МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

В. о. завідувача кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

ІНТЕЛЕКТУАЛЬНА СИСТЕМА ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ ЗА ТЕКСТОВИМ ОПИСОМ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Інтелектуальні інформаційні системи»

Здобувач _____ Олег СКИБА

«___»_____ 2025 р.

Керівник д-р техн. наук, професор _____ Олександр ГОЖИЙ

«___»_____ 2025 р.

Миколаїв – 2025 р.

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Другий (магістерський)
Освітній ступень	Магістр
Спеціальність	122 Комп'ютерні науки
Освітня програма	Інтелектуальні інформаційні системи

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри інтелектуальних інформаційних систем
_____ Євген СІДЕНКО
«____»_____ 2025 р.

ЗАВДАННЯ

На кваліфікаційну роботу здобувача

Скиба Олег Миколайович

1. Тема кваліфікаційної роботи: Інтелектуальна система генерації зображень за текстовим описом.
Керівник роботи: Гожий Олександр Петрович, професор кафедри, д.т.н., професор.
Затверджена наказом ЧНУ ім. Петра Могили від «07» липня 2025 р. № 184.
2. Строк представлення кваліфікаційної роботи «16» грудня 2025 р.
3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: інтелектуальна система, що генерує зображення на основі текстового опису з використанням трансформерів.

Початкові дані: текстові описи користувача; відкриті датасети зображень; нейронні моделі для генерації.

4. Перелік питань, що підлягають розробці:

– аналіз сучасних методів генерації зображень нейронними мережами та моделей дифузії;

– огляд існуючих технологій для обробки текстових описів природною мовою;

– розроблення архітектури системи для генерації зображень;

– реалізації API для взаємодії з користувачем;

– дослідження та проведення навчання моделі на вибраному датасеті;

– тестування та оцінювання якості роботи інтелектуальної системи.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Олександр ГОЖИЙ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Олег СКИБА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «28» червня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

кваліфікаційної роботи

Тема: Інтелектуальна система генерації зображень за текстовим описом

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	27.06.2025	30.06.2025	
2	Аналіз предметної області та постановка задачі	1.07.2025	20.07.2025	
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема аналіз публічних та аналогічних систам генерації зображень за текстовим описом	21.07.2025	30.07.2025	
4	Огляд існуючих алгоритмів, методів штучного інтелекту для вирішення поставленої задачі	01.09.2025	25.10.2025	
5	Реалізація обраних методів та алгоритмів з аналізом отриманих результатів	26.10.2025	21.11.2025	
6	Перший попередній захист КР на засіданні комісії кафедри	25.11.2025	26.11.2025	
7	Корегування роботи за результатами попереднього захисту	27.11.2025	05.12.2025	
8	Другий попередній захист КР на засіданні комісії кафедри	09.12.2025	09.12.2025	
9	Доробка та остаточне оформлення КР	10.12.2025	11.12.2025	
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.12.2025	16.12.2025	

Керівник роботи _____ Олександр ГОЖИЙ

(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Олег СКИБА

(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«08» липня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи

здобувача групи 601м ЧНУ ім. Петра Могили

Скиби Олега Миколайовича

на тему**: «Інтелектуальна система генерації зображень за текстовим описом»**

Актуальність. У сучасному світі розвиток цифрових технологій відбувається надзвичайно швидкими темпами, що істотно впливає на різні сфери життя людини. Діджиталізація охоплює не лише побутові процеси, а й професійну діяльність у галузях науки, бізнесу, освіти та мистецтва. Одним із напрямів, що активно розвивається останнім часом, є автоматизована генерація мультимедійного контенту, зокрема зображень, на основі текстових описів. Такий підхід дозволяє значно скоротити час і ресурси, які традиційно витрачаються на створення ілюстрацій за участі дизайнерів або художників, та забезпечує можливість отримання якісного результату у найкоротші терміни.

Об'єктом роботи є процес генерації зображень на основі текстового опису.

Предметом роботи є методи, моделі та алгоритми генерації зображень на основі текстового опису.

Мета роботи полягає у підвищенні ефективності методів генерації зображень по текстовому опису.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел та додатків.

У першому розділі було здійснено огляд сучасних підходів до генерації зображень за текстовим описом та ключових технологій, що їх забезпечують – від обробки природної мови до автокодерів і трансформерних моделей. Визначено переваги поєднання трансформерів і автокодерів для моделювання зв'язку між текстовими та візуальними представленнями, а також проаналізовано альтернативні генеративні архітектури.

У другому розділі було розглянуто особливості застосування трансформерів у задачі генерації зображень за текстовим описом. Проаналізовано принципи поєднання мовних і візуальних представлень, роль механізму уваги та компонентів трансформера в моделюванні семантики тексту. Особливу увагу приділено архітектурам VQ-VAE-2 та DALL·E, які інтегрують автокодер і трансформер для створення латентних кодів і синтезу зображень. Розглянуто переваги таких моделей, їх інновації, а також ключові обмеження, пов'язані з обчислювальною складністю та вимогами до даних.

У третьому розділі було розглянуто архітектуру трансформерів, яка сьогодні є основою більшості сучасних моделей глибинного навчання, зокрема систем генерування тексту та зображень. Детально проаналізовано ключові компоненти трансформерів: процес токенізації, побудову ембеддінгів та механізм уваги, що забезпечує здатність моделі враховувати глобальний контекст послідовності. Особливу увагу приділено токенізації як первинному етапу обробки текстових даних, що перетворює речення у послідовність числових представлень. Розглянуто різні підходи до токенізації – від простих методів до складних алгоритмів BPE, WordPiece та SentencePiece – та їх вплив на подальший процес моделювання.

У четвертому розділі було представлено програмну реалізацію інтелектуальної системи генерації зображень за текстовим описом. Система поєднує трансформерний текстовий енкодер, дифузійну модель Stable Diffusion, модуль суперрезолюції та механізми постобробки, що забезпечує отримання детальних і візуально якісних результатів. Завдяки використанню XLM-R, LoRa-адаптації та багатомовної обробки система коректно працює з українськими текстовими запитами та підтримує тонке налаштування під стилі користувача.

В результаті розроблено інтелектуальну систему генерації зображень по текстовому опису.

Кваліфікаційна робота містить 79 сторінок, 35 рисунків, 0 таблиць, 41 використаних джерел та 5 додатків.

Ключові слова: Diffusion Model, Transformers, GAN, U-Net, LoRA, Promt, NLP.

ABSTRACT

to the qualification work by the student of the group 601m of Petro Mohyla Black Sea National University

Skyba Oleh

«INTELLIGENT SYSTEM FOR GENERATING IMAGES BASED ON TEXT DESCRIPTIONS»

Relevance. In today's world, digital technologies are developing at an extremely rapid pace, which has a significant impact on various areas of human life. Digitisation encompasses not only everyday processes, but also professional activities in the fields of science, business, education and art. One of the areas that has been actively developing recently is the automated generation of multimedia content, in particular images, based on text descriptions. This approach significantly reduces the time and resources traditionally spent on creating illustrations with the help of designers or artists, and provides the opportunity to obtain high-quality results in the shortest possible time.

The object of the work is the process of generating images based on text descriptions.

The subject of the work is methods, models and algorithms for generating images based on text descriptions.

The aim of the work is to improve the efficiency of methods for generating images based on text descriptions.

The work consists of a professional section. The explanatory note consists of an introduction, four sections and conclusions.

In the first chapter, an overview of modern approaches to image generation based on text description and the key technologies provided by — from natural language processing to autocoders and transformer models was carried out. The advantages of combining transformers and autocoders to model the relationship between textual and visual representations are identified, and alternative generative architectures are analyzed.

In the second chapter, the peculiarities of the use of transformers in the task of generating images based on a textual description were considered. The principles of combining linguistic and visual representations, the role of the attention mechanism and transformer components in modeling text semantics are analyzed. Special attention is paid to the VQ-VAE-2 and DALL•E architectures, which integrate the autoencoder and transformer to create latent codes and synthesize images. The advantages of such models, their innovations, as well as key limitations related to computational complexity and data requirements are considered.

The third chapter examined the transformer architecture, which today forms the basis of most modern deep learning models, particularly text and image generation systems. The key components of transformers are analyzed in detail: the tokenization process, the construction of embeddings, and the attention mechanism that ensures the model's ability to take into account the global context of the sequence. Special attention is paid to tokenization as the primary stage of text data processing, which turns sentences into a sequence of numerical representations. Different approaches to tokenization of — are considered, from simple methods to complex algorithms of BPE, WordPiece and SentencePiece — and their influence on the further modeling process.

In the fourth chapter, the software implementation of the intelligent image generation system based on the text description was presented. The system combines a transformer text encoder, a Stable Diffusion model, a superresolution module and post-processing mechanisms, which ensures detailed and visually high-quality results. Thanks to the use of XLM-R, LoRa adaptation and multilingual processing, the system works correctly with Ukrainian text requests and supports fine-tuning according to user styles.

As a result, an intelligent system for generating images based on a text description was developed.

The qualification work contains 79 pages, 35 figures, 0 tables, 41 sources used and 5 appendices.

Keywords: Diffusion Model, Transformers, GAN, U-Net, LoRA, Promt, NLP.

ЗМІСТ

ВСТУП

У сучасному світі розвиток цифрових технологій відбувається надзвичайно швидкими темпами, що істотно впливає на різні сфери життя людини. Діджиталізація охоплює не лише побутові процеси, а й професійну діяльність у галузях науки, бізнесу, освіти та мистецтва. Одним із напрямів, що активно розвивається останнім часом, є автоматизована генерація мультимедійного контенту, зокрема зображень, на основі текстових описів. Такий підхід дозволяє значно скоротити час і ресурси, які традиційно витрачаються на створення ілюстрацій за участі дизайнерів або художників, та забезпечує можливість отримання якісного результату у найкоротші терміни.

Особливої актуальності ця проблема набуває в умовах стрімкого розвитку штучного інтелекту та машинного навчання. Застосування методів обробки природної мови (NLP) у поєднанні з глибокими генеративними моделями, такими як генеративно-змагальні мережі (GAN) чи дифузійні моделі, відкриває нові можливості для взаємодії людини з комп’ютером. Основним викликом є необхідність навчити систему адекватно інтерпретувати текстовий опис і трансформувати його у відповідне зображення, що відповідає очікуванням користувача.

Процес генерації зображень за текстовим описом складається з кількох етапів. Спершу текст перетворюється у числове представлення за допомогою моделей векторизації чи трансформерних архітектур, які здатні враховувати семантичні та контекстуальні зв’язки між словами. Далі ці представлення використовуються як вхідні дані для генеративної моделі, що створює візуальне зображення. У випадку генеративно-змагальних мереж генератор намагається побудувати реалістичне зображення за заданим описом, а дискримінатор оцінює його правдоподібність, стимулюючи систему до вдосконалення результату.

Сучасні підходи також передбачають об’єднання текстових та візуальних представлень у спільному латентному просторі, що дає змогу моделі краще узгоджувати опис і створюваний візуальний контент. Така інтеграція значно

підвищує точність та якість кінцевих результатів, а також забезпечує гнучкість у застосуванні технології в різних сферах – від дизайну та реклами до освіти й наукових візуалізацій.

Отже, генерація зображень на основі текстових описів є складною міждисциплінарною задачею, що поєднує досягнення обробки природної мови, комп'ютерного зору та глибокого навчання. Її розвиток відкриває перспективи для створення інтелектуальних систем нового покоління, здатних автоматизувати процес генерації контенту та розширювати можливості взаємодії користувачів з цифровими технологіями.

Об'єкт роботи – процес генерації зображень на основі текстового опису.

Предмет роботи – методи, моделі та алгоритми генерації зображень на основі текстового опису.

Мета кваліфікаційної роботи полягає у підвищенні ефективності методів генерації зображень по текстовому опису.

Для досягнення поставленої **мети** передбачається виконання наступних завдань:

– аналіз сучасних методів генерації зображень нейронними мережами та моделей дифузії;

– огляд існуючих технологій для обробки текстових описів природною мовою;

– розроблення архітектури системи для генерації зображень;

– реалізації API для взаємодії з користувачем;

– дослідження та проведення навчання моделі на вибраному датасеті;

– тестування та оцінювання якості роботи інтелектуальної системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

На сучасному етапі розвитку штучного інтелекту однією з найбільш актуальних задач є автоматизована генерація зображень на основі текстових описів природною мовою. Цей напрямок поєднує методи обробки природної мови (NLP) та комп'ютерного зору, створюючи передумови для інтеграції мовних і візуальних даних у єдиному інформаційному просторі. Основним викликом у даній галузі є не лише технічна коректність побудови зображення, але й ступінь відповідності його змісту текстовому опису, а також різноманітність і креативність згенерованих варіантів.

З точки зору методів глибокого навчання, сьогодні для вирішення цієї задачі широко застосовуються автокодери у поєднанні з трансформерними архітектурами, що забезпечують ефективне навчання моделей на великих мультимодальних наборах даних. Використання трансформерів дозволяє системі враховувати контекст і семантику текстового опису, що підвищує точність відображення змісту у візуальному результаті. Паралельно активно розвиваються підходи на основі дифузійних моделей та генеративно-змагальних мереж (GAN), які демонструють високі показники якості синтезу зображень [2] [30].

Спроби реалізації генерації зображень за текстовим описом також спрямовані на створення архітектур, що комбінують розпізнавальні та генеративні компоненти у спільному просторі представлень [1]. Такий підхід забезпечує тісний взаємозв'язок між мовними та візуальними аспектами даних і дозволяє досягати більшої узгодженості між описом та згенерованим результатом. Це відкриває перспективи для підвищення ефективності інтелектуальних систем, орієнтованих на автоматизоване створення контенту.

1.1 Аналіз предметної галузі

Обраний підхід до генерації зображень на основі текстових описів із використанням автокодерів у поєднанні з трансформерними архітектурами дає змогу створювати візуальний контент, що точно відповідає заданим сценаріям або описам. Така технологія має широкий спектр практичного застосування – від веброзробки та поліграфії до графічного дизайну, реклами та освітніх ресурсів.

Автокодери належать до класу нейронних мереж, які навчаються відтворювати вхідні дані на виході. Вони складаються з двох основних компонентів: енкодера та декодера. Енкодер перетворює вхідні дані у компактне векторне представлення (латентний простір), тоді як декодер відновлює інформацію з цього вектора у вигляді вихідних даних. У контексті генерації зображень автокодери дозволяють створювати реалістичні візуальні об’єкти на основі текстових описів, зберігаючи ключові ознаки відповідності між мовними та візуальними даними [4].

Трансформери, своєю чергою, є архітектурою нейронних мереж, що базується на механізмі уваги та спеціально розроблена для роботи з послідовностями, зокрема текстом. Їхньою основною перевагою є здатність моделювати довгострокові залежності у даних, що робить їх надзвичайно ефективними для обробки природної мови та формування семантично багатих векторних представлень [17].

Процес генерації зображень на основі даного підходу умовно можна поділити на два етапи: етап навчання та етап генерації. На етапі навчання модель отримує пари «текст–зображення», де текстові описи обробляються енкодером і перетворюються у векторні представлення. Ці вектори передаються декодеру, який навчається відтворювати відповідні зображення. У процесі навчання формується зв’язок між текстом та візуальним контентом, що дозволяє системі опановувати принципи генерації [3]. На етапі генерації нові текстові описи подаються на вхід енкодеру, який формує латентне представлення, після чого декодер створює

відповідне зображення. Таким чином забезпечується можливість автоматизованого синтезу візуального контенту за довільними текстовими описами.

Такий підхід поєднує переваги автокодерів у роботі з візуальними даними та трансформерів у моделюванні текстової інформації, забезпечуючи високу ефективність процесу генерації зображень.

1.2 Natural Language Processing в «text-to-image»

У процесі вирішення задачі генерації зображень за текстовим описом ключову роль відіграють методи обробки природної мови (Natural Language Processing). Саме ця галузь штучного інтелекту забезпечує перетворення текстових даних у структуровані векторні представлення, які надалі використовуються генеративними моделями для синтезу зображень. Таким чином, ефективність системи «text-to-image» значною мірою залежить від точності й повноти обробки текстового опису [9].

NLP – це напрям штучного інтелекту, що займається аналізом і моделюванням природної мови за допомогою алгоритмів та моделей машинного навчання. Сфера його застосування охоплює завдання розуміння тексту, генерації нових текстів, машинного перекладу, семантичного аналізу, розпізнавання іменованих сутностей, а також визначення емоційного тону (сентимент-аналіз). Кожен із цих підходів дозволяє глибше інтерпретувати зміст тексту, що безпосередньо впливає на якість відтворення візуальних образів[9].

Особливе значення для задачі генерації зображень має семантичний аналіз, оскільки саме він дозволяє системі розуміти зміст опису, враховувати контекст і коректно співвідносити мовні елементи з візуальними ознаками. Водночас синтаксичний аналіз допомагає структурувати текст, що важливо для точного відображення відносин між об’єктами в зображенні. Таким чином, NLP виступає проміжною ланкою між текстом і генеративною моделлю.

Для побудови інтелектуальних систем «text-to-image» сьогодні широко використовуються нейронні мережі, зокрема трансформерні архітектури. Вони

мають унікальну здатність моделювати довготривалі залежності у тексті завдяки механізмам уваги, що дозволяє системі враховувати як локальні, так і глобальні зв’язки між словами. Використання трансформерів у поєднанні з генеративними моделями (GAN, автокодерами або дифузійними моделями) забезпечує високу якість синтезованих зображень та їхню відповідність початковим текстовим описам[30].

Отже, NLP є фундаментальною складовою в системах генерації зображень за текстом, оскільки саме воно забезпечує адекватне відображення мовної інформації у візуальному просторі та створює основу для роботи генеративних моделей.

1.3 Автокодери

Автокодери належать до класу нейронних мереж, основним завданням яких є відтворення вхідних даних на виході. Їхня архітектура складається з двох ключових компонентів – енкодера та декодера. Енкодер приймає вхідні дані та перетворює їх у компактне векторне представлення (латентний код), тоді як декодер реконструює вихідні дані з цього коду (рисунок 1.1). У задачах генерації зображень автокодери можуть бути навчальні на парах «текст–зображення», що дозволяє відтворювати візуальний контент, максимально наближений до заданого опису[4].

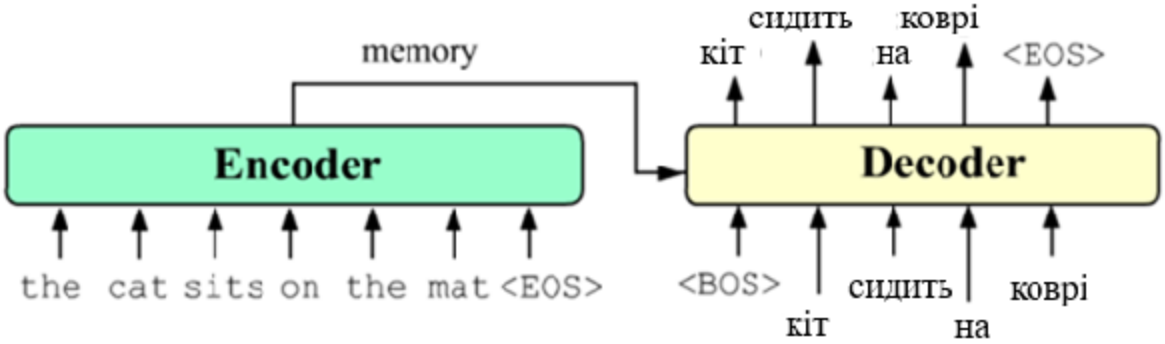

Рисунок 1.1 – Робота encoder та decoder

Основна мета автокодерів полягає у знаходженні ефективного відображення даних у прихованому просторі. Під час навчання енкодер виконує перетворення вхідних даних (x) у векторне представлення (z):

$$z = f_{enc}(x) \tag{1.1}$$

а декодер відновлює наближене відтворення даних (x') з використанням латентного коду:

$$x' = f_{dec}(z) \tag{1.2}$$

Навчання автокодера відбувається за принципом мінімізації функції втрат, яка обчислює різницю між вхідними даними (x) та їх відтворенням (x'). Для цього можуть застосовуватися різні функції втрат, зокрема середньоквадратична помилка (MSE) або перехресна ентропія. Оптимізація параметрів виконується за допомогою алгоритмів градієнтного спуску та його модифікацій [6].

Завдяки своїй архітектурі автокодери мають широкий спектр застосувань: стиснення та відновлення даних, виділення ознак, виявлення аномалій, а також генерація нових прикладів даних. У сфері генерації зображень вони часто використовуються як базові моделі або як складові частини більш складних архітектур, наприклад, у комбінації з трансформерами чи генеративно-змагальними мережами. Це дозволяє поєднувати можливості автокодерів у роботі з латентними просторами із здатністю інших моделей забезпечувати креативність та високу якість зображень.

У сучасних дослідженнях активно застосовуються різні модифікації автокодерів – згорткові, рекурентні та трансформерні, що дозволяє адаптувати їх до специфічних типів даних та підвищувати якість генерації зображень на основі текстових описів [4].

1.4 Трансформери

Трансформери – це архітектура нейронних мереж, що ґрунтується на механізмах уваги та орієнтована на ефективну обробку послідовностей даних, зокрема тексту. Вперше вони були представлені у роботі «Attention is All You Need» у 2017 році та з того часу стали фундаментом сучасних рішень у сфері обробки природної мови. Завдяки своїй гнучкості трансформери успішно застосовуються й у задачах генерації зображень за текстовим описом (text-to-image) [10].

Головна перевага трансформерів полягає у здатності моделювати довготривалі залежності між елементами тексту. Це особливо важливо при генерації зображень, оскільки модель повинна враховувати не лише окремі слова, а й їхні зв’язки та контекст у цілому.

Основні компоненти трансформера включають:

– механізм уваги (Attention Mechanism): дозволяє моделі виділяти ключові частини тексту, що мають найбільше значення для формування зображення;

– кодувальні та декодувальні шари (Encoder і Decoder): кодувальні блоки перетворюють текстовий опис у векторні представлення, а декодувальні блоки використовують ці представлення для генерації зображення;

– позиційне кодування (Positional Encoding): додає інформацію про порядок слів, що необхідно для збереження структури тексту;

– генеративний декодер (Generative Decoder): виконує побудову зображення, враховуючи ключові слова та фрази за допомогою механізму уваги.

До основних типів шарів у трансформерах відносяться (рис. 1.2):

– Embedding Layer – перетворює токени у векторні представлення. Це дозволяє моделі представляти слова у просторі з низькорозмірини векторами;

– Positional Encoding Layer – враховує порядок слів у послідовності. Він розв’язує проблему безпосереднього врахування порядку слів у трансформерах;

– Multi-Head Attention Layer – моделює взаємозв’язки між словами. Він обчислює ваговану суму значень вкладення для кожного слова;

– Feedforward Layer – забезпечує незалежну обробку представлень кожного слова. Він застосовує повністю зв’язаний шар до кожного позиційного вектора окремо, що дозволяє моделі незалежно обробляти кожне слово в реченні;

– Normalization Layer – стабілізує процес навчання для кожного з вище перерахованих шарів та прискорює збіжність моделі.

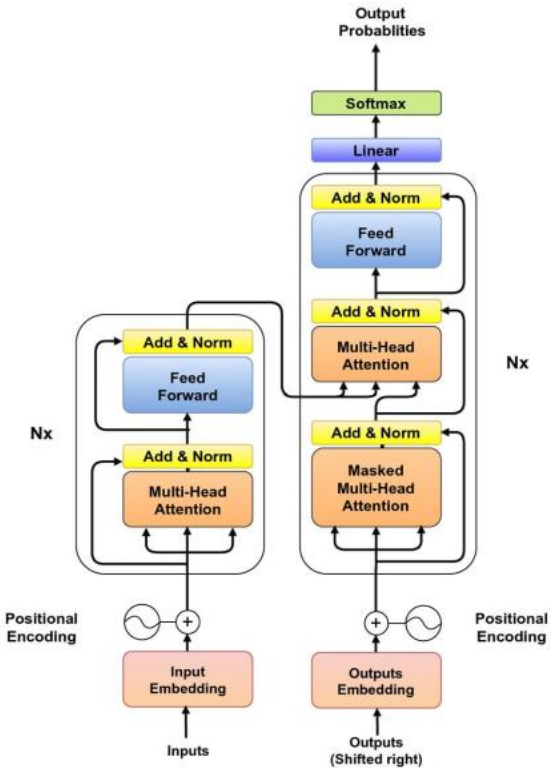

Рисунок 1.2 – Архітектура трансформера

Додаткові модифікації архітектури (наприклад, block-level attention чи спеціалізовані з’єднання) дозволяють підвищувати якість моделювання залежно від конкретних задач.

У сучасних системах генерації зображень за текстом трансформери використовуються як ключовий елемент архітектури. Серед найвідоміших прикладів можна відзначити DALL·E, Artbreeder, Runway ML, DeepAi Text to Image API та інші рішення, які демонструють високий рівень реалістичності та різноманітності результатів [12].

1.5 Generative Adversarial Network

Генеративно-суперницькі мережі (GAN, Generative Adversarial Networks) – це клас нейронних мереж, що складаються з двох взаємодоповнюючих компонентів: генератора та дискримінатора, які змагаються між собою у процесі навчання. Такий підхід до генерації даних був запропонований у 2014 році в роботі «Generative Adversarial Networks» і швидко став однією з ключових технологій для синтезу зображень, тексту, аудіо та інших типів даних [2].

Принцип роботи GAN ґрунтується на протиставленні двох моделей – генератор створює нові дані (наприклад, зображення) на основі випадкового шуму або векторного представлення. Його завдання – виробити такі приклади, які максимально наближені до реальних. Дискримінатор виконує роль «судді», розрізняючи справжні дані з навчального набору та синтетичні, створені генератором. Його мета – мінімізувати ймовірність помилкової класифікації.

У процесі навчання обидві мережі вдосконалюються: генератор навчається обманювати дискримінатор, а дискримінатор – краще розпізнавати підробки. Такий змагальний процес призводить до поступового підвищення якості синтезованих даних.

Функція втрат у GAN одночасно використовується для обох компонентів. Генератор – нейронна мережа, що призначена для генерації нових даних, що схожі на ті, що містяться в навчальному наборі та прагне мінімізувати ймовірність того, що його результати будуть класифіковані як синтетичні. Генератор має 5 шарів. Він приймає на вході випадковий шум чи інші векторні представлення і перетворює це в зображення, текст або інші дані [35].

Дискримінатор – нейронна мережа, яка використовується для класифікації справжніх або синтетичних вхідних даних та прагне максимізувати точність їхнього розпізнавання. Дискримінатор також складається з 5 шарів. Він навчається розрізняти між справжніми та синтетичними даними.

Сьогодні GAN широко застосовуються у сфері генерації зображень за текстовим описом. Однією з найвідоміших моделей цього типу є DALL·E,

розроблена OpenAI. Вона здатна створювати унікальні й креативні зображення навіть за складними та нестандартними текстовими запитами.

Архітектура DALL·E включає генератор, який перетворює текстові описи у відповідні зображення. Відомо, що модель використовує комбінацію трансформерних блоків та механізмів уваги, що забезпечує ефективну взаємодію між текстом і зображенням. Попри те, що всі технічні деталі архітектури не були повністю розкриті, відомо, що DALL·E вирізняється високим рівнем узгодженості та творчості при побудові візуального контенту [17].

Особливістю цієї моделі є здатність до генерації не лише реалістичних, а й фантазійних сцен. Наприклад, вона може створити «будинок у формі гігантського крокодила» або «кішку з тіста» на основі звичайного текстового опису. Це робить DALL·E прикладом креативного застосування GAN і трансформерів у сфері text-to-image.

1.6 Порівняння архітектури трансформерів та GAN

Трансформери та генеративно-суперницькі мережі (GAN) належать до сучасних архітектур нейронних мереж, проте кожна з них має свої переваги та сфери застосування. Трансформери виявилися особливо ефективними у завданнях обробки природної мови, зокрема у машинному перекладі, генерації тексту та його семантичному аналізі[13]. Їхня головна перевага полягає у здатності моделювати довготривалі залежності у послідовностях, що дозволяє враховувати контекст навіть на великих відстанях між словами. Завдяки використанню механізму уваги трансформери здатні ефективно працювати з різними типами даних, а їх архітектура дозволяє здійснювати обробку послідовностей паралельно [2]. Це забезпечує вищу швидкість навчання та генерації у порівнянні з GAN, де процес часто є більш обмеженим і пов’язаний із роботою з даними фіксованої розмірності.

Ще однією важливою характеристикою трансформерів є відносна простота навчання. На відміну від GAN, які потребують ретельного налаштування гіперпараметрів та застосування спеціальних методик стабілізації, трансформери

демонструють більш передбачувану та стабільну збіжність під час тренування. Це робить їх доступнішими для широкого кола завдань, включаючи не лише роботу з текстом, але й із зображеннями та іншими видами даних.

Водночас GAN залишаються надзвичайно цінними в тих випадках, коли критичною є реалістичність та деталізація згенерованого контенту. Їхня унікальна особливість полягає у конкуренції між генератором і дискримінатором, що дозволяє досягати високого рівня правдоподібності результатів. Це особливо помітно у завданнях генерації зображень, де GAN здатні створювати візуально насичені та правдоподібні приклади. Проте їх використання пов’язане з низкою обмежень: складністю навчання, можливістю виникнення модального колапсу та появою шумів у згенерованих зображеннях.

Таким чином, кожна з розглянутих архітектур має свої сильні сторони та слабкі місця. Трансформери краще підходять для роботи з текстовими та послідовнісними даними завдяки універсальності та стабільності навчання, тоді як GAN залишаються важливим інструментом у тих сферах, де на перший план виходить реалістичність і виразність синтезованого контенту. Вибір між цими архітектурами визначається конкретними вимогами задачі, характеристиками навчальних даних та ресурсними можливостями.

1.7 Механізм самоуваги в трансформерах

Механізм самоуваги (self-attention) є центральним елементом архітектури трансформерів і виконує ключову роль у їх здатності працювати з послідовними даними. Його основне завдання полягає у тому, щоб надати моделі можливість звертати увагу на різні частини вхідної послідовності при формуванні кожного елемента виходу. Завдяки цьому модель враховує як локальні, так і глобальні залежності між словами чи іншими об’єктами, що робить самоувагу основою успіху трансформерів у завданнях обробки природної мови та інших сферах, де важливо працювати з контекстом [10].

Функціонування цього механізму ґрунтується на трьох основних компонентах:

– запити (Queries) – визначають, на яку інформацію слід звернути увагу;

– ключі (Keys) – допомагають співвідносити запити з релевантними елементами;

– значення (Values) – містять фактичну інформацію, що використовується для обчислень.

Кожен елемент вхідної послідовності перетворюється в ці три вектори за допомогою окремих лінійних перетворень. Далі виконується обчислення ваг уваги. Для цього кожен запит порівнюється з усіма ключами за допомогою скалярного добутку, а отримані результати нормалізуються за допомогою функції softmax. Така операція дозволяє визначити, які частини вхідних даних є найбільш релевантними для поточного елемента.

$$Attenction(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (1.3)$$

де Q – матриця запитів;

K – матриця ключів;

V – матриця значень;

d_k – розмірність ключів.

Щоб уникнути занадто великих значень і стабілізувати навчання, скалярний добуток додатково масштабується на $\sqrt{d_k}$ (де d_k — розмірність ключів). Цей підхід відомий як масштабована точкова продукція (Scaled Dot-Product Attention).

Ще одним важливим розширенням є мультиголовий механізм уваги. Він передбачає, що одночасно використовується кілька «голів» уваги, кожна з яких формує власні ваги та результати. Потім усі вони об’єднуються і проходять через додатковий лінійний шар. Такий підхід дозволяє моделі навчатися одразу різним типам залежностей між елементами послідовності та вловлювати ширший спектр інформації.

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O \quad (1.4)$$

де $head_i$ = Attention(QW_i^Q, KW_i^K, VW_i^V);

W^O – лінійний шар для об'єднання результатів.

Переваги механізму самоуваги пояснюють його популярність і широке застосування:

– паралельна обробка – на відміну від рекурентних мереж, self-attention дозволяє одночасно працювати з усіма елементами, що значно пришвидшує навчання;

– гнучке врахування контексту – модель може фокусуватися на будь-яких частинах послідовності незалежно від їх відстані;

– масштабованість – архітектура трансформерів легко застосовується до великих наборів даних і може розширюватися до моделей із мільярдами параметрів.

Механізм самоуваги є критичним компонентом трансформерів, забезпечуючи ефективну та гнучку обробку послідовностей даних, що робить їх надзвичайно потужними для різних задач у галузі обробки природної мови та поза нею [10].

Практичне застосування механізму самоуваги охоплює як завдання обробки природної мови, так і інші галузі штучного інтелекту [21]. Наприклад:

– у машинному перекладі він використовується в оригінальній моделі Transformer – один з найвідоміших прикладів, де механізм самоуваги використовується для покращення перекладу тексту з однієї мови на іншу. Він замінив рекурентні нейронні мережі та LSTM завдяки здатності моделювати довготривалі залежності;

– у генерації тексту механізм реалізований у моделях GPT, які створюють зв'язний та осмислений текст на основі заданого контексту. GPT-5 є більш потужна

версія GPT, що використовує глибші архітектури та більші набори даних для покращення якості та креативності генерації тексту;

– у аналізі тональності самоувага застосовується в архітектурі BERT, що враховує контекст одночасно зліва і справа від слова, завдяки чому досягається висока точність класифікації;

– у резюмуванні текстів цей механізм використовують моделі BERTSUM і PEGASUS, які виділяють ключову інформацію для створення узагальнених викладів. BERT застосовує механізм самоуваги для вибору найважливіших частин тексту, що повинні бути включені в резюме. PEGASUS використовує механізм самоуваги для розуміння контексту та виділення ключових речень в тексті;

– у відповідях на запитання самоувага допомагає моделям BERT, RoBERTa які використовують самоувагу для розуміння запитання та знаходження відповідної інформації в тексті. Вони можуть точно відповісти на питання, базуючись на контексті наданого тексту. Також даний механізм використовується в T5 для різних NLP задач, включаючи відповіді на питання, де самоувага допомагає враховувати контекст запитання та тексту;

– у комп'ютерному зорі вона лежить в основі архітектури Vision Transformer, що обробляє зображення як послідовність фрагментів і використовує самоувагу для взаємодії між цими фрагментами. Це допомагає моделі краще класифікувати зображення та проводити їх сегментацію;

– у мультимодальних завданнях її застосовує модель CLIP, яка навчається на спільному представленні текстових і візуальних даних, дозволяючи виконувати завдання пошуку зображень за текстом.

Таким чином, механізм самоуваги є надзвичайно універсальним і потужним інструментом. Він забезпечує ефективну інтеграцію інформації з послідовних даних і відкрив шлях до створення сучасних трансформерних архітектур, які сьогодні визначають розвиток багатьох напрямів штучного інтелекту.

1.8 Постановка задачі

Метою даної роботи є підвищення ефективності методів генерації зображень на основі текстових описів із використанням автокодера, інтегрованого з трансформерною архітектурою. У процесі роботи необхідно проаналізувати існуючі методи синтезу зображень, визначити переваги і недоліки відповідних моделей, а також дослідити механізми взаємодії мовних і візуальних представлень.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– провести аналіз предметної галузі та сучасних методів генерації зображень за текстовим описом, включаючи автокодери, трансформери, генеративно-суперницькі мережі та дифузійні моделі;

– дослідити підходи обробки природної мови (NLP), які використовуються для формування семантичних представлень тексту в задачах «text-to-image»;

– проаналізувати архітектуру автокодерів та можливість їх інтеграції з трансформерними моделями для кодування текстових описів;

– дослідити механізм уваги та його роль у формуванні відповідності між текстом і зображенням;

– розробити або адаптувати модель автокодера на базі трансформера та здійснити її навчання на вибірці типу «text–image»;

– оцінити якість роботи моделі, проаналізувати отримані результати та визначити переваги і можливі напрями покращення генерації зображень.

Постановка задачі полягає у створенні та дослідженні моделі, здатної генерувати зображення на основі текстових описів, використовуючи комбінований підхід на базі автокодерів і трансформерних архітектур.

Висновки до розділу 1

Було здійснено огляд сучасних підходів до генерації зображень за текстовим описом та ключових технологій, що їх забезпечують – від обробки природної мови до автокодерів і трансформерних моделей. Визначено переваги поєднання трансформерів і автокодерів для моделювання зв’язку між текстовими та

візуальними представленнями, а також проаналізовано альтернативні генеративні архітектури. Підсумком аналізу стала постановка задачі дипломної роботи та формування теоретичного підґрунтя для подальшої розробки й дослідження моделі text-to-image.

2 ТРАНСФОРМЕРИ В ЗАДАЧІ «TEXT-TO-IMAGE»

2.1 Особливості застосування трансформерів в моделях генерації зображень

У сучасному світі значна кількість творчих і технічних процесів автоматизується завдяки стрімкому розвитку штучного інтелекту. Зокрема, сфера комп'ютерної графіки, дизайну та мультимедіа все активніше використовує нейронні мережі для створення візуального контенту. Раніше процес розробки ілюстрацій, логотипів або концепт-артів вимагав виключно людського креативу, тоді як сьогодні генеративні моделі здатні створювати нові, унікальні зображення лише за текстовим описом. Такі можливості забезпечуються завдяки архітектурам трансформерів, які стали фундаментом сучасних систем типу «text-to-image» [26].

Трансформери відіграють ключову роль у задачах генерації зображень на основі текстових описів, оскільки забезпечують тісну інтеграцію між мовними та візуальними даними. Вони дозволяють моделі «розуміти» контекст запиту, структурувати зміст опису та перетворювати його у візуальні ознаки, що визначають майбутнє зображення.

Основна архітектура трансформера складається з блоків самоуваги та пропускних лінійних шарів (Feed-Forward Network), які забезпечують ефективну роботу з послідовними даними. У контексті задач «text-to-image» це означає глибоке розуміння тексту, його структури та змісту, що безпосередньо впливає на якість згенерованого зображення [27].

Основні компоненти трансформера:

– енкодинг тексту. Трансформери перетворюють текстовий опис у векторні представлення, що містять семантичну інформацію. Завдяки цьому модель може розуміти контекст, ключові об'єкти та їхні властивості;

– механізм уваги. Забезпечує фокусування на найважливіших частинах тексту під час створення зображення. Це дозволяє моделі виділяти важливі деталі – форму об'єктів, кольори, просторове розташування тощо;

– генератор зображень. Використовує отримані векторні представлення тексту для побудови візуального зображення. Зазвичай базується на трансформерній архітектурі, доповненій спеціалізованими модулями для відтворення складних візуальних ознак;

– мультиголовий механізм уваги. Дозволяє моделі одночасно враховувати кілька різних аспектів тексту, що підвищує точність і деталізацію зображення;

– об’єднання текстових і візуальних представлень. У сучасних моделях, таких як DALL·E або Imagen, застосовується інтеграція двох типів даних у спільному семантичному просторі. Це забезпечує двосторонній зв’язок між текстом і зображенням, підвищуючи узгодженість між описом та результатом.

Однією з перших і найвідоміших моделей цього напрямку є DALL·E, розроблена компанією OpenAI. Архітектура DALL·E базується на поєднанні трансформерів і векторних квантованих автокодерів (VQ-VAE-2), що дозволяє здійснювати якісне перетворення тексту в зображення.

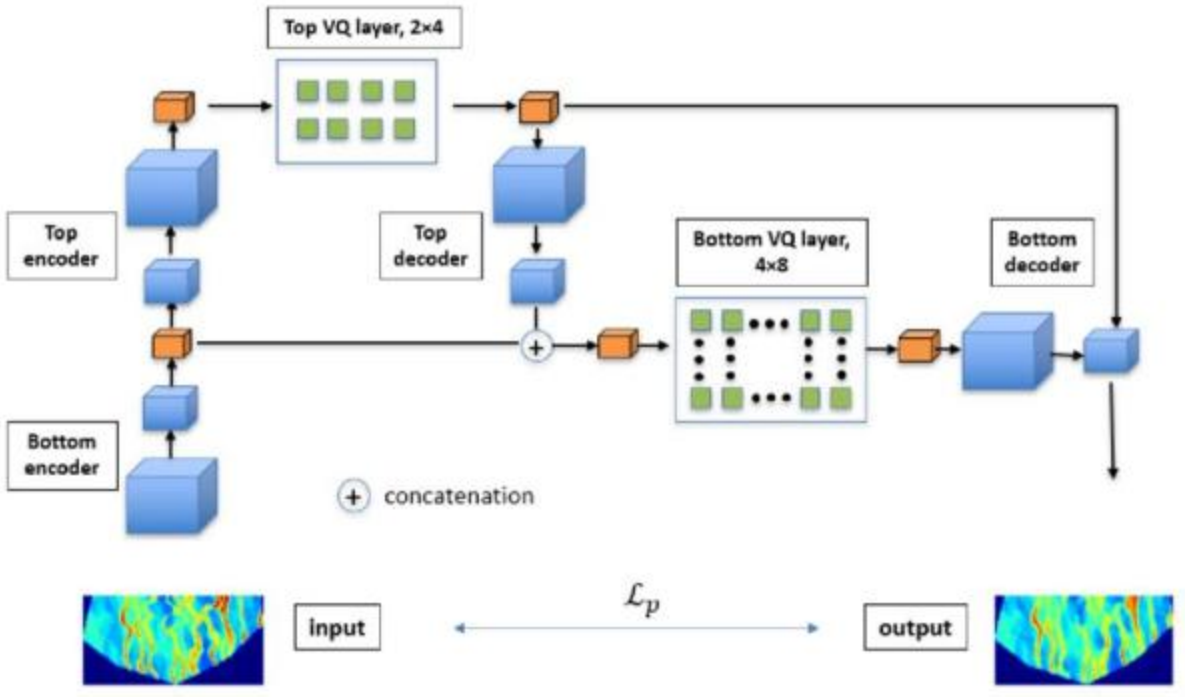

Рисунок 2.1 – Архітектура VQ-VAE-2

Звичайно повністю архітектуру DALL-E моделі розробники не розкривають, але основні моменти деякі все ж таки відомо. Модель, розроблена OpenAI, використовує трансформери для перетворення текстового опису у векторні представлення, які потім використовуються для генерації зображень. DALL-E демонструє здатність створювати високоякісні зображення, враховуючи складні текстові запити. Модель VQVAE-2, яка поєднує варіаційні автокодери з трансформерами. Вона кодує зображення у векторні представлення і використовує трансформери для обробки тексту, забезпечуючи синтез нових зображень на основі текстового опису [1].

Основні компоненти архітектури DALL-E:

– енкодер тексту. Отримує текстовий опис і перетворює його на послідовність векторів, подібно до моделей сімейства GPT;

– векторно-квантований автокодер (VQ-VAE-2). Виконує кодування зображень у компактні латентні представлення, які потім можуть бути відновлені у вигляді повноцінних зображень;

– трансформер генерації. Основний компонент моделі, який навчається передбачати латентні коди зображення на основі текстового опису, формуючи послідовність кодів для подальшої декомпресії.

Латентні коди є внутрішніми представленнями даних, які відображають стиснуту, але змістовну інформацію про вхідні зображення. Вони дозволяють моделі навчитися ефективно працювати з великими обсягами візуальної інформації.

Основні характеристики латентних кодів:

– стиснення даних: латентні коди являють собою стислий варіант вхідних даних. Наприклад, зображення високої роздільності представляється компактним вектором;

– збереження суттєвих ознак: у коді зберігаються ключові властивості зображення, наприклад, форми, кольори та інші візуальні характеристики;

– можливість варіацій: зміна латентних кодів дозволяє створювати нові, унікальні зображення на основі одного текстового опису.

Розглянемо приклад роботи з латентними кодами у VAE:

– енкодер: енкодер приймає вхідні дані (наприклад, зображення) і перетворює їх на латентний код. Цей код є вектором у латентному просторі;

– декодер: декодер приймає латентний код і відновлює з нього вхідні дані. В ідеалі, відновлені дані мають бути якомога ближче до оригіналу;

– варіаційний підхід: у варіаційних автокодерах (VAE) енкодер не просто створює один латентний код, а генерує параметри розподілу (середнє значення і дисперсію). З цього розподілу потім вибираються конкретні коди для декодування, що забезпечує варіативність у генерації нових даних.

Розглянемо варіанти використання латентних кодів у DALL-E:

– кодування зображень. DALL-E використовує векторноквантований автокодер (VQ-VAE-2) для кодування зображень у латентні коди. Ці коди містять стиснуту інформацію про зображення і можуть бути ефективно використані для генерації;

– генерація зображень за текстовим описом. Текстовий опис перетворюється на векторне представлення за допомогою трансформера. Потім модель трансформера генерує латентні коди зображення на основі цього векторного представлення тексту;

– відновлення зображення. Латентні коди, створені трансформером, подаються на вхід декодеру VQ-VAE-2, який відновлює зображення з цих кодів. В результаті, модель DALL-E створює нові зображення, відповідні текстовим описам.

Латентні коди є критично важливими для генеративних моделей, оскільки вони дозволяють стискати дані (латентні коди зменшують розмір даних, що полегшує їх обробку), генерувати нові приклади (латентні коди можна використовувати для створення нових зображень або інших даних, які схожі на оригінальні, але мають деякі нові властивості), забезпечувати варіативність (моделі

можуть створювати різні варіанти зображень або інших даних, змінюючи латентні коди).

Латентні коди відіграють ключову роль у багатьох сучасних генеративних моделях, забезпечуючи ефективне кодування та декодування складних даних, таких як зображення.

Процес роботи DALL-E можна розділити на кілька етапів:

– спочатку відбувається навчання автокодера: автокодер VQ-VAE-2 навчається на великому наборі зображень. Він вчиться кодувати зображення у латентні вектори (кванти) та відновлювати зображення з цих векторів. Це забезпечує компактне представлення зображень, яке можна використовувати для генерації;

– далі проводиться навчання трансформера: трансформер навчений на парі текстових описів і відповідних латентних кодів зображень. Використовуючи механізм уваги, трансформер навчається передбачати послідовність латентних кодів, які відповідають заданому текстовому опису;

– заключним етапом є генерація зображень: під час генерації зображення текстовий опис подається на вхід трансформеру, який генерує відповідні латентні коди. Ці коди потім подаються на вхід декодеру VQ-VAE-2, який відновлює зображення з цих кодів [20].

Модель DALL-E була однією з перших моделей по генерації зображень по текстовому опису, а отже саме вона і внесла в розвиток цього напрямку деякі інновації. Розглянемо переваги та інновації DALL-E:

– гнучкість у генерації: DALL-E здатний створювати як зображення з широким спектром текстових описів, включаючи як реалістичні, так і абстрактні концепції та конкретні деталі;

– висока якість результату: поєднання VQ-VAE-2 і трансформерної архітектури забезпечує високу якість згенерованих зображень, оскільки автокодер здатен зберігати важливі деталі зображення у своїх латентних представленнях;

– глибока інтеграція модельності: трансформери добре справляються з обробкою послідовностей і можуть ефективно інтегрувати текстову інформацію, враховуючи контекст та деталі опису.

Якою б не була модель продуманою, інноваційною, все одно є певні недоліки. Розглянемо недоліки моделі DALL-E більш детально:

– висока обчислювальна складність. Навчання таких моделей вимагає значних обчислювальних ресурсів, що може бути викликом для менших організацій;

– потреба у великих обсягах даних. Для навчання DALL-E потрібні великі набори даних пар «текст-зображення», що також може бути обмеженням;

– недостатній контроль над результатом. Генеровані зображення можуть іноді не повністю відповідати текстовому опису або містити артефакти, що вимагає подальшого вдосконалення моделей.

Висновки до розділу 2

Використання трансформерів у задачах генерації зображень є одним із найперспективніших напрямів розвитку штучного інтелекту. Їхня здатність моделювати взаємозв’язок між текстом і зображенням відкриває нові можливості для автоматизації творчих процесів і підвищення ефективності генеративних методів.

3 РОЗРОБКА АРХІТЕКТУРИ СИСТЕМИ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ

Трансформери (Transformers) – це сучасна архітектура нейронних мереж, що суттєво підвищила ефективність моделей машинного навчання, особливо у завданнях обробки природної мови (NLP), генерації тексту та розпізнавання зображень. Дана архітектура була вперше представлена у 2017 році дослідниками компанії Google у науковій роботі «Attention Is All You Need», яка започаткувала новий етап розвитку штучного інтелекту. Найкращим прикладом роботи трансформерів є речення, так як воно складається з впорядкованого набору слів [10].

Трансформери створюють цифрове уявлення кожного елемента послідовності, інкапсулюють важливу інформацію про нього і навколишній контекст. Основна ідея трансформерів полягає в тому, що вони працюють із послідовними даними, де кожен елемент має власне значення, але також залежить від контексту інших елементів. Це зроблено для подальшої можливості цих нейронних мереж скористаються цією інформацією задля розв'язання певних задач, зокрема для синтезу і класифікації. Отримані представлення можуть надалі використовуватися іншими нейронними мережами для розв’язання різних завдань – зокрема класифікації, синтезу або генерації даних. Таким чином, трансформери допомагають ефективно виявляти приховані закономірності та взаємозв’язки у вхідних даних, що робить їх надзвичайно потужними у задачах, де необхідно аналізувати та створювати складні послідовності – текстові, звукові або візуальні [17].

Головною перевагою трансформерів є здатність обробляти послідовності даних паралельно, що досягається завдяки механізму уваги (Attention Mechanism), зокрема багатоголовій самоувазі (Multi-Head Self-Attention). Це забезпечує значні переваги порівняно з попередніми архітектурами, такими як рекурентні нейронні мережі (RNN) та довго-короткочасна пам’ять (LSTM). Основні переваги можна узагальнити наступним чином:

Паралелізація:

– ефективність обчислень: трансформери дозволяють одночасно обробляти всі елементи послідовності, що значно скорочує час навчання моделей;

– використання апаратного прискорення: архітектура трансформерів чудово масштабуються на GPU та TPU, що робить можливим тренування моделей з мільярдами параметрів.

Контекст та довгострокові залежності:

– глобальний контекст: механізм уваги дозволяє враховувати всі елементи послідовності при формуванні кожного вихідного вектора, завдяки чому модель «розуміє» зміст усієї фрази, а не лише окремих слів;

– довгострокові залежності: трансформери ефективно зберігають зв’язки між далекими елементами тексту, чого не могли досягти класичні RNN через втрату контексту при довгих послідовностях.

Гнучкість та узагальнюваність:

– універсальність: трансформери застосовуються не лише для NLP, але й у генерації тексту, машинному перекладі, розпізнаванні зображень, створенні аудіо та навіть генерації відео;

– масштабованість: ця архітектура дозволяє будувати надзвичайно великі моделі, такі як GPT-3 (OpenAI) або BERT (Google), які містять мільярди параметрів і демонструють високий рівень узагальнення [11].

Розглянемо приклади застосування трансформерів:

– BERT (Bidirectional Encoder Representations from Transformers) — модель, що використовується для розуміння тексту: аналізу настроїв, пошуку інформації, відповідей на запитання та розпізнавання сутностей [21];

– GPT (Generative Pre-trained Transformer) — архітектура, орієнтована на генерацію тексту, створення діалогових систем, машинний переклад та узагальнення інформації [35].

Перед подачею текстових даних у трансформер необхідно виконати процес токенізації, тобто розбиття тексту на окремі складові – токени. Токени – це основні

елементи, на які розбивається текст при обробці природної мови (NPL). Токенізація – це перетворення тексту у список базових елементів (слів, частин слів або символів), які модель може обробляти чисельно. Кожен токен може бути словом, частиною слова, символом або групою символів залежно від завдання та методів токенізації, що використовуються [22].

Види токенів:

– слова (Word Tokens): кожне слово є окремим токеном. Наприклад, у реченні «The cat sat on the mat» кожне слово розглядається як окремий токен → [«The», «cat», «sat», «on», «the», «mat»];

– підслова або морфеми (Subword Tokens): слова розділяються на менші частини або морфеми, особливо у мовах з великою кількістю словоформ, для гнучкішої обробки. Наприклад, слово «unhappiness» може бути розділене [«un», «happiness»];

– символи (Character Tokens): кожен символ тексту є окремим токеном. Це корисно для мов з великим алфавітом або для оброки текстів з помилками або нестандартним написанням. Наприклад слово «hello» може бути токенізовано як [«h», «e», «l», «l», «o»].

Методи токенізації:

– Пробіл-розділена токенізація (Whitespace Tokenization): розбиття тексту на слова за пробілами. Це є найпростіший метод, але він не враховує пунктуацію та інші символи;

– токенізація за регулярними виразами (Regex Tokenization): використання регулярних виразів для точнішого виділення слів із урахуванням пунктуації;

– Byte-Pair Encoding (BPE): це алгоритм, який об’єднує часто вживані пари символів у нові токени, що робить його ефективним при роботі з рідкісними словами. Він дуже часто використовується у трансформерах;

– WordPiece: подібний до BPE, але використовує інші алгоритми для створення токенів. Зазвичай застосовується в моделях типу BERT;

– SentencePiece: універсальний метод, що не потребує попередньої обробки тексту, тому підходить для багатомовних моделей і задач [22].

Застосування токенів у моделях машинного навчання:

– у моделях NLP: токени виступають основними вхідними даними для моделей, як трансформери, які навчаються на послідовностях токенів;

– під час векторизації: токени перетворюються у числові вектори (ембеддінги), які використовуються для подачі на вхід нейронної мережі;

– для аналізу тексту: токени використовуються для різних задач аналізу тексту, як частотного аналізу, пошуку ключових слів або машинного перекладу.

Простий приклад роботи з трансформерами та токенами є задачі перетворення певної послідовності токенів, використовуючи словник, який буде відігравати роль таблиці для пошуку[8]. Необхідно зіставити між собою слова та числа, адже можна співставити будь-яке слово з будь-яким числом (рис. 3.1).

```
Raw text: A black cat
Vocab: {"A": 0, "black": 1, "cat": 2, "cats": 3}
Tokenized text: [0, 1, 2]
```

Рисунок 3.1 – Приклад кодування тексту

На основі даного прикладу можна помітити, що слова cats та cat було закодовано різними токенами, незважаючи на те, що це одне і те слово, але у різній формі множини.

Також існують вже більш продвинуті способи токенізації при якій перед тим як присвоїти індекси словам, їх розбивають на фрагменти. В ході експериментів також була помічена зручність у використання окремих токенів, які б позначали кінець та початок речень, адже це надає більше контексту [8].

Розглянемо такий приклад на токенізації представленого речення: «Hello there, isn’t the weather nice today in Drosval?». Використовуючи токенізатор bert-

base-uncased з бібліотеки transformers library, речення перетворюється у послідовність числових токенів (рис. 3.2).

```
[101, 7592, 2045, 1010, 3475, 1521, 1056, 1996, 4633, 3835, 2651, 1999, 2852, 2891, 10175, 1029, 102]
```

Рисунок 3.2 – Перетворення речення на послідовність токенів

Числа, які були присвоєно кожному слову будуть змінюватись в залежності від обраного способу токенізації та від методу навчання моделі.

Після зворотного декодування отриманні токени повертаються у слова (рис. 3.3).

```
[CLS] hello there , isn ’ t the weather nice today in dr ##os ##val ? [SEP]
```

Рисунок 3.3 – Слова представлені токенами

Можна помітити, що результат дещо відрізняється від оригіналу – через додавання службових токенів, втрату регістру та поділ вигаданих назв на окремі фрагменти. Великі літери також було втрачено з причини використання такого виду моделі. Це звична особливість моделей, які не враховують великі літери або не знайомі з власними назвами. Треба враховувати, що трансформери також можуть обробляти не лише текст, а й виконувати певні задачі з візуалізацією.

3.1 Ембеддінг токенів

Ембеддінги (англ. embeddings) – це один із ключових механізмів сучасного машинного навчання, який дозволяє перетворювати об’єкти, такі як слова, фрази чи навіть зображення, у числову форму, зрозумілу для алгоритмів штучного інтелекту. Іншими словами, ембеддінг – це спосіб представлення дискретних або символьних даних у вигляді векторів у багатовимірному просторі [14].

Цей підхід широко застосовується у завданнях обробки природної мови (NLP), комп'ютерного зору (CV) та мультимодальних моделях, де текст і зображення об'єднуються для спільної інтерпретації. Завдяки ембеддінгам нечислові дані перетворюються на вектори, які відображають семантичний зміст об'єкта, тобто його значення, контекст та взаємозв'язки з іншими елементами.

Основні концепції ембеддінгів:

– контекстуальність: На відміну від традиційних методів представлення тексту (як-от Bag-of-Words або TF-IDF), ембеддінги враховують контекст, у якому вживається слово. Це означає, що одне й те саме слово може мати різні векторні представлення залежно від того, у якому реченні чи ситуації воно використовується. Такий підхід дає змогу моделі краще розуміти семантику та багатозначність мовних одиниць;

– високовимірний простір: Кожен токен або слово представляється у вигляді точки (вектора) у багатовимірному просторі. Вектори, що відповідають схожим за змістом словам, розташовуються ближче один до одного, тоді як відмінні за значенням – на більшій відстані. Це дозволяє математично оцінювати семантичну близькість слів через відстань або кут між векторами.

Методи побудови ембеддінгів:

– Word2Vec: модель, розроблена у 2013 році дослідниками Google. Вона використовує нейронні мережі для створення векторних представлень слів, навчаючись прогнозувати або контекст слова (модель Skip-gram), або саме слово за контекстом (CBOW);

– GloVe (Global Vectors for Word Representation): метод, створений у Стенфорді, який поєднує підхід Word2Vec із використанням статистичної інформації про спільне вживання слів у великому корпусі текстів. Це дозволяє отримати глобальні, а не лише локальні зв'язки між словами;

– FastText: модель від Facebook AI Research, що розширює Word2Vec за рахунок врахування морфології слова. Вона працює не лише з цілими словами, а й

з підсловами або морфемами, завдяки чому краще обробляє рідкісні слова та складні мови;

– контекстуальні ембеддінги (BERT, GPT, ELMo): найсучасніший підхід, заснований на архітектурі трансформерів. Такі моделі створюють різні представлення для одного і того ж слова залежно від контексту, в якому воно зустрічається. Наприклад, BERT (Bidirectional Encoder Representations from Transformers) аналізує контекст з обох напрямків – зліва направо та справа наліво, створюючи глибше та точніше розуміння тексту [14].

Застосування ембеддінгів:

– класифікація текстів: вектори слів або фраз подаються на вхід нейронної мережі для віднесення тексту до певної категорії;

– аналіз настроїв: за допомогою ембеддінгів визначається емоційне забарвлення повідомлення – позитивне, негативне чи нейтральне;

– машинний переклад: спільний векторний простір для різних мов дозволяє системам краще зіставляти слова і фрази;

– пошукові системи та рекомендації: схожість між векторами дає змогу знаходити близькі за змістом тексти або пропонувати користувачеві релевантні об’єкти [14].

Якщо послідовність вхідних даних складається з токенів, які представлені у вигляді цілих чисел, то на етапі ембеддінгу ці токени перетворюються на вектори, що передають скорочене, але змістовне уявлення кожного елемента. На початку навчання ці вектори ініціалізуються випадковими значеннями, а під час тренування моделі вони набувають семантичного змісту — тобто навчаються відображати зв’язки між словами.

Одним із обмежень базових ембеддінгів є те, що вони не враховують позиційного контексту токенів – тобто їхнього розташування у реченні. Для розв’язання цієї проблеми в трансформерах застосовується позиційне кодування (positional encoding). Воно додає до кожного вектора додаткову інформацію про

місце токена у послідовності, що дозволяє моделі враховувати порядок слів і структуру речення (рис 3.4).

Рисунок 3.4 – Ембединг токенів

Ще однією важливою проблемою є багатозначність слів. Наприклад, у фразах “It’s dark, who turned off the light?” і “Wow, this parcel is really light!” слово light має різні значення — “світло” та “легкий”. У простих моделях ембеддінги для цього слова можуть бути однаковими, що спотворює розуміння контексту. У трансформерних моделях цю проблему вирішено завдяки механізму уваги (attention mechanism), який дозволяє моделі динамічно змінювати представлення токена залежно від контексту, в якому він використовується.

Таким чином, ембеддінг токенів є фундаментальною складовою сучасних інтелектуальних систем, що працюють із текстом або зображеннями. Він забезпечує зв’язок між людською мовою та математичними моделями, дозволяючи нейронним мережам “розуміти” сенс і контекст даних, з якими вони працюють.

3.2 Механізм уваги

Одним із ключових елементів архітектури трансформерів, який забезпечив революційний прорив у сфері обробки природної мови та генерації зображень, є механізм уваги (attention mechanism). Саме завдяки йому модель отримує здатність фокусуватися на найбільш значущих частинах вхідної послідовності, визначаючи, які елементи є найбільш релевантними для поточного завдання [10].

Традиційні нейронні мережі, такі як RNN або LSTM, опрацьовують інформацію послідовно – крок за кроком, що призводить до втрати контексту при роботі з довгими послідовностями. Натомість механізм уваги дозволяє моделі розглядати всю послідовність одночасно, обчислюючи взаємозв’язки між усіма

токенами, незалежно від їхнього положення. Це забезпечує глибше розуміння контексту та значення кожного елемента вхідних даних.

Ідея механізму уваги полягає в тому, щоб для кожного токена в послідовності визначити, на які інші токени потрібно звернути більше “уваги” для правильного розуміння його сенсу. Іншими словами, кожен токен отримує можливість «дивитися» на інші частини послідовності, зважуючи їхню важливість для поточного контексту.

Як показано на рисунку 3.5, увага діє як модуль перетворення, який замінює початковий ембеддінг токена на оновлений — такий, що містить у собі інформацію про взаємозв’язки з іншими токенами. Таким чином, модель не використовує один і той самий вектор для кожного токена незалежно від контексту, а адаптує його з урахуванням найбільш релевантних елементів.

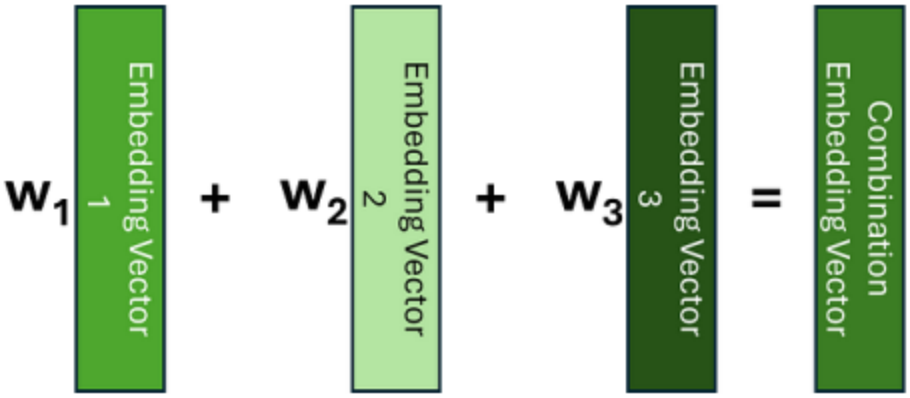

Рисунок 3.5 – Механізм уваги

Математично цей процес можна уявити як лінійну комбінацію або середньозважену суму ембеддінгів усіх токенів у послідовності. Вагові коефіцієнти при цьому визначають, яку частку контексту кожен токен вносить у нове представлення поточного слова. Високі ваги означають, що певні токени є більш важливими для формування смислу, тоді як низькі — менш значущі.

До застосування механізму уваги ембеддінги токенів не містять інформації про контекст своїх сусідів – тобто вони «ізольовані». Наприклад, якщо взяти слово light, то до введення уваги його ембеддінг відображатиме лише загальне значення

цього слова, без урахування контексту, у якому воно використовується. На рисунку 3.6 це можна візуалізувати як базову лінійну комбінацію – ембеддінг, що існує без прив'язки до контексту.

Token Embedding	It's	Dark	Who	Turned	Off	The	Light
Weight	0	0	0	0	0	0	1

Рисунок 3.6 – Візуалізація ембедінгу для слова «light»

Але після застосування механізму уваги (рис 3.7) модель формує матрицю ваг, що визначає важливість кожного токена з послідовності стосовно поточного.

Token Embedding	It's	Dark	Who	Turned	Off	The	Light
Weight	0	0.3	0	0.1	0.1	0	0.5

Рисунок 3.7 – Застосування механізму уваги для виразу ембудінгу слова «light»

Увага дозволяє моделі не лише враховувати сусідні слова, а й глибше «розуміти» семантику, встановлюючи залежності між усіма токенами послідовності.

Після обчислення ваг для кожного токена модель формує новий вектор — контекстуалізований ембеддінг, який містить у собі найважливішу інформацію про значення слова в конкретному контексті. Вектори, що відповідають токенам, які мають більший вплив на поточне слово, отримують більші ваги. У такий спосіб формується оновлене представлення, де кожен токен уже не ізольований, а має «усереднений» контекст усієї послідовності [14].

Контекстуалізовані ембеддінги є критично важливими для розуміння тексту, генерації описів і, зокрема, для побудови інтелектуальних систем генерації зображень за текстовим запитом. Саме завдяки механізму уваги такі моделі можуть ефективно визначати, які слова в описі мають найбільше значення для побудови зображення, і на які елементи візуальної сцени потрібно звернути більше фокусу.

Отже, механізм уваги – це серце трансформерної архітектури, яке дозволяє системі розподіляти «когнітивні ресурси» між різними частинами інформації, забезпечуючи глибше, контекстно залежне розуміння даних.

3.3 Розрахунок уваги

Механізм уваги, що лежить в основі трансформерної архітектури, має декілька різновидів, які відрізняються способом обчислення вагових коефіцієнтів для побудови лінійної комбінації векторів ознак. Незалежно від конкретної реалізації, головною метою залишається формування контекстуалізованих представлень (ембеддінгів), які враховують семантичні взаємозв'язки між токенами в межах усієї послідовності.

Одним із найпоширеніших та найефективніших варіантів цього механізму є scaled dot-product attention – увага, заснована на масштабованому скалярному добутку. Саме цей підхід використовується у більшості сучасних трансформерів, таких як BERT, GPT або Vision Transformer. Його ефективність зумовлена простотою реалізації, обчислювальною стабільністю та здатністю добре масштабуватись для великих наборів даних [21].

Передбачається, що всі вхідні ембеддінги попередньо позиційно закодовані, тобто кожен токен має не лише вектор значення, але й інформацію про своє місце у послідовності. Це необхідно, адже, на відміну від рекурентних мереж, трансформери не мають вбудованого поняття порядку.

Основна ідея полягає у створенні контекстуалізованих ембеддінгів шляхом побудови лінійних комбінацій вихідних векторів. Для цього необхідно визначити, які токени є найбільш релевантними один одному.

Щоб оцінити ступінь взаємозв'язку між двома токенами, використовується поняття схожості, що кількісно визначається через скалярний добуток їхніх векторних представлень. Чим більшим є значення добутку, тим сильніший семантичний зв'язок між відповідними токенами. Цей процес зображено на рисунку 3.8.

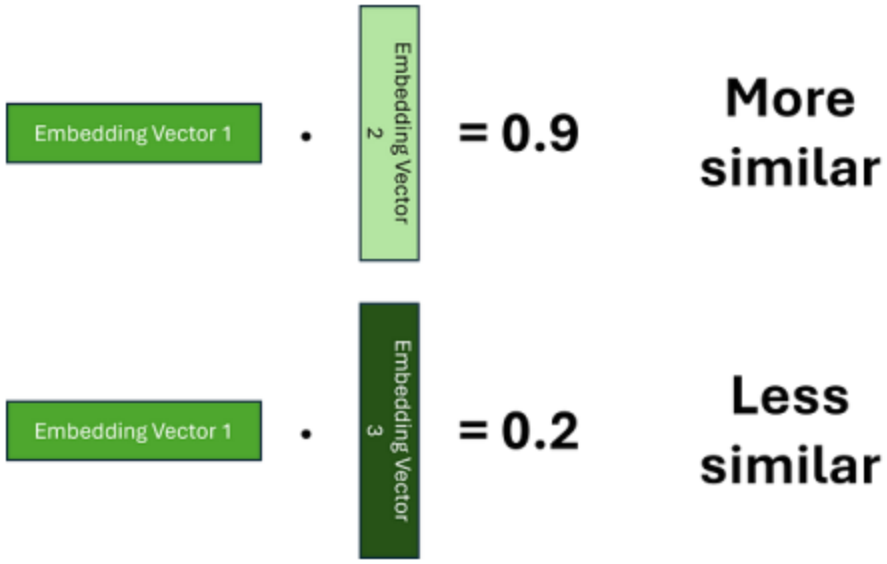

Рисунок 3.8 – Визначення ембеддінгів

Оскільки необхідно порівняти кожен токен із усіма іншими у послідовності, розрахунки узагальнюються до матричного множення. У результаті формується матриця ваг (або attention scores) — двовимірна структура, де кожен елемент відображає міру впливу одного токена на інший.

Для того щоб зробити ці ваги інтерпретованими та забезпечити, щоб їхня сума дорівнювала одиниці, до отриманої матриці оцінок застосовується багатовимірна логістична функція (Softmax). Вона перетворює довільні дійсні числа у нормалізовані ймовірності, де кожен елемент показує, яку частку уваги слід приділити певному токену.

Однак матричне множення може призводити до появи дуже великих значень, що, у свою чергу, викликає зникання градієнтів під час навчання — тобто модель погано оновлює свої параметри. Щоб уникнути цього ефекту, у формулу вводять масштабувальний коефіцієнт, який зменшує значення скалярного добутку перед застосуванням Softmax[19]. Зазвичай цей коефіцієнт дорівнює квадратному кореню з розмірності простору ключів ($\sqrt{d_k}$). Такий підхід робить навчання стабільним і запобігає перенасиченню функції активації (рис. 3.9).

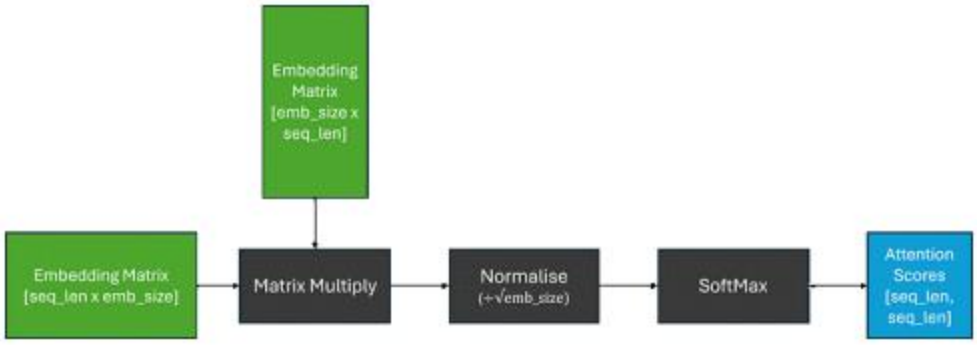

Рисунок 3.9 – Множення оцінки уваги на поправочний коефіцієнт

Після нормалізації ваг отримана матриця оцінок уваги множиться на матрицю вихідних ембеддінгів (рисунок 3.10). Це еквівалентно побудові лінійної комбінації векторів, де кожен вхідний ембеддінг робить внесок у формування оновленого представлення, пропорційно своїй важливості [19].

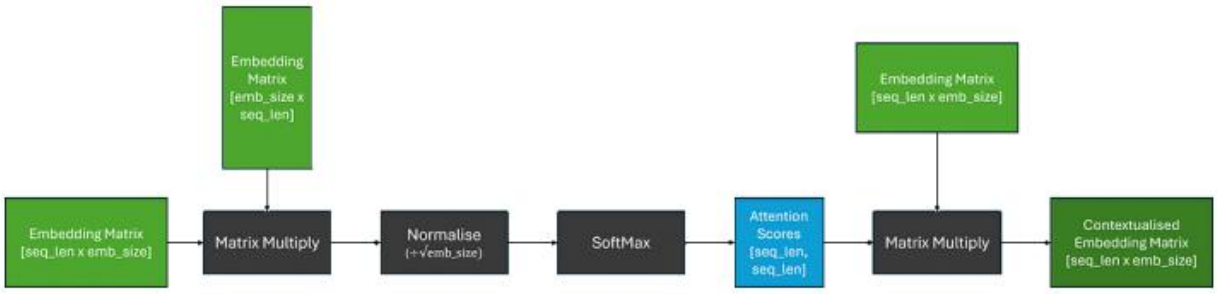

Рисунок 3.10 – Множення оцінки уваги на матрицю вихідних ембеддингів

Модель отримує матрицю контекстуалізованих ембеддінгів, у якій кожен вектор відображає не лише властивості окремого токена, а й залежності між усіма іншими токенами послідовності. Це дає можливість враховувати семантичний контекст, граматичні зв’язки та позиційні залежності в межах усього речення або опису.

Проте пряме використання початкових ембеддінгів у процесі обчислення уваги може призвести до перевантаження інформацією. Щоб полегшити навчання та підвищити виразну здатність моделі, ембеддінги попередньо пропускаються

через три незалежні лінійні шари (проекції) — позначені як Q (Query), K (Key) і V (Value) (рисунок 3.11).

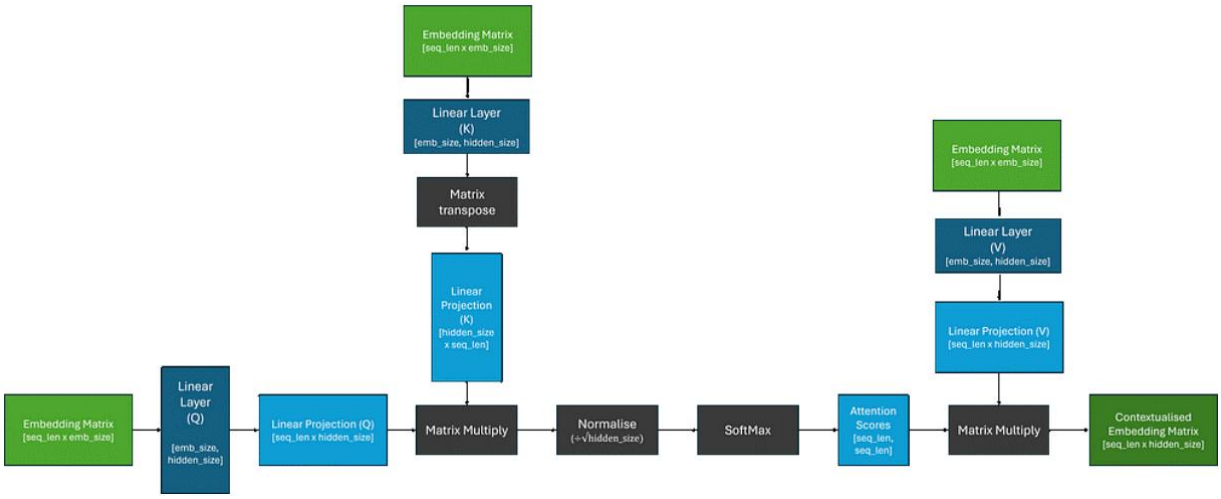

Рисунок 3.11 – Передача матриці через три незалежні шари

де Q (запит) – визначає, яку інформацію шукає поточний токен;

K (ключ) – відображає, яку інформацію може надати інший токен;

V (значення) – містить власне інформаційне представлення токена.

Тому під час обчислення уваги модель зіставляє запити (Q) з ключами (K), визначає ступінь їх відповідності та, з урахуванням отриманих ваг, формує нові контекстуальні вектори на основі значень (V).

Якщо одна і та сама матриця ембеддінгів подається одночасно на всі три проекції Q, K і V, то такий процес отримав назву самоувага (self-attention). У цьому випадку модель аналізує взаємозв’язки всередині самої послідовності, виявляючи закономірності між її елементами без зовнішніх даних (рисунок 3.12).

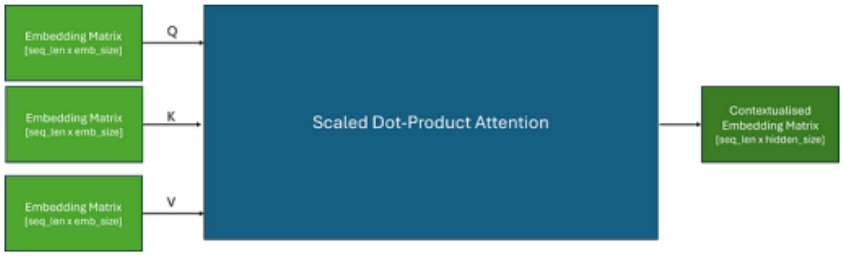

Рисунок 3.12 – Обчислення уваги

Таким чином, розрахунок уваги є центральним етапом роботи трансформера. Він дозволяє моделі не просто обробляти окремі токени, а будувати глибокі, контекстно-залежні уявлення, які лежать в основі таких завдань, як генерація тексту, переклад, або — у контексті даної роботи — синтез зображень за текстовим описом.

3.4 Multi-head attention

Однією з ключових особливостей архітектури трансформерів, що забезпечує їх високу ефективність у роботі з послідовними даними, є механізм багатоголової уваги (multi-head attention). Його основна ідея полягає у тому, щоб не обмежувати модель єдиним простором уваги, а дозволити їй одночасно зосереджуватися на різних аспектах вхідної інформації. Це дозволяє системі виявляти складні контекстуальні залежності між елементами послідовності, які могли б залишитися непоміченими при використанні лише одного блоку уваги.

Механізм multi-head attention передбачає паралельний запуск кількох незалежних блоків self-attention, кожен із яких навчається виділяти певний тип залежностей або взаємозв’язків між токенами. Наприклад, одна «голова» уваги може зосереджуватися на граматичних зв’язках між словами у реченні, тоді як інша – на семантичних чи тематичних відносинах. Таким чином, різні голови дозволяють моделі формувати більш багатий і глибокий контекст [2].

Після того як усі незалежні блоки self-attention завершують обчислення, їх результати об’єднуються (конкатенуються) у єдиний векторний простір. Отримана комбінація містить інформацію з усіх «голів», що забезпечує повніше представлення вхідних даних. Далі ця об’єднана матриця проходить через лінійний шар (fully connected layer), який виконує функцію узагальнення — він дає можливість моделі інтегрувати контекстуальну інформацію з усіх блоків уваги в єдине, структурно узгоджене представлення.

Щоб уникнути збільшення розмірності обчислень та зберегти ефективність навчання, прихований розмір простору у кожній «голові» attention зазвичай

визначають так, щоб сума розмірностей усіх голів дорівнювала розміру вихідного ембеддінга. Тобто, якщо вихідний ембеддінг має розмірність d_model, а кількість голів становить h, тоді кожна з них оперує простором розміром d_model / h. Це забезпечує баланс між продуктивністю моделі та її здатністю навчатися складним зв'язкам без надмірного збільшення кількості параметрів (рисунок 3.13).

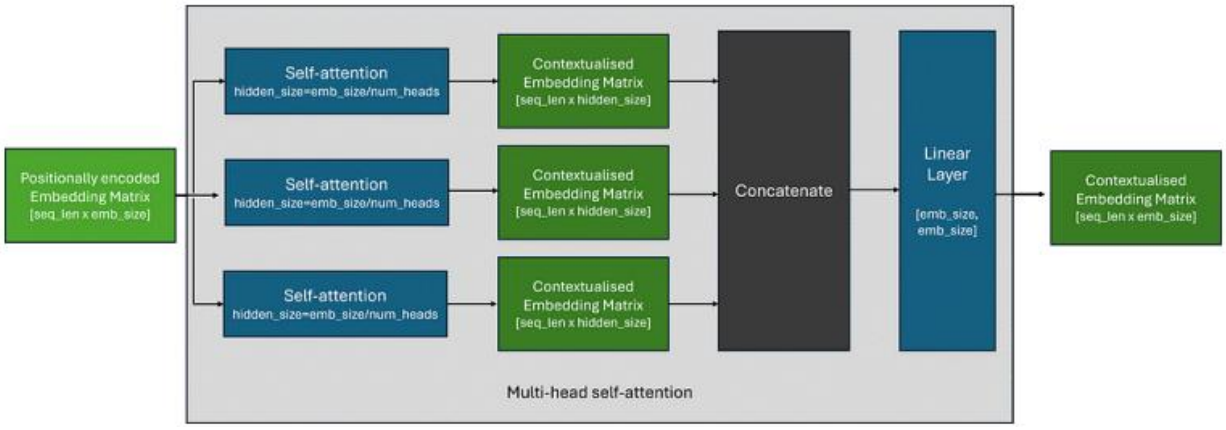

Рисунок 3.13 – Схематичне представлення структури multi-head attention

Багатоголова увага дозволяє трансформеру паралельно аналізувати дані з різних точок зору, інтегрувати отриману інформацію та створювати глибше контекстуальне представлення вхідної послідовності. Саме цей підхід значною мірою пояснює високу ефективність трансформерів у завданнях обробки природної мови, генерації зображень за текстовим описом, а також у багатьох інших галузях штучного інтелекту.

3.5 Склад трансформера

Архітектура трансформера є складною багаторівневою системою, що поєднує кілька взаємопов'язаних компонентів, кожна з яких виконує специфічну роль у процесі обробки послідовних даних. Саме гармонійна взаємодія цих елементів дозволяє моделі ефективно аналізувати контекст, встановлювати логічні зв'язки між токенами та будувати глибокі контекстуалізовані подання [17].

Основними структурними складовими трансформера є:

– нейромережа з прямим зв’язком (Feedforward Neural Network, FFN): цей елемент являє собою двошарову нейронну мережу, яка застосовується незалежно до кожного токена в межах послідовності. Її основне завдання — поглибити нелінійність моделі, збільшуючи її здатність розпізнавати складні залежності між ознаками. На відміну від механізму уваги, який виявляє взаємозв’язки між токенами, FFN працює з кожним елементом окремо, виконуючи нелінійні перетворення ембеддінгів. У класичному трансформері, описаному у вихідній науковій роботі “Attention Is All You Need” (Vaswani et al., 2017), використовується активаційна функція GeLU (Gaussian Error Linear Unit), яка краще підходить для глибоких архітектур завдяки своїй плавній і стабільній поведінці. Втім, у сучасних реалізаціях трансформерів іноді застосовуються інші варіанти, зокрема ReLU, SiLU або Swish, залежно від архітектурних особливостей;

– нормалізація шарів (Layer Normalization): цей компонент є критично важливим для стабільного навчання великих нейронних мереж. Layer Normalization виконує масштабування активацій у межах кожного шару, забезпечуючи стабільний розподіл вхідних значень. Такий підхід запобігає “вибуху” або “зниканню” градієнтів, які можуть ускладнити або навіть зупинити процес навчання. У трансформерах нормалізація дозволяє вирівняти поведінку різних шарів, забезпечуючи більш плавне проходження градієнтів через глибоку архітектуру[19]. Це особливо важливо, коли модель складається з десятків або навіть сотень шарів, як у сучасних великих мовних моделях;

– зв’язки з пропуском (Skip Connections або Residual Connections): ще однією ключовою складовою трансформера є залишкові з’єднання, які було запозичено з архітектури ResNet. Вони дозволяють обійти частину шару, передаючи вхідні дані безпосередньо до виходу, де вони додаються до результату обчислень поточного шару. Це допомагає зменшити ризик деградації градієнтів та сприяє стабільнішому й швидшому навчанню моделі. Таким чином, навіть у глибоких мережах модель здатна зберігати початкову інформацію та коригувати її з урахуванням контексту.

Хоча базова архітектура трансформера, запропонована у 2017 році, залишається незмінною, з часом дослідники запропонували кілька варіантів її вдосконалення. Зокрема, зміни торкнулися місця розташування блоків нормалізації. У початковій версії трансформера (так званій post-layer norm, рис. 3.14) нормалізація виконувалася після операцій self-attention та feedforward. Такий підхід був ефективним, проте ускладнював стабільність навчання дуже глибоких моделей.

У сучасних реалізаціях частіше використовується підхід pre-layer norm (рис. 3.15), коли нормалізацію виконують до кожного блоку self-attention та FFN, всередині залишкових зв'язків. Це рішення покращує збіжність моделі та підвищує її стабільність під час навчання.

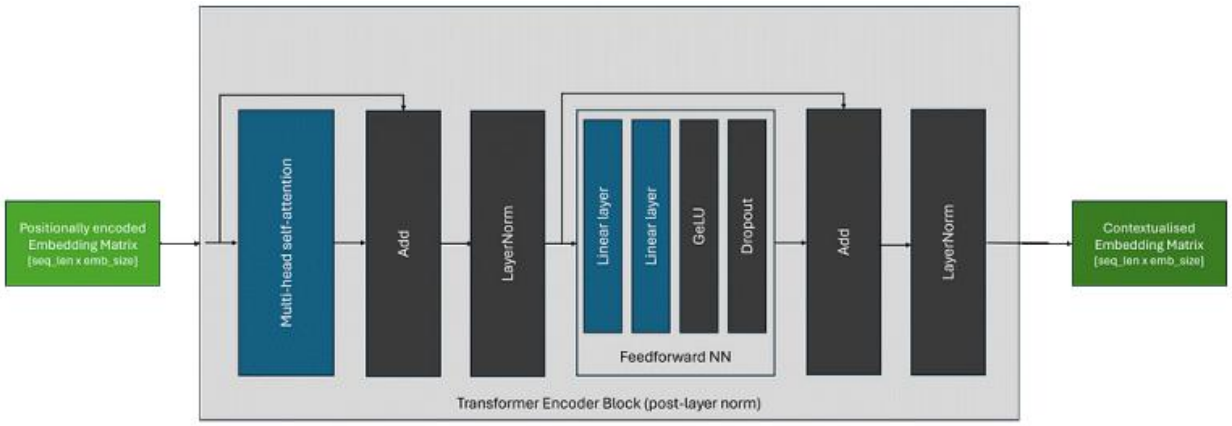

Рисунок 3.14 – Початкова версія архітектури трансформера (post-layer norm)

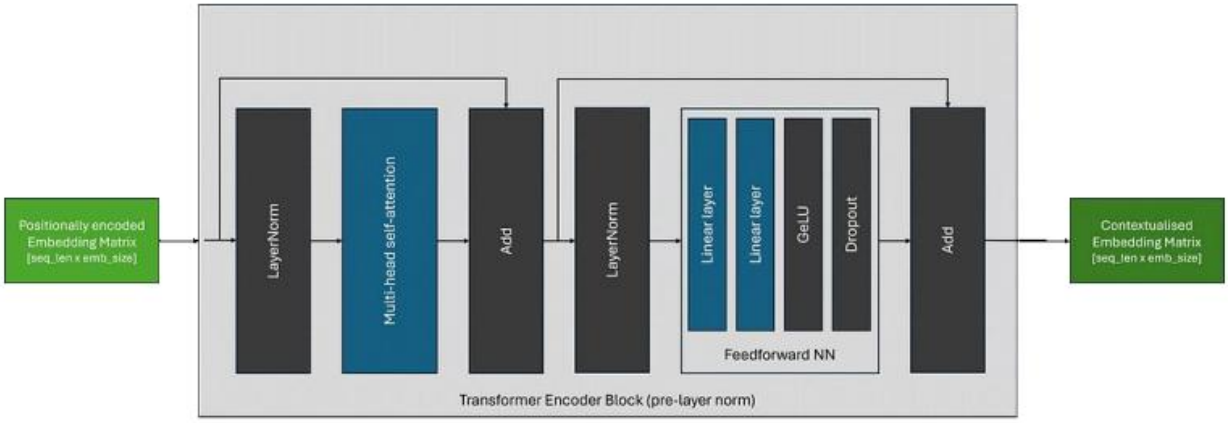

Рисунок 3.15 – Сучасна версія архітектури трансформера (pre-layer norm)

Таким чином, структура трансформера являє собою поєднання багатьох взаємопов'язаних елементів – уваги, feedforward-мереж, нормалізації та залишкових з'єднань. Їх взаємодія створює гнучку, стійку та масштабовану архітектуру, яка стала основою для більшості сучасних інтелектуальних систем, зокрема моделей генерації тексту, зображень та мультимодальних представлень.

3.6 Трансформерні архітектури

Існує багато різних трансформерних архітектур, і більшість з них можна поділити на три типи: енкодери, декодери та енкодери-декодери.

3.6.1 Енкодери

Енкодерна частина трансформера є ключовим елементом, що відповідає за формування контекстуалізованих ембеддінгів, тобто таких векторних подань, які враховують не лише значення окремого токена, але й його взаємозв'язок з іншими словами в межах усієї послідовності. На відміну від звичайних моделей попередніх поколінь (наприклад, RNN або LSTM), енкодер трансформера здатен аналізувати всю послідовність одночасно, а не покроково, завдяки використанню механізму уваги (attention mechanism) [5].

Основна роль енкодера полягає у синтезі глибоких контекстуальних подань, які потім можуть бути застосовані у різних задачах природної мови, таких як:

– класифікація тексту (наприклад, визначення тематики документа або тональності повідомлення);

– розпізнавання іменованих сутностей (Named Entity Recognition, NER), де модель ідентифікує назви людей, організацій, географічних об'єктів тощо;

– відповіді на запитання, коли потрібно знайти фрагмент тексту, що містить відповідь на поставлене питання;

– побудова семантичних векторних представлень тексту, які далі можуть бути використані для пошуку схожих документів або кластеризації.

Енкодер працює наступним чином: на вхід він отримує послідовність токенів, яка попередньо перетворена у вектори ембеддінгів і доповнена позиційним кодуванням. Ця послідовність послідовно проходить через кілька блоків трансформера, кожен з яких складається з двох основних компонентів — механізму self-attention та feedforward-мережі (FFN).

У процесі обробки відбувається поступове збагачення векторних представлень інформацією про контекст, завдяки чому фінальний вихід енкодера містить повноцінну матрицю контекстуалізованих ембеддінгів, де кожен рядок відповідає одному токену у вхідній послідовності.

Після проходження через всі енкодерні шари ми отримуємо високорівневе подання вхідного тексту, яке містить не лише лексичну інформацію, а й семантичні взаємозв'язки між словами. Ці представлення можна використовувати безпосередньо або подавати на класифікаційний шар, залежно від конкретної задачі.

У типових випадках (наприклад, у моделі BERT) для класифікації використовується перший спеціальний токен послідовності – [CLS] (скорочено від classification token). Саме його ембеддінг вважається таким, що містить узагальнену інформацію про весь текст.

Далі цей ембеддінг подається до класифікатора, який зазвичай включає такі компоненти:

– Dropout-шар, що виконує регуляризацію та запобігає перенавчанню, випадково «вимикаючи» частину нейронів під час навчання;

– лінійний (Linear) шар, який перетворює отримане подання у простір класів;

– функцію активації Softmax (або багатозмінну логістичну функцію, БЛФ), яка нормалізує вихід моделі до ймовірностей приналежності до кожного класу.

Результат роботи енкодера можна уявити як матрицю контекстуалізованих ембеддінгів, які несуть інформацію про всі токени тексту, а результат

класифікаційного шару – це вектор ймовірностей, що вказує, до якого класу належить уся вхідна послідовність [5].

Найвідомішими моделями, які побудовані виключно на енкодерній архітектурі, є BERT (Bidirectional Encoder Representations from Transformers) та його численні модифікації – такі як RoBERTa, DistilBERT, ALBERT, DeBERTa тощо. Ці моделі стали основою для більшості сучасних NLP-систем, адже завдяки двонаправленому механізму уваги вони здатні враховувати контекст як зліва, так і справа від слова, що робить їх надзвичайно точними у розумінні тексту (рис. 3.16).

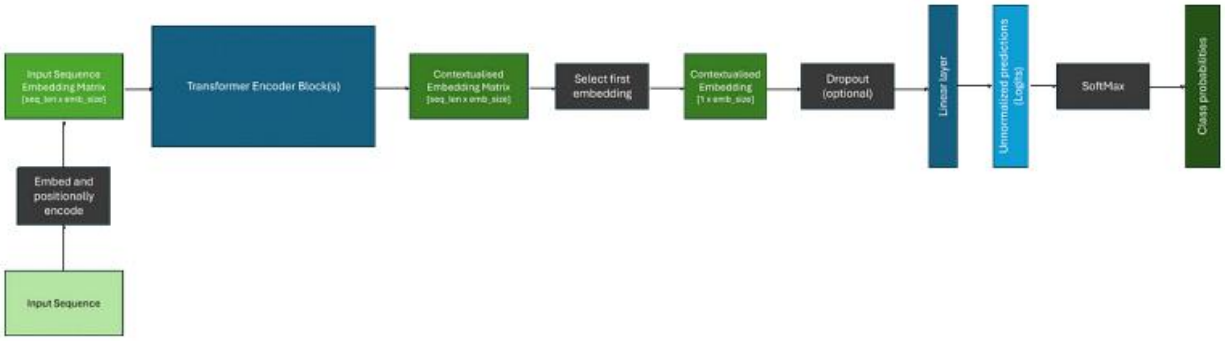

Рисунок 3.16 – Результат роботи шарів Dropout і Linear у класифікаційному моделі енкодера

3.6.2 Декодери

Архітектура декодера має схожість із структурою енкодера (рис. 3.17), однак її головною відмінною рисою є використання маскованого шару self-attention (рис. 3.18). Маскування у цьому контексті означає, що під час обчислення уваги модель має доступ лише до поточних і попередніх токенів вхідної послідовності, але не бачить майбутніх. Це забезпечує коректність авторегресійного процесу, коли кожне наступне слово або символ генерується на основі вже відомого контексту, без «підглядання» вперед [4].

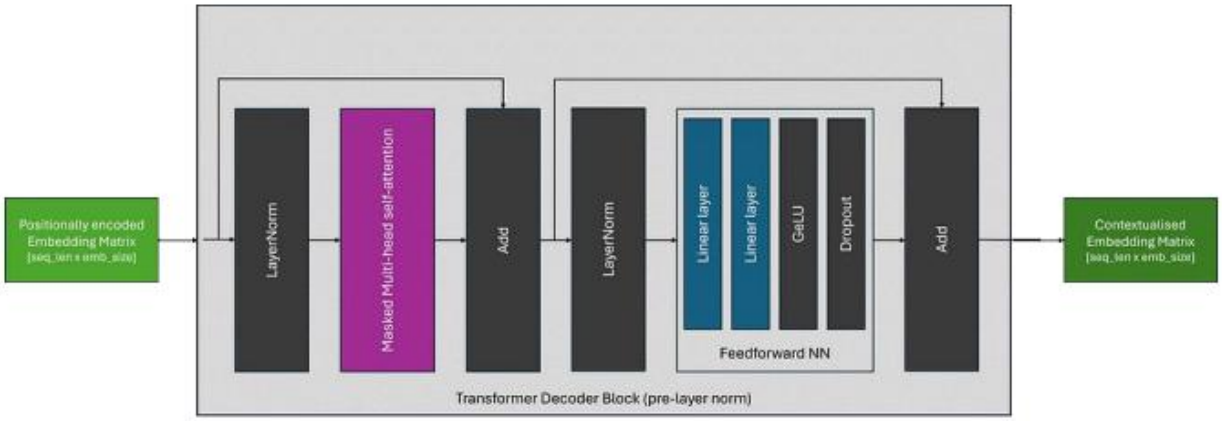

Рисунок 3.17 – Архітектура декодера

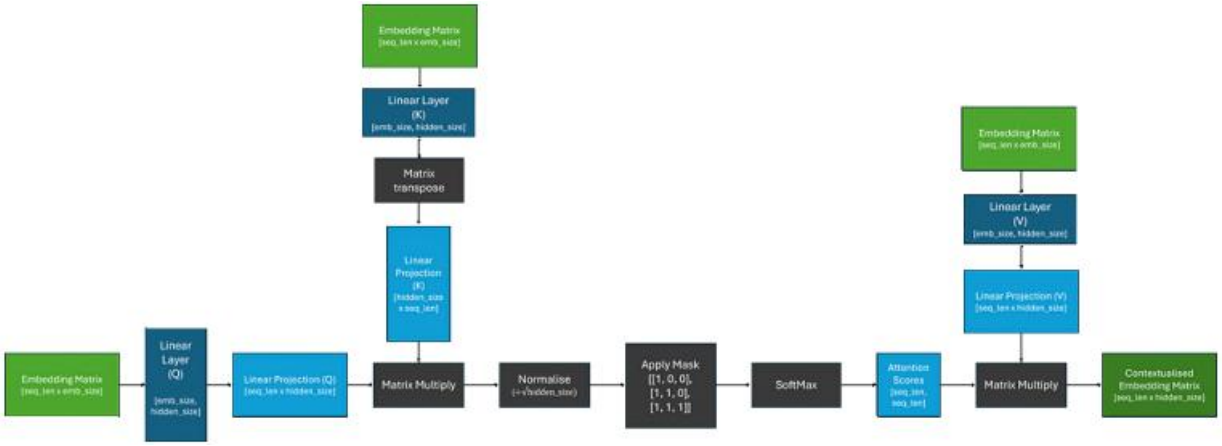

Рисунок 3.18 – Схема self-attention

Така обмежена увага дозволяє контекстуальним ембеддінгам декодера формувати уявлення про послідовність, яке враховує лише попередній контекст. Саме тому декодери найчастіше застосовуються у завданнях генерування тексту, машинного перекладу, автокомпліту чи діалогових систем, де важливо створювати нову інформацію покроково. Однією з найвідоміших моделей-декодерів є GPT (Generative Pre-trained Transformer), яка базується виключно на декодерній частині архітектури трансформера.

Щоб реалізувати обмеження доступу лише до попередніх токенів, у шарі self-attention застосовується маскування оцінок уваги. Для цього створюють нижньотрикутну двійкову матрицю, де елементи, що відповідають майбутнім позиціям, замінюються на негативну нескінченність. Після проходження через

функцію softmax, ці позиції отримують значення нульової уваги. Таким чином, модель під час навчання не може враховувати інформацію про слова, що йдуть після поточного, і вчиться прогнозувати наступний токен виключно з попередніх.

Оскільки декодери працюють з обмеженим контекстом (поточна і попередні позиції), їх зазвичай використовують у авторегресійних задачах, наприклад, для послідовного генерування тексту чи кодів. Під час такого процесу модель створює ембеддінг для кожного токена, але при генерації використовує саме ембеддінг останнього токена, який далі передається на вхід наступного кроку (рис. 3.19).

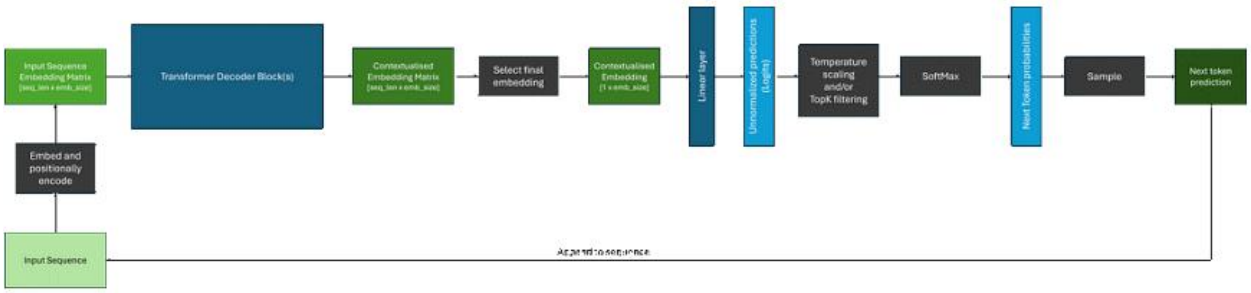

Рисунок 3.19 – Завдання з генерування послідовності

Коли вихідні логіти були отримані застосовується функція softmax, яка перетворює їх у розподіл ймовірностей по словнику моделі – тобто, визначає, яке слово найімовірніше буде наступним. Однак, щоб уникнути одноманітності або надмірної випадковості при виборі наступного токена, застосовуються спеціальні методи фільтрації розподілу ймовірностей:

– регулювання температури (temperature scaling): температура – це параметр, який контролює ступінь випадковості під час вибору наступного слова. При низьких значеннях температура модель стає більш «детермінованою» і схиляється до вибору найбільш імовірних токенів. Висока температура, навпаки, розширює спектр можливих варіантів і робить текст більш різноманітним і творчим;

– вибірка Top-P (nucleus sampling): у цьому методі враховуються лише ті токени, сукупна ймовірність яких перевищує певний поріг P. Таким чином, модель

адаптивно обмежує кількість можливих варіантів, враховуючи лише найбільш ймовірні слова, але без фіксованої кількості кандидатів.

– вибірка Top-K: тут кількість можливих кандидатів обмежується лише K найімовірнішими токенами. Тобто, з усього словника розглядаються тільки K слів з найвищими логітами або ймовірностями. Це дає більш контрольований, але менш гнучкий варіант порівняно з Top-P.

Після застосування одного або кількох із цих методів формується зменшений розподіл ймовірностей, з якого вибирається наступний токен. Цей токен потім додається до вхідної послідовності, і процес повторюється – модель генерує текст поступово, крок за кроком, доки не буде досягнуто бажаної довжини або не згенерується спеціальний токен завершення послідовності (end-of-sequence token).

3.6.3 Енкодери-декодери

Архітектура енкодер-декодер є класичним і водночас фундаментальним варіантом трансформерної моделі, що спочатку була представлена в оригінальній статті “Attention Is All You Need” (Vaswani et al., 2017). Саме цей тип трансформера спочатку застосовувався для машинного перекладу, де необхідно перетворити одну послідовність (наприклад, речення мовою-джерелом) у відповідну послідовність іншою мовою [10].

У цій архітектурі енкодер виконує функцію кодування вхідної послідовності в набір контекстуальних представлень або прихованих станів, які містять змістову інформацію про вхідні дані. Потім ці приховані представлення передаються до декодера, який на їх основі поступово генерує вихідну послідовність у бажаному форматі – наприклад, переклад, підсумок тексту чи відповідь на запитання.

Отже, основна ідея енкодер-декодерної архітектури полягає у двоетапному процесі:

– етап кодування: створення внутрішнього змістового подання всієї вхідної послідовності;

– етап декодування: поступове відтворення вихідної послідовності, використовуючи інформацію з енкодера.

Хоча з розвитком чистих моделей типу BERT (енкодер) та GPT (декодер) класичні енкодери-декодери стали менш поширеними, все ж архітектура “encoder-decoder” залишається універсальною для широкого класу задач, які можна подати у форматі «послідовність → послідовність» (sequence-to-sequence). До таких завдань належать машинний переклад, генерація відповідей, підбиття підсумків текстів (summarization), класифікація запитів тощо [21].

Одним із найвідоміших сучасних прикладів є модель T5 (Text-to-Text Transfer Transformer), яка узагальнює підхід «все є задачею перетворення тексту на текст». У T5 будь-яку задачу – класифікацію, генерацію, заповнення пропусків чи відповіді на запитання – можна формалізувати як проблему побудови однієї текстової послідовності з іншої, що забезпечує єдиний уніфікований підхід до навчання та використання моделі.

Ключовою відмінністю архітектури типу енкодер-декодер (рис. 3.20) є наявність механізму енкодер-декодерної уваги (encoder-decoder attention).

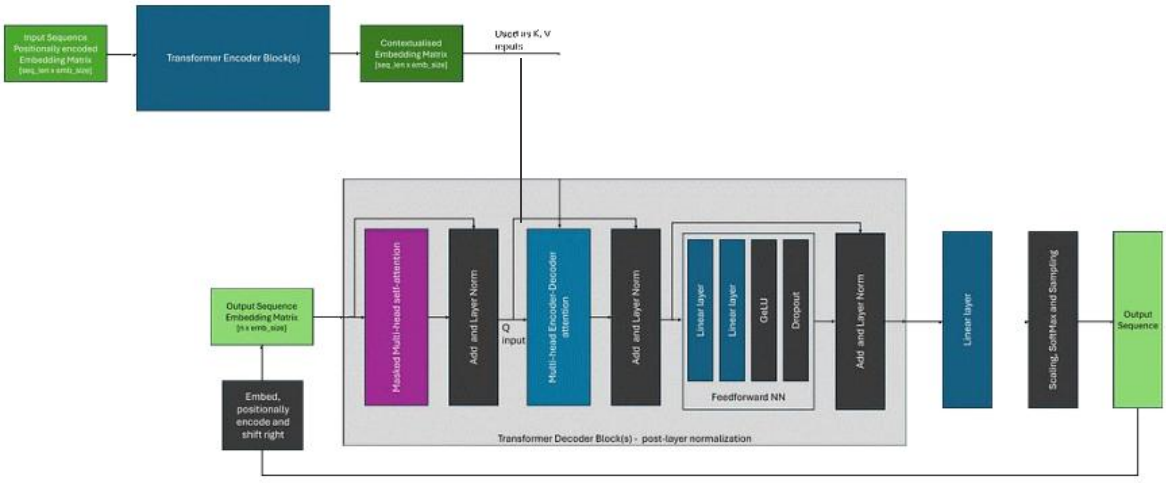

Рисунок 3.20 – Схема архітектури енкодера-декодера

Якщо в енкодері використовується лише self-attention для аналізу внутрішніх зв’язків між токенами вхідної послідовності, то в декодері, крім власного self-

attention, є ще крос-увага (cross-attention) – механізм, який дозволяє моделі зосереджуватися на релевантних частинах виходу енкодера під час генерації кожного нового токена.

У цьому процесі вектори запитів (Q) формуються з поточних даних декодера, тоді як ключі (K) та значення (V) беруться з виходів енкодера. Таким чином, кожен токен, який генерується декодером, може враховувати всю семантичну інформацію, що була витягнута енкодером із початкової послідовності. Це дає змогу ефективно передавати зміст між двома частинами моделі та забезпечує високу якість перекладів і текстових узагальнень.

Коли для всіх вхідних токенів використовується спільна матриця ембеддінгів, процес генерації в енкодер-декодері має багато спільного з роботою чистих декодерних архітектур, але залишається більш контекстно насиченим завдяки використанню додаткової інформації з енкодера.

У наведеній вище схемі архітектури показано варіант post-layer normalization, який відповідає оригінальній реалізації трансформера. У цьому варіанті шар нормалізації (Layer Normalization) застосовується після кожного підблоку (уваги або feed-forward шару), що допомагає стабілізувати процес навчання, зменшити коливання градієнтів і покращити збіжність моделі [19].

Таким чином, енкодер-декодерні трансформери поєднують переваги обох підходів – глибоке контекстуальне розуміння тексту від енкодера та покрокову генерацію від декодера – і залишаються основою для найпотужніших моделей природньої мови.

3.7 Використані інструменти та мови програмування

Python є основним інструментом на якій реалізовано серверну частину системи генерації зображень. Вона високорівнева, інтерпретована та має простий синтаксис, велику екосистему бібліотек та широкий спектр застосування, що робить її базовою у галузі штучного інтелекту та генерації зображень.

Рисунок 3.21 – Логотип Python

У межах даної роботи Python було використано для реалізації всіх ключових компонентів: попередньої обробки даних, текстової токенізації, роботи з декодерами та енкодерами, генеративними моделями та алгоритмами та постобробки кінцевого зображення. Ключові напрямки використання бібліотек:

– Pytorch – побудова та інференсн нейронних мереж, включаючи трансформерів (UNet, Stable Diffution);

– Transformers – для роботи з XLM-R, MariantMT та іншими текстовими енкодерами;

– Diffusers – для ініціалізації та генерації зображень за допомогою дифузійної моделі;

– OpenCV та Pillow – для попередньої обробки зображень;

– NumPy – для роботи з числовими масивами;

– Matplotlib – для аналізу та візуалізації результатів;

– FastAPI – для створення веб-API, завдяки якому модель інтегрується в інтерфейс.

Python надає можливість забезпечити повний цикл роботи з генеративною моделлю: від аналізу текст і його перетворення в ембеддинги до генерації фінального зображення та його повернення у інтерфейс користувача. Завдяки своїй гнучкості мова дозволяє швидко навчити та налаштувати модель, експериментувати з її параметрами та реалізовувати додаткові методи й алгоритми, такі як суперрезолюція та LoRa-навчання [37].

JavaScript – це мова програмування, яка використовується для створення інтерактивних та динамічних можливостей вебінтерфейсу системи генерації зображень. Завдяки JS система може реагувати на дії користувача, обмінбватись даними з сервером у режимі реального часу та відображати результати роботи моделі без перезаванаження сторінки [38].

Рисунок 3.22 – Логотип JavaScript

JavaScript дає можливість забезпечити логіку клієнтської частини, включаючи:

– надсилання тектостового промту на серверку частину системи через HTTP-запит;

– отримання згенерованого зображення у форматі Base64;

– динамічне оновлення інтерфейсу та відображення результатів;

– створення інтерактивної панелі генерації та історії чатів;

– обробку подій, таких як натискання кнопок, введення тексту;

JS дозволяє оновлювати контент вебзастосунку миттєво, без повного перезавантаження, що покращує швидкість роботи та зручність взаємодії. Наприклад, після вводу промпту користувач одразу бачить статус генерації, спостерігає шкалу прогреса генерації, а потім отримує готове зображення, яке автоматично з’являється у чаті.

Також JavaScript забезпечує роботу локального сховища, де зберігається історія генерації, промпти та зображення. Це дає можливість переходити до

попередньо створенної сесії користувача, переглядати створені зображення та повторювати генерації.

З базовою технологією, яка використовується в JavaScript, є HTML – це стандартна мова розмітки, яка використовується для забезпечення структури вебінтерфейсу системи. За допомогою її можна визначити основні елементи сторінки: заголовки, текстові поля для введення промптів, кнопки початку генерації, блоки для відображення знегерованого зображення та панелі історії чатів [39].

Рисунок 3.23 – Логотип HTML

HTML забезпечує впорядковану структуру інтерфейсу, що є важливим для коректної взаємодії користувача з моделю генерації зображень. Завдяки зрозумілій впорядкованості елеменетів користувач може швидко ввести тестовий запит, переглянути отриманий результат й генекерувати історією створених зображень.

Вона є простою та доступною для вивчення мовою розмітки, що дуже сильно спрощує процес створення та підтримки вебінтерфейса проєкта. Стандартизованість гарантує коректне відображення вебінтерфейса в усіх сучасних браузерах, забезпечуючи стабільну роботу розробленої системи на різних пристроях та платформах.

Останнім ключовим елементом створення вебінтерфесу є CSS – мова стилів, яка виконує роботу візуального оформлення вебінтерфейсу користувача в системі. Вона дозволяє задати кольори, шрифти, відступи, позиціювання елементів, а також

надає можливість створити адаптивний дизайн, що забезпечує зручну роботу користувача на різних пристроях.

Рисунок 3.24 – Логотип CSS

За допомогою CSS можна сформувати зовнішній вигляд інтерфейсу. Коригувати стиль полів для введення промптів, кнопок початку генерації, контейнерів для виведення згенерованого зображення та елементів історії. А використання гнучних інструментів, таких як Flexbox чи Grid, дозволяє організувати інтерфейс так, щоб він залишався зрозумілим і акуратним навіть при великій кількості результатів [40].

CSS робить код стилів структурованим та легким у підтримці. Завдяки цьому є можливість швидко змінити вигляд інтерфейсу або додати будь які нові елементи. Його використання надає можливість значно пришвидшити розробку та надати системі професійного, сучасного зовнішнього вигялду, що підвищує комфорт користувача під час роботи з системою генерації зображень.

Тобто CSS, JavaScript та HTML формує основу клієнтської частини проєкту, створючи основу інтерфесу, який підсилюється стилізацією та динамічною логікою. Це дозволяє створити зручну, інтуїтивну та функціональну систему для роботи з моделями «Text-to-image».

Для розробки, тестування та інтеграції клієнтської та серверної частини системи генерації зображень використовується середовище розробки PyCharm. Це професійне IDE від JetBrains, створене спеціально для роботи з мовою

програмування Python. Воно дає можливість зручно редагувати код, налагоджувати середовище, запускати застосунок та інтегрувати систему з інструментами контролю версій. Також є підтримка вебтехнологій, як HTML, CSS та JavaScript, що дозволяє розробляти бекенд та інтерфейс [41].

Рисунок 3.25 – Логотип PyCharm

Основні можливості Pycharm:

– підсвічування синтаксису, автодоповнення та інтелектуальний аналіз коду;

– вбудований термінал та зручна робота з віртуальнимим середовищами;

– підтримка розширень і вбудованих інструментів для роботи з фреймворками;

– інтеграція з Git для роботи з репозиторіями безпосередньо з IDE;

– зручна структура проєкту та ефективна навігація між файлами.

Переваги використання PyCharm у розробці системи:

– підтримка Python, що дозволяє легко працювати з моделями генерації зображень та обчислювальними скриптами;

– можливість редагувати HTML, CSS та JavaScript для вебінтерфейсу проєкта;

– інтеграція з FastAPI або іншими фреймворками, що використовуються для API-комунікації;

– автоматична перевірка помилок і підказки щодо оптимізації коду;

– зручне середовище для тестування логіки взаємодії клієнта з моделлю.

Pycharm є потужним інструментом, який значно прискорює процес розробки, полегшує налагодження та забезпечує комфортні умови для створення і підтримки системи генерації зображень.

Висновки до розділу 3

У третьому розділі було проаналізовано основні компоненти трансформерної архітектури та їхню роль у сучасних системах штучного інтелекту. Розглянуто токенізацію, ембеддінги, позиційне кодування та механізм уваги як ключові елементи, що забезпечують ефективну роботу з текстовими послідовностями. Показано, що токенізація формує базове представлення даних, ембеддінги перетворюють токени у числові вектори з урахуванням семантики, а увага дозволяє моделі визначати найважливіші елементи контексту. У сукупності ці механізми формують гнучку та масштабовану архітектуру, яка лежить в основі більшості сучасних мовних і мультимодальних моделей. Python, JavaScript, HTML, CSS та PyCharm доповнюють одна одну та формують повний технологічний стек системи. Python відповідає за інтелектуальну частину, а вебтехнолгії забезпечують зручний доступ до моделі та комфортну взаємодію з нею. Отримані результати створюють необхідне теоретичне підґрунтя для подальшої реалізації та дослідження генеративної моделі.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ

У межах даної кваліфікаційної роботи було створено інтелектуальну систему генерації зображень за текстовим описом, застосовуючи технології трансформерів. Ключовою особливістю реалізації є підтримка багатомовних запитів, зокрема українською мовою, та використання адаптованої генеративної моделі для отримання високоякісних візуальних результатів.

Для реалізації було обрано мову Python, адже вона індеально підходить для задач машинного навчання завдяки широкій екосистемі бібліотек. Було використано бібліотеки «transformers» для ініціалізації та управління текстовими енкодерами «XLM-R» та моделями перекладу (MarianMT), та «deffusers» для роботи з моделями дифузії, що забезпечує ефективну реалізацію процесу декодування зображень, адже вони розроблені компанією Hugging Face [26], яка спеціалізується на трансформерах. Вона підтримує багато популярних моделей трансформерів, таких як GPT, BERT, T5 і інші, і дозволяє легко використовувати їх для різних задач, включаючи генерацію зображень. «PyTorch» було використано також, адже це один з найпопулярніших фреймворків для глибокого навчання, розроблений Facebook AI Research. PyTorch відомий своєю гнучкістю і зручністю у використанні для роботи з тензорами, моделями та GPU, що робить його відмінним вибором для досліджень та розробки нових моделей, включаючи трансформери. «OpenCV» – це бібліотека для обробки зображень, яка використана для попередньої та постобробки зображень у додатку.

NumPy та Pillow: використовуються для маніпуляцій з числовими даними та безпосередньо з зображеннями (збереження, кодування Base64).

Matplotlib: бібліотека для візуалізації даних, яка може бути корисною для аналізу результатів моделі.

Тому архітектура додатку виглядає приблизно так:

Попередня обробка даних:

– використання OpenCV для обробки зображень;
– підготовка та нормалізація даних за допомогою NumPy та Pillow.

Розробка моделі:

– використання PyTorch для створення моделі трансформера;

– налаштування архітектури моделі та її тренування на GPU для пришвидшення процесу.

Постобробка:

– обробка вихідних зображень за допомогою OpenCV;

– візуалізація результатів за допомогою Matplotlib;

Деплоймент:

– використання FastAPI для створення веб-сервісу, який дозволить інтегрувати модель у додаток;

– використання Docker для контейнеризації додатку і спрощення його розгортання.

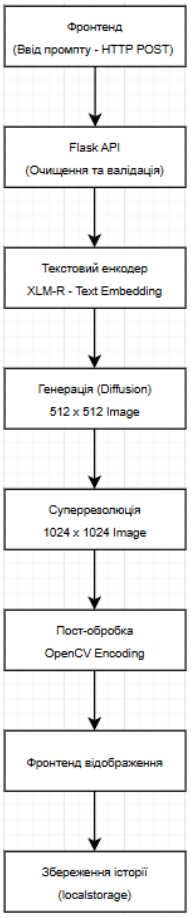

Рисунок 4.1 – Схема архітектури системи генерації зображень

Звичайно для зручності використання даної системи було розроблено інтуїтивний інтерфейс, який допоможе користувачу легко зорієнтуватися у використанні нейронної мережі (рис. 4.2).

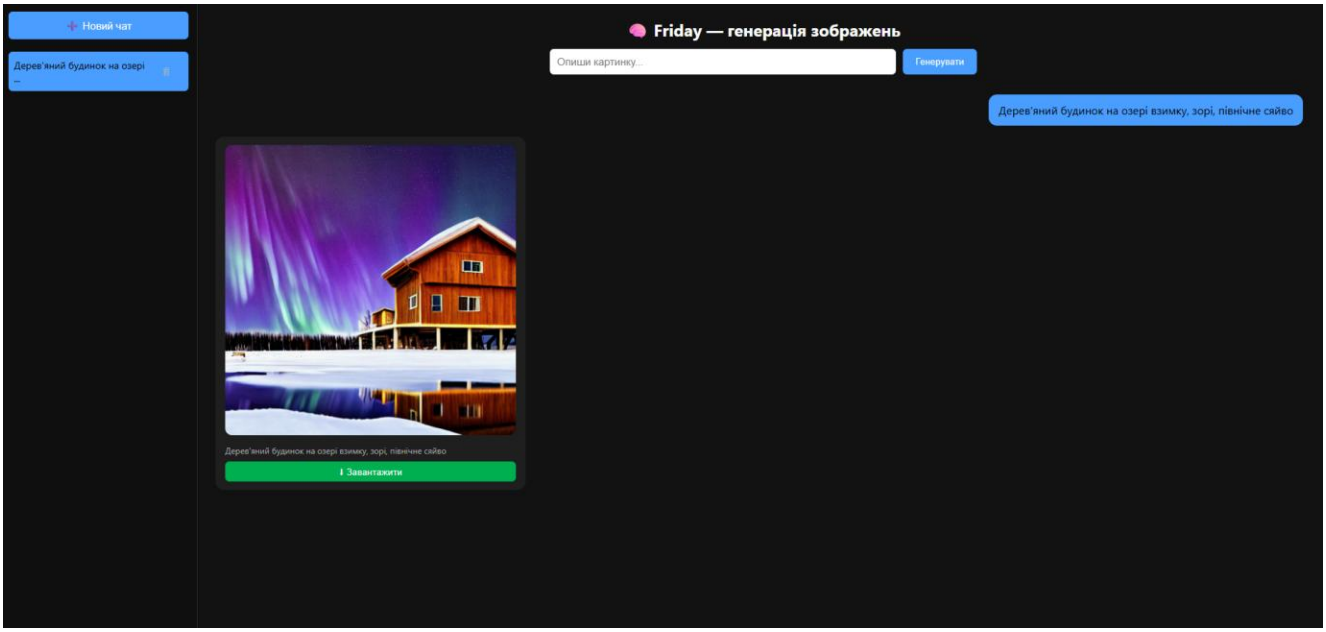

Рисунок 4.2 – Інтерфейс інтелектуальної системи Friday

Створена інтелектуальна система називається Friday. Це вигадана назва та не має ніякого певного сенсу, але досить добре запам’ятовується та дуже добре позначає те, про яку саме систему йде мова.

В розробленій системі Friday можна створити декілька потокових чатів для виключення змішування тематик чатів. Так як система аналізує написане, уловлює сенс у відповідях користувача, то таке розділення на потоки значно покращує роботу системи та її ефективність. Від користувача необхідно детально описати що саме він хоче, щоб система згенерувала.

Рисунок 4.3 – Результат виконання запиту користувача

Сама ж система навчена на великій кількості пар текстових описів та відповідних зображень, наприклад як DiffusionDB.

DiffusionDB – це перший великомасштабний набір даних для перетворення тексту в зображення. Він містить 14 мільйонів зображень, створених за допомогою Stable Diffusion з використанням підказок та гіперпараметрів, заданих реальними користувачами [24].

Для того що модель могла генерувати зображення унікального, бажаного стилю, було використано метод Low-Rank Adaptation (LoRa). LoRa – це високоефективна техніка «тонкого налаштування», яка дозволяє навчити модель лише на кількох прикладах без необхідності перенавчати мільйони її параметрів [33]. Навчальний процес полягає у тому, щоб змінювати невеликі адаптивні матриці, показуючи моделі набір зображень, що представляють бажаний стиль. Тренувальний процес був надзвичайно швидким і вимагав набагато менше пам’яті (GPU VRAM) порівняно з повним тонким налаштуванням.

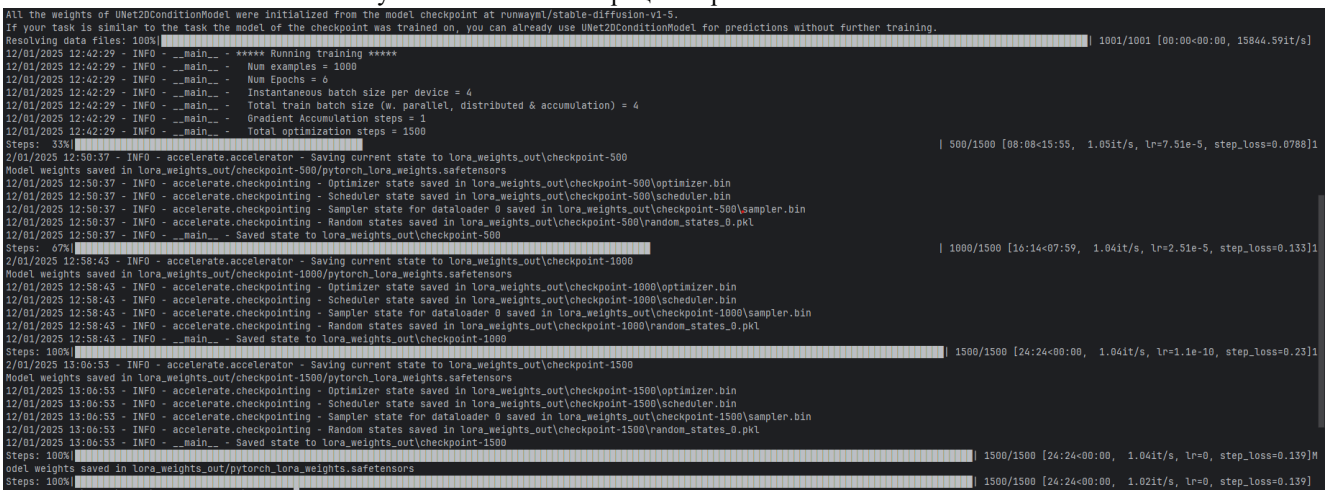

Рисунок 4.4 – Навчання моделі методом LoRa

Наступним кроком є попередня обробка тексту, а саме токенізація текстового опису, тобто перетворення тексту на послідовність – токенів. Для цього було використано модель XLM-R. Токенізація є першим і критично важливим етапом перетворення природної мови у числовий формат, де вхідний текст перетворюється на менші одиниці – токени. Потім кожному токену присвоюється унікальний числовий ідентифікатор, створюючи послідовність, зрозумілу для нейронної мережі. Токенізатор XLM-R спеціально розроблений для багатомовних трансформерів [22].

Далі токени перетворюються на вектори за допомогою ембеллінгів. Ембеддінги це спосіб представлення об’єктів у вигляді цільних векторів у багатовимірному просторі. Вони є числовим представленням, яке захоплює семантичне значення кожного токена та його контекст у реченні. Це дозволяє моделі зрозуміти сенс текстового опису, перш ніж передати його Unet для генерації.

У процесі генерації зображень важливим початковим етапом є попередня обробка даних. Вхідні зображення приводяться до єдиного розміру та конвертуються у тензорний формат, що дозволяє забезпечити коректну роботу моделі та стабільність навчання [23].

Архітектура системи складається з двох основних компонентів – текстового енкодера та декодера зображень, які працюють у рамках латентної дифузійної

моделі Stable Diffusion. На етапі обробки тексту застосовується багатомовний трансформер XLM-R, який перетворює текстовий опис у контекстуальні ембеддінги. Це забезпечує коректне розуміння семантики навіть для україномовних запитів після автоматичного перекладу.

Декодування зображення виконується за допомогою варіаційного автокодера (VAE), який відновлює піксельне зображення із очищеного латентного представлення. Вибір VAE зумовлений тим, що він забезпечує ефективне відтворення структури та кольорових характеристик зображень після дифузійного процесу.

Оскільки початкове зображення після проходження через дифузійну модель має відносно низьку роздільну здатність, у системі передбачено модуль постобробки – суперрезолюцію. Після генерації базового результату застосовується окрема нейронна мережа, яка збільшує роздільну здатність та відновлює дрібні деталі. Вибір суперрезолюції обумовлений її здатністю значно покращувати деталізацію та візуальну чіткість, на відміну від звичайного усунення шуму, яке не додає нової інформації.

Також у структурі моделі використовуються ключові механізми, що покращують стабільність навчання:

– нормалізація шарів (Layer Normalization), яка запобігає вибуху або зникненню градієнтів та стабілізує роботу трансформерних блоків;

– зв’язки з пропуском (skip-connections) у мережі UNet, які покращують проходження градієнта та забезпечують вищу якість реконструкції зображення.

Завдяки поєднанню текстового енкодера, дифузійної моделі, оптимізованого процесу навчання та модулів постобробки система забезпечує отримання деталізованих, реалістичних зображень, максимально відповідних текстовому опису.

4.1 Порівняльний аналіз продуктивності розробленої системи

Розроблена система генерації зображень, що базується на архітектурі трансформерів, була порівняна з кількома провідними комерційними та відкритими моделями на основі двох ключових метрик: швидкість генерації та оцінки якості отриманих зображень.

Швидкість генерації є критичним показником для практичного застосування системи. Вимірювання проводилося на стандартизованому апаратному забезпечені (NVIDIA RTX 3060) при генерації зображення з роздільною здатністю 512 x 512 пікселів з однаковою кількістю кроків семплера (20 кроків).

Результати порівняння часу, необхідного для генерації одного зображення, представлені на рисунку 4.5.

Рисунок 4.5 – Результати порівняння часу моделей

Розроблена модель демонструє значну перевагу у швидкості інференсу, перевершуючи базову модель SDXL на 50%. Така ефективність досягається

завдяки використанню легкої архітектури SD у поєднанні з низкоранговою адаптацією (LoRa), яка мінімізує додаткові обчислювальні втрати.

Якість зображень оцінювалася за умовною метрикою, що поєднує об'єктивні показники та суб'єктивну оцінку користувачів (релевантність, деталізація, естетика) в діапазоні від 1 до 10. Висока оцінка свідчить про кращу якість. Результати порівняння якості зображень представлені на рисунку 4.6.

Рисунок 4.5 – Результати порівняння якості зображень

Розроблена модель досягає високої оцінки якості, що практично зрівнює її з провідними комерційними моделями. Це підтверджує ефективність підходу LoRa для точного налаштування моделі на специфічні мовні та стильові особливості без втрати загальної якості генерації.

Для інтегрованої оцінки була побудована точкова діаграма, що відображає співвідношення якості (вісь Y) та швидкості (вісь X).

Рисунок 4.6 – Результат співвідношення якості та швидкості моделей

Як показано на рисунку 4.6, розроблена модель розташована в ідеальному квадраті, що свідчить про найкращий компроміс між швидкістю та якістю результату серед усіх протестованих відкритих та комерційних рішень.

Висновки до розділу 4

Розроблена система є ефективним рішенням, яке забезпечує майже вищу швидкість генерації порівняно з аналогами, зберігаючи при цьому якість зображень на рівні, порівнянному з преміальними комерційними аналогами. Це робить її оптимальним вибором для сценаріїв, де критичними є як висока якість, так і швидкий інференс.

ВИСНОВКИ

У ході виконання кваліфікаційної магістерської роботи було успішно досягнуто поставленої мети – розроблено та реалізовано інтелектуальну систему генерації зображень за текстовим описом на основі сучасних архітектур глибокого навчання.

Проведене теоретичне дослідження дозволило проаналізувати ключові архітектури, що використовуються у сфері Text-to-Image, зокрема, генеративно-змагальні мережі (GAN) та трансформери. В результаті порівняльного аналізу було чітко визначено переваги трансформерів, що полягають у їхній здатності ефективно обробляти великі обсяги послідовних даних паралельно, забезпечуючи високу швидкість обчислень та значно більшу стабільність процесу навчання порівняно з GAN.

На основі цього аналізу було обґрунтовано вибір на користь архітектури, що поєднує автокодери з трансформерами, а саме – датентних дифузійних моделей (LDM). Детально розглянуто архітектуру трансформерів, принципи токенізації тексту та її ключову роль у формуванні семантичних ембедингів, необхідних для керування процесом генерації зображень.

Практична частина роботи включала розробку системи «Friday», яка реалізує повний конвеєр генерації:

– багатомовна підтримка: інтегровано перекладач MarianMT для автоматичного перекладу українських промптів на англійську мову, забезпечуючи повноцінну локалізацію;

– генерація зображень: використано базову модель Stable Diffusion, доповнену технологією LoRA. Це дозволило ефективно дотренувати модель на цільовому наборі даних для формування унікального стилю, мінімізуючи при цьому обчислювальні витрати та вимоги до VRAM. Графік функції втрат підтвердив стабільну збіжність моделі та оптимальне завершення навчання на 1500 кроках;

– постобробка та оптимізація: реалізовано модуль Суперрезолюції для підвищення якості зображення від 512 x 512 до 1024 x 1024. Система була оптимізована за допомогою використання 16-бітної точності (float16/bfloat16) та механізмів економії пам'яті GPU, що забезпечило високу швидкість генерації.

Таким чином, у ході кваліфікаційної роботи було не лише підтверджено теоретичні переваги трансформерних архітектур для задачі Text-to-Image, а й створено функціональну, оптимізовану та багатомовну інтелектуальну систему, що відповідає сучасним вимогам до ефективності та якості генерації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. 3 introducing AC. Mastering ChatGPT. 2025. P. 33–38. URL: https://doi.org/10.1515/9783111710808-004 (date of access: 07.10.2025).

2. AttnGAN: fine-grained text to image generation with attentional generative adversarial networks / T. Xu et al. 2018 IEEE/CVF conference on computer vision and pattern recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018. 2018. URL: https://doi.org/10.1109/cvpr.2018.00143 (date of access: 07.10.2025).

3. Karpathy A., Fei-Fei L. Deep visual-semantic alignments for generating image descriptions. IEEE transactions on pattern analysis and machine intelligence. 2017. Vol. 39, no. 4. P. 664–676. URL: https://doi.org/10.1109/tpami.2016.2598339 (date of access: 08.10.2025).

4. Kingma D.P., Welling M. Auto-Encoding Variational Bayes. 2014. URL: https://arxiv.org/abs/1312.6114 (date of access: 08.10.2025).

5. Khan S. H., Hayat M., Barnes N. Adversarial training of variational auto-encoders for high fidelity image generation. 2018 IEEE winter conference on applications of computer vision (WACV), Lake Tahoe, NV, 12–15 March 2018. 2018. URL: https://doi.org/10.1109/wacv.2018.00148 (date of access: 09.10.2025).

6. Language models are few-shot multilingual learners / G. I. Winata et al. Proceedings of the 1st workshop on multilingual representation learning, Punta Cana, Dominican Republic. Stroudsburg, PA, USA, 2021. URL: https://doi.org/10.18653/v1/2021.mrl-1.1 (date of access: 09.10.2025).

7. Recent Advances in Google Translate. Google Research - Explore Our Latest Research in Science and AI. URL: https://research.google/blog/recentadvances-in-google-translate/ (date of access: 09.10.2025).

8. Montuschi P., Valenzano A., Ciminiera L. Selection of token holding times in timed-token protocols. IEEE transactions on industrial electronics. 1990. Vol. 37, no. 6. P. 442–451. URL: https://doi.org/10.1109/41.103447 (date of access: 10.10.2025).

9. Natural language processing with transformers: building language applications with hugging face. O'Reilly Media, Incorporated, 2022 (date of access: 10.10.2025).

10. Attention Is All You Need / Vaswani A. et al. 2023. URL: https://arxiv.org/abs/1706.03762 (date of access: 10.10.2025).

11. Introducing ChatGPT. 2022. URL: https://openai.com/index/chatgpt/ (date of access: 10.10.2025).

12. Osborne C., Ding J., Kirk H. R. Correction to: The AI community building the future? A quantitative analysis of development activity on Hugging Face Hub. Journal of computational social science. 2024. URL: https://doi.org/10.1007/s42001-024-00308-0 (date of access: 11.10.2025).

13. RoFormer: enhanced transformer with rotary position embedding / J. Su et al. Neurocomputing. 2023. P. 127063. URL: https://doi.org/10.1016/j.neucom.2023.127063 (date of access: 11.10.2025).

14. Getting Started With Embeddings. Hugging Face – The AI community building the future. URL: https://huggingface.co/blog/getting-started-withembeddings (date of access: 11.10.2025).

15. How GitHub Copilot is getting better at understanding your code. The GitHub Blog. URL: https://github.blog/2023-05-17-how-github-copilot-isgetting-better-at-understanding-your-code/ (date of access: 12.10.2025).

16. Shafiq M., Gu Z. Deep residual learning for image recognition: a survey. Applied sciences. 2022. Vol. 12, no. 18. P. 8972. URL: https://doi.org/10.3390/app12188972 (date of access: 12.10.2025).

17. Transformers for natural language processing and computer vision: explore generative AI and large language models with hugging face, chatgpt, GPT-4V, and DALL-E 3. de Gruyter GmbH, Walter, 2024 (date of access: 13.10.2025).

18. Ziaee A., ÇAno E. Batch Layer Normalization A new normalization layer for CNNs and RNNs. ICAAI 2022: 2022 the 6th international conference on advances in

artificial intelligence, Birmingham United Kingdom. New York, NY, USA, 2022. URL: https://doi.org/10.1145/3571560.3571566 (date of access: 13.10.2025).

19. Layer Normalization / Ba J. L. et al. 2016. URL https://arxiv.org/abs/1607.06450 (date of access: 13.10.2024).

20. Deep Residual Learning for Image Recognition / He K. et al. 2015. URL: https://arxiv.org/abs/1512.03385 (date of access: 13.10.2025).

21. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding / Devlin J. et al. 2019. URL: https://arxiv.org/abs/1810.04805 (date of access: 12.10.2025).

22. Token selection strategies: Top-K, Top-P, and Temperature. Peter Chng. URL: https://peterchng.com/blog/2023/05/02/token-selection-strategiestop-k-top-p-and-temperature/ (date of access: 13.10.2025).

23. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer / Raffel C. et al. 2023. URL: https://arxiv.org/abs/1910.10683 (date of access: 13.10.2025).

24. DeffusionDB. 14 Million Image-Prompt Pairs. URL: https://huggingface.co/datasets/poloclub/diffusiondb (date of access: 13.01.2025).

25. Vidrih M. New AI Breakthrough – Google's Muse: Text-To-Image Generation via Masked Generative Transformers. 2023. URL: https://vidrihmarko.medium.com/new-ai-breakthrough-googles-muse-text-toimage-generation-via-masked-generative-transformers-6bedd5b0cad9 (date of access: 13.10.2025).

26. Transformers. Hugging Face – The AI community building the future. URL: https://huggingface.co/docs/transformers/index (date of access: 13.10.2025).

27. Fotedar N. A., Wang J. H. Bumblebee: Text-to-Image Generation with Transformers. URL: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/custom/15709283.pdf (date of access: 13.10.2025).

28. Дімура М. Нейромережі для малювання та створення зображень – огляд 2023 року. 2023. https://www.site2b.ua/web-blog/nejroseti-dlyarisovaniya-i-sozdaniya-izobrazhenij.html (дата звернення: 13.10.2025).

29. Stable diffusion text-image generation / S. S. alam.A et al. Interantional journal of scientific research in engineering and management. 2023. Vol. 07, no. 02. URL: https://doi.org/10.55041/ijsrem17744 (date of access: 13.10.2025).

30. Text to image generation using GAN / R. Sawant et al. SSRN electronic journal. 2021. URL: https://doi.org/10.2139/ssrn.3882570 (date of access: 13.10.2025).

31. AI image generation using text prompt. International research journal of modernization in engineering technology and science. 2025. URL: https://doi.org/10.56726/irjmets77332 (date of access: 10.12.2025).

32. Huang Youwen, Zhou Bin, Tang Xin. Text image generation method with scene description. Laser & optoelectronics progress. 2021. Vol. 58, no. 4. P. 0410012. URL: https://doi.org/10.3788/lop202158.0410012 (date of access: 10.12.2025).

33. Text-to-Image generation using deep learning. International journal of advanced innovative technology in engineering. 2025. Vol. 10, no. 2. P. 113–115. URL: https://doi.org/10.5281/zenodo.15407889 (date of access: 10.12.2025).

34. Text-conditioned image generation using diffusion models / D. Srinivasan et al. Multimedia tools and applications. 2025. URL: https://doi.org/10.1007/s11042-025-20990-0 (date of access: 10.12.2025).

35. Text to image generation using BERT and GAN / Kavitha Soppari et al. International journal of science and research archive. 2025. Vol. 14, no. 1. P. 720–725. URL: https://doi.org/10.30574/ijsra.2025.14.1.0137 (date of access: 10.12.2025).

36. Tiwari R., K. C. Text to image generation using machine learning. INTI journal. 2024. Vol. 2022, no. 1. URL: https://doi.org/10.61453/jods.v2024no60 (date of access: 10.12.2025).

37. K H M. A., B E M., A S M. C. Python - based image processing. International journal of scientific research and management. 2021. Vol. 9, no. 11. P. 635–638. URL: https://doi.org/10.18535/ijsrm/v9i11.ec03 (date of access: 10.12.2025).

38. Pfeiffer S. JavaScript API. The definitive guide to HTML5 video. Berkeley, CA, 2010. P. 81–134. URL: https://doi.org/10.1007/978-1-4302-3091-5_4 (date of access: 10.12.2025).

39. Firdian M. I., Darwiyanto E., Adrian M. Web scraping with HTML DOM method for website news API creation. JIPI (jurnal ilmiah penelitian dan pembelajaran informatika). 2022. Vol. 7, no. 4. P. 1211–1219. URL: https://doi.org/10.29100/jipi.v7i4.3235 (date of access: 10.12.2025).

40. Дакетт Д., HTML та CSS: Дизайн та розробка. Вілямс, 2018. 480 с. URL: http://vk.academy.lv/file/Dakett_J_HTML_i_CSS_Razrabotka_i_dizayn_web_stranic.pdf (data of access: 10.12.2025).

41. Gupta S. Installing PyCharm IDE. Practical python projects. Berkeley, CA, 2022. URL: https://doi.org/10.1007/978-1-4842-8202-1_2 (date of access: 10.12.2025).

ДОДАТОК А

Основний файл генерації

```
from fastapi import FastAPI, Form
from fastapi.responses import HTMLResponse, JSONResponse
from fastapi.staticfiles import StaticFiles
import uvicorn
import io
import base64
from PIL import Image
import numpy as np
import torch
import re

LORA_WEIGHTS_PATH = "weights/my_dataset_lora.safetensors"


def is_english(text: str) -> bool:

    return re.fullmatch(r"[A-Za-z0-9\s\.,!?'\-_:;()]+", text) is not None


from transformers import MarianMTModel, MarianTokenizer

from models.image_decoder import ImageDecoderWrapper
from models.superres import SuperResWrapper

app = FastAPI(title="Friday — text to image")

app.mount("/static", StaticFiles(directory="app/static"), name="static")


if torch.cuda.is_available() and torch.cuda.get_device_properties(0).major >= 8:
    DEVICE = "cuda"
    DTYPE = torch.bfloat16
    print("Running on CUDA with bfloat16 (recommended).")
elif torch.cuda.is_available():
    DEVICE = "cuda"
    DTYPE = torch.float16
    print("Running on CUDA with float16 (using half precision).")
else:
    DEVICE = "cpu"
    DTYPE = torch.float32
    print("Running on CPU (generation will be very slow).")


decoder = ImageDecoderWrapper(device=DEVICE, lora_path=LORA_WEIGHTS_PATH)

superres = SuperResWrapper(device=DEVICE)


print("Loading translator: Helsinki-NLP/opus-mt-uk-en")
model_name = "Helsinki-NLP/opus-mt-uk-en"
tok = MarianTokenizer.from_pretrained(model_name)
translator = MarianMTModel.from_pretrained(model_name, use_safetensors=True).to(DEVICE)
```

```
def translate(prompt_uk: str) -> str:

    batch = tok(prompt_uk, return_tensors="pt", padding=True).to(DEVICE)
    gen = translator.generate(
        **batch,
        max_new_tokens=128,
        num_beams=4,
        early_stopping=True
    )
    return tok.batch_decode(gen, skip_special_tokens=True)[0]



@app.get("/", response_class=HTMLResponse)
async def index():
    try:
        html = open("app/ui.html", "r", encoding="utf-8").read()
    except FileNotFoundError:
        return HTMLResponse(
            "<html><body><h1>Error: 'app/ui.html' not found.</h1><p>Please ensure 'ui.html' is in the 'app/'
directory.</p></body></html>")
    return HTMLResponse(content=html)


@app.post("/generate")
async def generate(prompt: str = Form(...)):
    print(f"Original prompt: {prompt}")

    if is_english(prompt):
        prompt_en = prompt
        print("Detected EN → skip translation")
    else:
        prompt_en = translate(prompt)
        print(f"Translated prompt: {prompt_en}")

    with torch.no_grad():
        low_res_imgs = decoder.decode(prompt_en)

        print(
            f"Low-res image stats (from decoder): Shape={low_res_imgs.shape}, Dtype={low_res_imgs.dtype},
Min={low_res_imgs.min()}, Max={low_res_imgs.max()}")
        up = superres.upscale(low_res_imgs[0], prompt=prompt_en)
    final_processed_array_rgb = up[0]
    pil_final = Image.fromarray(final_processed_array_rgb)
    buf = io.BytesIO()
    pil_final.save(buf, format="JPEG", quality=90)
    buf.seek(0)
    b64 = base64.b64encode(buf.read()).decode("utf-8")

    print("Image generation complete.")

    return JSONResponse({"image_base64": b64, "mime_type": "image/jpeg"})
if __name__ == "__main__":
    uvicorn.run("api:app", host="0.0.0.0", port=8000, reload=False)
```

ДОДАТОК Б

Файл обробки текта

```
import torch
from transformers import XLMRobertaTokenizer, XLMRobertaModel

class TextEncoder:
    def __init__(self, model_name="xlm-roberta-base", device="cpu"):
        self.device = device
        print(f"Loading XLM-R model: {model_name}")
        self.tokenizer = XLMRobertaTokenizer.from_pretrained(model_name)
        self.model = XLMRobertaModel.from_pretrained(model_name).to(device)
        self.model.eval()

    def encode(self, text: str):
        tokens = self.tokenizer(
            text,
            return_tensors="pt",
            truncation=True,
            padding=True,
            max_length=128
        ).to(self.device)

        with torch.no_grad():
            outputs = self.model(**tokens)

        pooled = outputs.last_hidden_state[:, 0, :] # (1, 768)

        return {
            "last_hidden": outputs.last_hidden_state,
            "pooled": pooled
        }
```

ДОДАТОК В

Файл обробки зображення

```
import torch
import numpy as np
from diffusers import StableDiffusionUpscalePipeline
from PIL import Image


class SuperResWrapper:
    def __init__(self, device: str):
        self.device = device

        if torch.cuda.is_available() and torch.cuda.get_device_properties(0).major >= 8:
            DTYPE = torch.bfloat16
        elif torch.cuda.is_available():
            DTYPE = torch.float16
        else:
            DTYPE = torch.float32

        print(f"🔄 Завантаження Stable Diffusion Upscaler з dtype: {DTYPE}")
        model_id = "stabilityai/stable-diffusion-x4-upscaler"
        self.pipeline = StableDiffusionUpscalePipeline.from_pretrained(
            model_id,
            torch_dtype=DTYPE,
            safety_checker=None,
            use_safetensors=True
        ).to(self.device)

        self.pipeline.enable_attention_slicing()

    @torch.no_grad()
    def upscale(self, low_res_img: np.ndarray, prompt: str = "") -> np.ndarray:

        low_res_pil = Image.fromarray(low_res_img)


        upscaled_images = self.pipeline(
            prompt=prompt,
            image=low_res_pil,
            num_inference_steps=20,
            output_type="np"
        ).images


        upscaled_images = upscaled_images * 255.0
        final_output = upscaled_images.astype(np.uint8)

        return final_output
```

ДОДАТОК Г

Файл завантаження моделі

```
import torch
from diffusers import StableDiffusionPipeline
import numpy as np
import os


class ImageDecoderWrapper:
    def __init__(self, device="cuda", z_dim=512, out_size=512, lora_path=None):
        self.device = device
        self.out_size = out_size
        self.lora_path = lora_path
        if torch.cuda.is_available() and torch.cuda.get_device_properties(0).major
>= 8:
            DTYPE = torch.bfloat16
        elif torch.cuda.is_available():
            DTYPE = torch.float16
        else:
            DTYPE = torch.float32

        print(f"🔄 Завантаження базової моделі Stable Diffusion з dtype: {DTYPE}")
        self.pipe = StableDiffusionPipeline.from_pretrained(
            "runwayml/stable-diffusion-v1-5",
            torch_dtype=DTYPE,
            safety_checker=None,
        ).to(device)

        if self.lora_path:
            if os.path.exists(self.lora_path):
                print(f"🔗 Завантаження та застосування LoRA ваг з:
{self.lora_path}")

                adapter_name = "custom_lora_adapter"
                self.pipe.load_lora_weights(self.lora_path,
adapter_name=adapter_name)

                self.pipe.set_adapters([adapter_name])
                print("✅ LoRA ваги успішно застосовані.")
            else:
                print(f"❌ Помилка: LoRA файл не знайдено за шляхом:
{self.lora_path}. Продовжуємо з базовою моделлю.")

        self.pipe.enable_attention_slicing()

        print("✅ Модель завантажена та готова до роботи.")
    @torch.no_grad()
    def decode(self, prompt: str):

        result = self.pipe(prompt=prompt, num_inference_steps=50)

        image = result.images[0]

        img = np.array(image, dtype=np.uint8)[None, ...]  # Shape: (1, 512, 512,
3)
        return img
```

ДОДАТОК Д

Файл обробки числових масивів

```
import cv2
import numpy as np
from PIL import Image
import os

def load_image_cv(path: str, target_size: int = None):
    img = cv2.imread(path, cv2.IMREAD_COLOR)
    if img is None:
        raise FileNotFoundError(f"Image not found: {path}")
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    if target_size is not None:
        img = cv2.resize(img, (target_size, target_size),
interpolation=cv2.INTER_AREA)
    return img

def save_image_pil(np_img, path: str):
    Image.fromarray(np_img).save(path)

def normalize_img_for_model_uint8_to_float(img_uint8):
    return img_uint8.astype("float32") / 255.0

def ensure_dir(path: str):
    os.makedirs(path, exist_ok=True)
```