’єктів та збалансовану композицію (рис 3.4). Euler забезпечує акуратну реконструкцію деталей навіть при мінімальній кількості кроків. Такий стиль генерації робить Euler одним із найкращих варіантів для швидкого отримання результатів без втрати основної семантики зображення.

Рисунок 3.4 – Приклад результату генерації Euler, 512×512, balanced

Euler та DPM++ показують близьку за якістю передачу форми та структури (рис. 3.5). DPM++ дає трохи більш стабільний і передбачуваний результат із чистими контурами, тоді як Euler A робить картинку більш «творчою», з м’якшими переходами та багатшою текстурою. Візуальна різниця показує, що

Euler A може бути кращим для художніх стилів, тоді як DPM++ більш підходить для задач, де важлива геометрична точність і чіткість.

Рисунок 3.5 – Порівняння Euler A (справа) та DPM++ (зліва) при роздільності 640×640

DDIM відтворює загальний зміст зображення коректно, однак у порівнянні з Euler/DPM++ контури менш різкі, а текстури – більш згладжені (рис. 3.6). Це підтверджує характер DDIM як «нейтрального» планувальника, який забезпечує прийнятний результат, але не досягає рівня деталізації, властивого сучаснішим схемам DPM++. Разом з тим DDIM може бути корисним у випадках, коли необхідно отримати м’якші та менш контрастні зображення.

Рисунок 3.6 – Результат DDIM для якісного профілю

Ці приклади дозволяють побачити різницю в стилі та деталізації. Euler та DPM++ дають більш виразні контури й стабільну композицію, Euler A – трохи «творчіший», DDIM – згладжений і більш нейтральний.

Повні результати експериментальних вимірювань наведені у Додатку Б (таблиця Б.1).

3.3.5 Підсумковий висновок щодо вибору стилю генерації

За підсумками експериментальних досліджень можна рекомендувати використання планувальників Euler або DPM++ як основних для побудови системи генерації зображень за текстовим описом. Вони забезпечують:

– найменший час виконання серед усіх протестованих варіантів;

– стабільну якість зображень;

– хорошу масштабованість при збільшенні роздільності.

Це робить їх оптимальними для практичного використання в умовах обмежених обчислювальних ресурсів, таких як локальні робочі станції без кластерів чи розподілених GPU.

ВИСНОВКИ

В ході дослідження було реалізовано програмну частину системи генерації зображень, що базується на дифузійній моделі Stable Diffusion та програмній екосистемі Python. Для цього було використано бібліотеки PyTorch та HuggingFace Diffusers, налаштовано середовище виконання у формі віртуального оточення, виконано завантаження моделі, її ініціалізацію та генерацію зображень на основі текстових описів. Реалізація забезпечила повний цикл обробки запиту користувача, включаючи параметризацію генерації та збереження результатів.

Було розроблено серверний модуль для взаємодії між моделлю та апаратною платформою ESP32, що працює через протокол MQTT. Сервер реалізує приймання команд, запуск генерації, формування відповідей та передавання статусів виконання. Це дало змогу забезпечити коректний обмін даними між програмною частиною та апаратним індикатором системи.

У ході експериментальної частини було досліджено ефективність різних планувальників дифузійної моделі (DDIM, Euler, Euler A, DPM++, PNDM, DDPM, Heun). На основі отриманих бенчмарків визначено, що найкраще співвідношення швидкості та якості забезпечують методи Euler, Euler A та DPM++, тоді як Heun є найповільнішим і менсш придатним для практичного застосування. Це дозволило сформувати рекомендації щодо вибору оптимальних параметрів генерації.

Було візуалізовано процес роботи системи шляхом отримання згенерованих зображень у різних профілях якості та продемонстровано відмінності між результатами, отриманими за допомогою різних планувальників. Окрім цього, реалізовано відображення статусу генерації на апаратному індикаторі, що підвищує наочність та інтерактивність роботи моделі.

У роботі було спроєктовано та реалізовано повну архітектуру програмно-апаратної системи, яка включає модель генерації зображень, серверну частину, модуль обміну даними через MQTT та апаратний блок на основі ESP32.

Реалізація довела можливість інтеграції сучасних генеративних моделей у ресурсно обмежені кіберфізичні системи з мінімальними апаратними вимогами.

В результаті проведених досліджень отримано повнофункціональну систему генерації зображень за текстовим описом, здатну працювати у взаємодії з апаратними компонентами та забезпечувати стабільні результати. Як показують проведені експерименти, найкращих результатів можна досягти, використовуючи планувальники сімейств Euler та DPM++, а сама система може бути адаптована для використання в навчальних, дослідницьких або демонстраційних цілях.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Пех П.А., Фролов О.Д. Генерація зображень за текстовим описом за допомогою дифузійної моделі. *Інновації та перспективні шляхи розвитку інформаційних технологій*: зб. матеріалів доп. учасн. IV Міжнар. наук.-практ. конф. Черкаси, 2025. С. 211-213.

2. Pekh P., Frolov O. Research on the technology of image generation based on text description using the Stable Diffusion model. *COMPUTER-INTEGRATED TECHNOLOGIES: EDUCATION, SCIENCE, PRODUCTION*. 2025. Vol. 8, No 61. P. 58-63.

3. Ghosh A. *6 GAN Architectures: Everything You Need to Know About Generative Adversarial Networks*. URL: https://neptune.ai/blog/6-gan-architectures (дата звернення: 25.07.2025)

4. Radford A., Kim J. W., Hallacy C. *Learning Transferable Visual Models From Natural Language Supervision (CLIP). URL*: https://arxiv.org/abs/2103.00020 (дата звернення: 28.07.2025)

5. Song J., Meng C., Ermon S. *Denoising Diffusion Implicit Models*. URL: https://scholar.google.com/scholar?q=Denoising+Diffusion+Implicit+Models+Song+Meng+Ermon (дата звернення: 02.08.2025)

6. Liu F., Zhang W. *Text-to-Image Evaluation on MS-COCO Captions*. URL: https://www.researchgate.net/figure/Some-examples-on-MS-COCO-Captions-Caption-description-from-dataset-Predict1_fig1_361298727 (дата звернення: 10.08.2025)

7. Xu J., Li Z., *processes of the diffusion model*. URL: https://www.researchgate.net/figure/The-forward-and-backward-processes-of-the-diffusion-model-The-credit-of-the-used-images_fig1_382128283 (дата звернення: 15.08.2025)

8. NVIDIA Developer Blog.*: Demystifying Diffusion-Based Models*. URL: https://developer.nvidia.com/blog/generative-ai-research-spotlight-demystifying-diffusion-based-models/ (дата звернення: 18.08.2025)

9. Chaphekar N. *Introduction to Stochastic Differential Equations for Score-Based Diffusion Modelling*. URL: https://medium.com/@ninadchaphekar/introduction-to-stochastic-differential-equations-for-score-based-diffusion-modelling-9b8e134f8e2c (дата звернення: 25.08.2025)

10. Wang C., Zhao S. *Diffusion Training: Loss Functions and Convergence Dynamics*. URL: https://www.researchgate.net/figure/Diffusion-Training-a-Training-loss-MSE-per-epoch-averaged-over-all-200-000-training_fig2_392168033 (дата звернення: 29.08.2025)

11. Ho J., Jain A., Abbeel P. *Denoising Diffusion Probabilistic Models*. URL: https://arxiv.org/abs/2006.11239 (дата звернення: 01.09.2025)

12. Radford A., Kim J. W., Hallacy C. *CLIP: Connecting Text and Images*. URL: https://www.researchgate.net/figure/CLIP-consists-of-a-visual-encoder-V-a-text-encoder-T-and-a-dot-product-between-their_fig2_359227413 (дата звернення: 05.09.2025)

13. Chen J., Song Y. *Noise schedules in shifted cosine and linear β families*. URL: https://www.researchgate.net/figure/Some-noise-schedules-in-the-shifted-cosine-family-and-the-linear-b-noise-schedule_fig1_391329262 (дата звернення: 09.09.2025)

14. Rombach R., Blattmann A., Lorenz D., Esser P. *High-Resolution Image Synthesis with Latent Diffusion Models*. URL:https://arxiv.org/abs/2112.10752 (дата звернення: 12.09.2025)

15. NVIDIA Developer Blog. *Generative AI Research Spotlight: Demystifying Diffusion-Based Models*. URL: https://developer.nvidia.com/blog/generative-ai-research-spotlight-demystifying-diffusion-based-models/ (дата звернення: 15.09.2025)

16. Chaphekar N. *Introduction to Stochastic Differential Equations for* URL: https://medium.com/@ninadchaphekar/introduction-to-stochastic-differential-equations-for-score-based-diffusion-modelling-9b8e134f8e2c (дата звернення: 17.09.2025)

17. Wang C., Zhao S. *Convergence Dynamics*. URL: https://www.researchgate.net/figure/Diffusion-Training-a-Training-loss-MSE-per-epoch-averaged-over-all-200-000-training_fig2_392168033 (дата звернення: 20.09.2025)

18. Dosovitskiy A., Beyer L., Kolesnikov A. *An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale* URL: https://scholar.google.com/scholar?q=An+Image+is+Worth+16x16+Words+Dosovitskiy+Beyer+Kolesnikov (дата звернення: 24.09.2025)

19. Paszke A., Gross S., Massa F. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. URL: https://arxiv.org/abs/1912.01703

20. MQTT.org. *MQTT Version 5.0 Specification*. URL: https://mqtt.org/mqtt-specification/ (дата звернення: 01.10.2025)

21. Espressif Systems. *ESP32 Series Datasheet*. URL: https://www.espressif.com/en/products/socs/esp32 (дата звернення: 06.10.2025)

22. Wokwi Online Simulator: ESP32 + OLED project. URL: https://wokwi.com/projects/305568836183130690 (дата звернення: 09.10.2025)