Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
Кафедра Комп'ютерних інтелектуальних технологій та систем

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Нейромережева модель для генерації поетичних текстів
(тема)

Виконав:
здобувач ІІ року навчання
групи КІТм-23-1
Писаренко С. В.
(прізвище, ініціали)

Спеціальність 123 «Комп'ютерна інженерія»
Тип програми освітньо-професійна
Освітня програма Комп'ютерні інтелектуальні технології
Керівник проф. Руденко О. Г.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ О. Г. Руденко
(підпис)

2024

Харківський національний університет радіоелектроніки

Факультет	Комп'ютерної інженерії та управління
Кафедра	Комп'ютерних інтелектуальних технологій та систем
Рівень вищої освіти	другий (магістерський)
Спеціальність	123 Комп'ютерна інженерія
Тип програми	освітньо-професійна
Освітня програма	Комп'ютерні інтелектуальні технології

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
« 28 » жовтня 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Писаренку Станіславу Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Нейромережева модель для генерації поетичних текстів

затверджена наказом по університету від « 28 » жовтня 2024 р. № 1156 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 19. 01. 2025 р.

3. Вихідні дані до роботи (проекту)
1) Бібліотека transformers, peft
2) Фреймворк Flask
3) Модель Gemma-2-2b
4) Платформа Kaggle
5) Набір даних української поезії

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз стану питання
2) Технології та підходи для реалізації моделей генерації тексту
3) Особливості навчання мовних моделей
4) Реалізація рішення генерації. Навчання моделі. Розробка додатку

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням кафедри)

Слайд-презентація – 12 слайдів

6. Консультанти розділів роботи (п.6 включається до завдання за наявністю консультантів згідно до наказу, зазначеному у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термів виконання етапів роботи	Примітка
1	Видача та узгодження теми проекту	28.10.2024	Виконано
2	Огляд стану проблеми та постановка задачі	01.11-10.11.2024	Виконано
3	Аналіз літератури	14.11-28.11.2024	Виконано
4	Вибір архітектури моделі генерації	30.11-11.12.2024	Виконано
5	Збір та обробка даних	12.12-19.12.2024	Виконано
6	Тренування та тестування моделі	20.12-30.12.2024	Виконано
7	Розробка та тестування системи	30.12-31.12.2024	Виконано
8	Оформлення записки	05.01-18.01.2025	Виконано
9	Подання до ЕК	19.01.2025	Виконано
10	Захист роботи		

Дата видачі завдання 28 жовтня 2024 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф.. Руденко О. Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 88 с., 18 рис., 1 дод., 3 табл., 16 джерел.

ПОЕЗІЯ, ГЕНЕРАЦІЯ ТЕКСТУ, ВЕБ-СЕРВЕР, ПАЙТОН, ВЕЛИКІ МОВНІ МОДЕЛІ, ОБРОБКА ТЕКСТУ, ТРАНСФОРМЕРНА АРХІТЕКТУРА, РЕКУРЕНТНІ НЕЙРОННІ МЕРЕЖІ.

У магістерській кваліфікаційній роботі представлено рішення генерації україномовних поетичних текстів за заданим початком, на основі мовної моделі з архітектурою трансформера.

Метою кваліфікаційної роботи є дослідження підходів до побудови мовних моделей та їх адаптація для генерації поетичних текстів.

Об’єктом дослідження цієї роботи є мовні моделі для текстової генерації.

Предметом дослідження у даній роботі є налаштування великих мовних моделей для генерації поетичних текстів із урахуванням тематичних особливостей.

У роботі проведено аналіз і порівняння різних підходів до побудови мовних моделей, показано їх сильні та слабкі сторони щодо генерації тексту, зокрема поезії. Було обрано найбільш оптимальну модель на основі архітектури декодера трансформера. Визначено та реалізовано низку метрик, для оцінювання якості згенерованих віршів.

ABSTRACT

Explanatory note of qualification work: 88 pages, 18 figures, 1 appendices, 3 table, 16 sources.

POETRY, TEXT GENERATION, WEB SERVER, PYTHON, LARGE LANGUAGE MODELS, TEXT PROCESSING, TRANSFORMER ARCHITECTURE, RECURRENT NEURAL NETWORKS.

The master's thesis presents a solution for generating Ukrainian poetic texts based on a given prompt, using a language model with a transformer architecture.

The goal of the qualification work is to explore approaches to building language models and their adaptation for generating poetic texts.

The object of this study is language models for text generation.

The subject of the research is the fine-tuning of large language models for generating poetic texts, taking into account thematic features.

The thesis analyzes and compares different approaches to building language models, highlighting their strengths and weaknesses in text generation, particularly poetry. The most optimal model based on the transformer decoder architecture was selected. A set of metrics for evaluating the quality of generated poems was defined and implemented.

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
Кафедра Комп'ютерних інтелектуальних технологій та систем

АНОТАЦІЯ
КВАЛІФІКАЦІЙНОЇ РОБОТИ

Рівень вищої освіти другий (магістерський)

Нейромережева модель для генерації поетичних текстів
(тема)

Виконав:
здобувач II року навчання
групи КІТм-23-1
Писаренко С. В.
(прізвище, ініціали)

Спеціальність 123 «Комп'ютерна інженерія»

Тип програми освітньо-професійна

Освітня програма Комп'ютерні інтелектуальні технології

Керівник проф. Руденко О. Г.
(посада, прізвище, ініціали)

2024 р.

Писаренко С. В. Тема магістерської кваліфікаційної роботи – Нейромережева модель для генерації поетичних текстів.

У магістерській кваліфікаційній роботі представлено рішення генерації україномовних поетичних текстів за заданим початком, на основі мовної моделі з архітектурою трансформера.

Метою кваліфікаційної роботи є дослідження підходів до побудови мовних моделей та їх адаптація для генерації поетичних текстів.

Об’єктом дослідження цієї роботи є мовні моделі для текстової генерації.

Предметом дослідження у даній роботі є налаштування великих мовних моделей для генерації поетичних текстів із урахуванням тематичних особливостей.

Технології генерації текстів відкривають нові можливості для розвитку поетичного мистецтва, дозволяючи експериментувати з різними стилістичними прийомами та темами. Завдяки штучному інтелекту, люди можуть використовувати комп'ютерні алгоритми для створення нових творів, що може слугувати джерелом натхнення та допомагати визначати напрямок творчого процесу. Таким чином, дана технологія також сприяє збереженню та популяризації української поетичної спадщини в цифровому просторі.

Особливістю цієї роботи є дослідження налаштування мовних моделей для роботи з менш поширеними мовами, такими як українська. Це є важливий крок у напрямку розвитку штучного інтелекту для невеликих культурних і мовних груп. Такі групи часто залишаються поза увагою під час створення великих моделей через обмежену кількість даних, що зумовлена меншою кількістю носіїв мови. Крім того, дослідження компактних моделей, таких як Gemma 2B, є перспективним у контексті роботи на обмежених обчислювальних ресурсах. Тематика роботи є актуальною як з погляду розвитку технологій штучного інтелекту в контексті культурної спадщини, так і в розширенні їхнього застосування для мов з меншими корпусами текстів.

В першому розділі кваліфікаційної роботи проведено аналіз стану

питання з теми дослідження. Розглянуто етапи розвитку обробки природної мови, основні типи задач в аналізі текстової інформації, а також області, в яких використовуються великі мовні моделі.

Визначено, що аналіз тексту, як метод перетворює неструктуровані дані на структуровані, що полегшує подальші маніпуляції ними. Наприклад, за допомогою словників виділяються власні імена, а текстові дані можуть бути переведені в кількісні. Аналіз тексту використовується для інтерпретації змісту і прийняття обґрунтованих висновків. У бізнесі цей метод застосовується для семантичного аналізу, машинного перекладу, пошукових запитів та підтримки клієнтів.

З’ясовано, що сучасні методи обробки природної мови беруть початок від статистичних моделей та систем, заснованих на правилах, що спиралися на вручну створені функції. Однак вони не змогли охопити всю складність та нюанси мови. Перехід до методів машинного навчання став важливою зміною, дозволивши моделям вивчати шаблони безпосередньо з даних. Вбудовування слів, як Word2Vec, змінили підхід до представлення слів у векторних просторах, де схожі слова мають схоже представлення. Це значно покращило ефективність моделей в обробці мови. Рекурентні моделі дозволили враховувати залежності між словами в тексті, але їхня обмеженість полягала в складності роботи з довгими послідовностями та проблемі зникання градієнтів. Механізм уваги, що з'явився в архітектурі Transformer, дозволив ефективно виявляти довготривалі залежності у текстах і розпаралелювати обчислення, що дало змогу працювати з великими об'ємами даних. Досягнення в апаратному забезпеченні, зокрема збільшення потужності TPU та GPU, надало можливість створювати моделі з великою кількістю параметрів. Великі мовні моделі впроваджуються у різні сфери та галузі, відкриваючи нові можливості для аналізу текстів.

В другому розділі проведено аналіз технологій та підходів на яких базуються мовні моделі. Завдання мовних моделей полягає у визначенні

ймовірності для послідовності слів, що дозволяє передбачати ймовірність наступного слова. Якщо початкова послідовність є якісною, згенероване речення буде виглядати логічним і природним. Такі моделі навчаються на великих текстових масивах і використовують отриманий контекст для прогнозування наступного слова на основі нового вводу. Розглянуто етапи розвитку архітектур, рекурентні, лстм, трансформери.

Методи на основі рекурентних нейронних мереж та трансформерів мають свої переваги та недоліки. Виявлено, що основною проблемою рекурентних нейромереж є затухання градієнтів при роботі з довгими послідовностями. Їх покращена модифікація з довготривалою - короткостроковою пам’яттю частково вирішує це завдяки механізму довготривалої пам'яті, що дозволяє зберігати контекст, який втрачається у стандартних рекурентних мережах. Однак навіть така архітектура демонструє обмежену ефективність при дуже довгих контекстах — сигнал, що проходить через численні часові кроки, згасає, а механізм воріт має обмежену здатність зберігати всю інформацію. Трансформери з механізмом уваги долають ці обмеження, ефективно обробляючи залежності між будь-якими елементами послідовності, незалежно від їхньої віддаленості. Це забезпечує перевагу в утриманні контексту, що є вирішальним для задач генерації текстів, де важлива логічна послідовність думок.

В третьому розділі представлено особливості тонкого налаштування та процесу генерації у великих мовних моделях. Файнтюнінг великих мовних моделей є ресурсозатратним через оновлення численних параметрів. Досліджено параметрично ефективні підходи, які оновлюють лише обрані параметри, залишаючи решту незмінними. Одним із найпоширеніших методів є низькорангова адаптація. Суть даного методу полягає в замороженні основних шарів трансформера, а для тонкого налаштування оновлюється лише їхня низькорангова апроксимація — невеликі додаткові матриці з набагато меншою кількістю параметрів. Визначено, що перед поданням тексту у модель потрібна

його токенізація для розділення на токени та створення словника. Виявлено переваги такого методу токенізації, як кодування пар байтів — адаптація до різних мов та здатність працювати з рідкісними словами.

З’ясовано, що за допомогою методу стандартної випадкової вибірки ймовірностей токенів прогнозованих моделлю, часто генеруються алогічні фрази через вибір слів із низькою ймовірністю, що спільно мають значний вплив. Було розглянуто методи які балансують між якістю та різноманітністю тексту, фокусуючись на більш імовірних словах або враховуючи середні за ймовірністю варіанти.

Четвертий розділ присвячено підготовці даних, тренуванню моделі, та виділенню метрик якості генерації і їх проходження, а також реалізації простого веб додатку для демонстрації рішення. Визначено, що для задачі генерації поезії найкращим рішенням є використання трансформерних моделей. Було обрано дві моделі для донавчання на віршах: базову модель Gemma 2b та попередньо натреновану Gemma 2b на українських текстах. Приведені результати тренування свідчать про кращу якість базової моделі.

Модель може продовжувати розвивати сюжет з об’єктами із поданих на вхід рядків, але інколи генерує неповні фрази, що ускладнює їх інтерпретацію, що призводить до слабкої зв'язності окремих фраз, але в цілому згенерований текстовий потік тримається навколо теми. Метрики показали, що частина згенерованих віршів має меншу попарну схожість та більшу варіативність в іменниках порівняно з оригінальними поезіями, тому було запропоновано використовувати техніку cherry-picking, генерувати, доки не з’явиться гарний варіант. Було розроблено протестовано веб додаток, для реалізації якого обрано фреймворк Flask, що працює на платформі Kaggle, а також використовуються тунель ngrok, для обходу обмеження доступу до сервера, розгорнутого у сердовищі Kaggle.

ПОЕЗІЯ, ГЕНЕРАЦІЯ ТЕКСТУ, ВЕБ-СЕРВЕР, ПАЙТОН, ВЕЛИКІ МОВНІ МОДЕЛІ, ОБРОБКА ТЕКСТУ, ТРАНСФОРМЕРНА АРХІТЕКТУРА, РЕКУРЕНТНІ НЕЙРОННІ МЕРЕЖІ.

Публікації здобувача використані в магістерській роботі:

1. Писаренко С. В. Реалізація процедури аутенфікації людини на базі Face ID. *Радіоелектроніка та молодь у XXI столітті*: матеріали 27-го міжнар. лол. форуму. Харків, 2023. С. 68.

Публікації наукового керівника за темою магістерської роботи:

1. Руденко О., Бодянський Є. Штучні нейронні мережі : навч. посіб. Харків : Компанія СМІТ, 2006. 404 с.

ЗМІСТ

ВСТУП 14
1 АНАЛІЗ СТАНУ ПИТАННЯ 16
1.1 Історія розвитку обробки природної мови 16
1.2 Типи задач в аналізі текстової інформації 18
1.3 Сфери застосування великих мовних моделей 22
2 АНАЛІЗ ТЕХНОЛОГІЙ ТА ПІДХОДІВ ДЛЯ РЕАЛІЗАЦІЇ МОДЕЛІ 25
2.1 Рекурентні нейронні мережі 25
2.2 Двонаправлені та багатошарові RNN 31
2.3 Мережі довготривалої короткострокової пам’яті 34
2.4 Архітектура трансформера 40
3 ОСОБЛИВОСТІ НАВЧАННЯ ТА ГЕНЕРАЦІЇ В LLM 47
3.1 Тонке налаштування моделей 47
3.2 Низькорангова адаптація 48
3.3 Токенізація в великих мовних моделях 51
3.4 Вибір наступного слова під час генерації в LLM 54
4 РЕАЛІЗАЦІЯ РІШЕННЯ ГЕНЕРАЦІЇ ПОЕЗІЇ ТА АНАЛІЗ ЙОГО ЯКОСТІ . 59
4.1 Застосовані технології та інструменти 59
4.2 Підготовка даних 60
4.3 Характеристики моделі та її навчання 63
4.4 Виділення та оцінка метрик якості генерації поезії 70
4.5 Реалізація додатку 73
4.6 Тестування додатку 76
ВИСНОВКИ 78
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 80

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

BPE – алгоритм токенізації текстів, кодування пар байтів (англ., Byte Pair Encoding);

TTR – метрика співвідношення токенів, що використовується для вимірювання лексичної різноманітності тексту (англ., Type-Token Ratio)

LLM – велика мовна модель (англ., Large Language Model)

RNN – рекурентна нейромережа, тип нейронної мережі, який використовує циклічні зв'язки для обробки послідовних даних(англ., Recurrent Neural Network)

LSTM - тип рекурентної нейронної мережі, який призначений для роботи з послідовними даними та здатний запам'ятовувати інформацію тривалий час (англ., Long Short-Term Memory)

SOTA – найсучасніші та найрозвиненіші досягнення в певній галузі (англ., State of The Art)

LoRA – метод для налаштування параметрів великих мовних моделей, який використовує низькорангові апроксимації (англ., Low Rank Adaptation)

API – набір інструментів та протоколів, що дозволяють різним програмам взаємодіяти між собою (англ., Application Program Interface)

PEFT – бібліотека, що надає інструменти для параметрично ефективного тонкого налаштування великих мовних моделей (англ. Parameter-Efficient Fine-Tuning)

ВСТУП

Поезія займає особливе місце в культурній спадщині України, відображаючи багатогранність її мови, глибину історичних подій та різноманіття емоційних переживань. Від класиків, таких як Тарас Шевченко, Іван Франко та Леся Українка, до сучасних поетів, українська поезія завжди залишалася джерелом натхнення та самовираження.

Протягом останніх двох десятиліть розвиток штучного інтелекту досяг значних висот, що призвело до появи численних нових технологій, що знаходять застосування в багатьох сферах людської діяльності. Однією з таких технологій є генерація текстів, зокрема поезії.

Генерація поезії за допомогою штучних нейронних мереж є сучасним напрямом, що поєднує досягнення в галузях лінгвістики, штучного інтелекту та комп'ютерних наук. Спостерігається зростаючий інтерес до автоматизації творчих процесів, зокрема у сфері літератури, мистецтва та культурної спадщини. Поетичні тексти мають особливу естетичну природу, що потребує від моделей здатності розуміти та відтворювати ритмічні, стилістичні й емоційні особливості.

Технології генерації текстів відкривають нові можливості для розвитку поетичного мистецтва, допомагають поетам експериментувати з різноманітними стилістичними прийомами та темами. Завдяки штучному інтелекту, тепер для створення нових творів, поети чи звичайні люди можуть звертатися до комп'ютерних алгоритмів, що можуть слугувати джерелом натхнення та задавати певний напрямок для подальшого розвитку творчого процесу. Цим самим дана технологія сприяє збереженню та популяризації української поетичної спадщини у цифровому вимірі.

Поезія – це особливий вид тексту, який вимагає від моделі не тільки синтаксичної та граматичної правильності, але й творчого підходу, емоційної

глибини та здатності до метафоричного мислення. Найпоширенішим підходом до генерації поезії є авторегресійні моделі - великі мовні моделі на базі трансформерів, такі як GPT, Gemma, LLaMa. Ці моделі показують високу якість генерації поетичних текстів, дозволяючи створювати вірші різних жанрів і стилів.

Іншим підходом до генерації поезії є рекурентні нейронні мережі. RNN добре працюють з послідовними даними та можуть зберігати контекст на коротких відрізках тексту, проте вони мають обмежену здатність утримувати інформацію на великих відстанях через проблему зникнення градієнта. Це ускладнює генерацію довгих і складних віршів, де важливо зберігати контекст та римованість. Великі мовні моделі на трансформерній архітектурі, вирішують цю проблему за рахунок механізму уваги, що дозволяє моделі одночасно враховуват и всі слова у вірші та створювати більш узгоджені та глибокі поетичні тексти.

Тож, зважаючи на обмежені обчислювальні ресурси було обрано маленьку, у порівнянні з більш масштабними моделями, але передову Gemma 2B, для досягнення оптимального результату. Вона демонструє здатність створювати зв’язні та контекстуально релевантні поетичні тексти, що робить її гарним вибором для поставленого завдання в умовах обмежених ресурсів.

1 АНАЛІЗ СТАНУ ПИТАННЯ

1.1 Історія розвитку обробки природної мови

Сучасна обробка природної мови бере свій початок від статистичних моделей та систем, заснованих на правилах, що спиралися на створені вручну функції. Хоча ці підходи продемонстрували непогані результати у певних задачах, проте вони не змогли осягнути певні нюанси складності мови. Перехід до методів, заснованих на машинному навчанні, став важливою зміною парадигми, дозволивши моделям вивчати шаблони безпосередньо з даних.

Словесні вбудовування, такі як Word2Vec, змінили підхід до представлення слів у векторних просторах. Це вивчене уявлення тексту означає, що слова із схожим значення, мають схоже представлення. Тобто, слова представляються в системі координат, де подібні слова розташовуються ближче одне до одного. Заміна розріджених одноразових кодувань на щільні безперервні представлення суттєво покращила продуктивність систем обробки природної мови.

Механізм уваги став вирішальним компонентом з виходом архітектури Transformer. Запропонована в основоположній статті 2017 року "Attention is All You Need" [1], модель Transformer показала ефективність механізмів самоуваги для вловлювання довготривалих залежностей у тексті. Також даний архітектурний підхід сприяв розпаралелюванню обчислень, дозволяючи ефективно навчатись на досить великих об’ємах даних.

Досягнення в сфері апаратного забезпечення, а саме збільшення потужності тензорних процесорів (TPU) та графічних процесорів (GPU), стали вирішальними у збільшенні розміру та складності моделей обробки природної мови. Ці чинники дозволили навчати моделі з небаченою раніше кількістю параметрів. І як результат, з’явились такі моделі, як Grok на 314 мільярдів і

GPT-3 та 4 з 175 мільярдами та 1.7 трильйонами параметрів відповідно.

Ряд моделей GPT, що розроблені компанією OpenAI, є значним кроком в еволюції LLM. Ці моделі використовують стратегію попереднього навчання та тонкого налаштування: спочатку вони попередньо навчаються на великих корпусах мультимовних текстових даних, а потім налаштовуються для вирішення конкретних завдань. GPT-3, проявив надзвичайну продуктивність у широкому спектрі завдань обробки природної мови.

Збільшення розміру моделей має свої недоліки. Навчання громіздких моделей потребує значних обчислювальних ресурсів і може створювати проблеми при розгортанні через їх високу ресурсомісткість. Крім того, етичні питання, пов'язані з упередженнями в даних та поведінці моделей, стають все більш важливими, оскільки великі мовні моделі набувають широкого поширення.

З розвитком галузі обробки природної мови, еволюція великих мовних моделей, продовжиться, рисунок 1.1. Майбутні дослідження будуть зосереджені на вдосконаленні методів навчання, вирішенні етичних питань та вивченні нових архітектур для подальшого підвищення можливостей цих моделей.

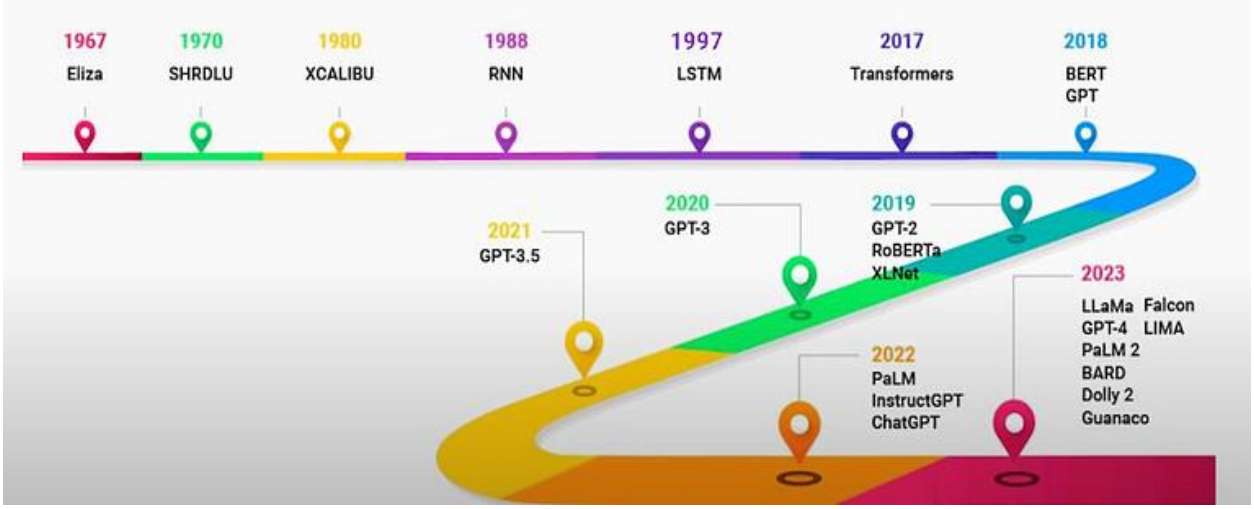

Рисунок 1.1 – Розвиток мовних моделей

1.2 Типи задач в аналізі текстової інформації

Аналіз тексту - це метод в області машинного навчання та штучного інтелекту, що застосовується для вилучення цінної інформації з людської мови розумним та ефективним способом. Дослідники та розробники можуть використовувати цей метод для структурування різноманітних та неорганізованих даних. У цьому процесі документи розбиваються на складові частини для зручного управління, тобто неструктурований текст перетворюється у структуровані дані. Деякі елементи, такі як власні імена, відокремлюються за допомогою списку слів зі словника. Аналіз текстових даних дозволяє перетворювати якісні дані в кількісні. Аналіз тексту — це дослідницький підхід, який використовується для отримання розумних висновків шляхом інтерпретації змісту. У бізнесі аналіз тексту охоплює такі теми, як семантичний аналіз пошукових запитів і управління контентом для збору інформації, машинний переклад, а також задачу питання – відповідь, яка є популярною в службах підтримки тощо.(рисунок 1.2).

Існують як базові стратегії аналізу тексту так і більш просунуті методи, які використовують машинне навчання, статистичні та лінгвістичні методи, що дозволяють отримувати вагомі ідеї з текстових даних.

Рисунок 1.2 – Найпоширеніші типи задач аналізу текстів

У даному розділі описуються різноманітні типи задач аналізу тексту, від простих до складніших, що надає повний огляд методів, для обробки та розуміння текстової інформації:

Попередня обробка тексту включає видалення зайвих слів і приведення схожих слів до однієї форми, що зменшує час, необхідний для аналізу, та підвищує ефективність. Текстові дані зазвичай потребують очищення від шуму, такого як пунктуація чи стоп-слова, які мають низьку інформаційну цінність. Для скорочення слів до їхніх базових форм можна застосувати такі методи, як формування основи та лематизації - процесу перетворення слова до його словникової форми, подібний до стемінгу. Однак, на відміну від стемінгу, лематизація знаходить базову словникову форму слова, а не кореневу. Наприклад, слова "зеленого", "зелені", "зелененький" можуть бути зведені до слова "зелений" [2].

Класифікація текстів - це процес віднесення документів в одну чи декілька заздалегідь визначених певних категорій. Класифікація може здійснюватися власноруч або автоматично, за допомогою створеного набору правил чи із застосуванням методів машинного навчання, таких як наївний баєсів класифікатор, метод опорних векторів, та моделей глибокого навчання. Присвоєння категорій текстів, які можуть бути веб-сторінкою, книгою, статтею для ЗМІ та іншим, має безліч застосувань, наприклад: аналіз настроїв, позначення тем, класифікація новин, визначення мови, виявлення намірів, виявлення спаму.

Розпізнавання частин мови: це метод аналізу тексту, що використовується для визначення граматичної ролі кожного слова в реченні. POS-тегування позначає слова у тексті відповідно до їхніх частин мови, таких як іменник, дієслово, прикметник або прийменник, спираючись на контекст і граматичні правила. Наприклад, у реченні: "Кіт спить на дивані", результат POS-тегування може виглядати так: ('Кіт', 'NOUN'), ('спить', 'VERB'), ('на', 'ADP') і так далі. POS-тегування використовується для аналізу синтаксичної структури тексту,

розуміння семантики речення та покращення роботи алгоритмів обробки мови. Завдяки цьому процесу стає можливим точніше інтерпретувати текст для таких завдань, як машинний переклад, текстова аналітика.

Розпізнавання іменованих сутностей (NER): відомий також як ідентифікація сутностей, це метод аналізу тексту, що використовується для аналізу іменованих параметрів тексту і позначення їх у заздалегідь визначених категоріях, таких як назви місць, особи чи абревіатури. NER використовує граматичні техніки або методи числового аналізу, або обидва, щоб створити уривок з тексту. Аналіз NER відбувається, коли невизначене речення, наприклад: "Джек володіє фермою площею 300 акрів з 2001 року" перетворюється на визначене, як-от: "Джек(особа) володіє фермою площею 300 акрів у Нешвіллі(Місце) з 2001 року(Час)". NER покращує вилучення даних, спрощує пошук інформації та підтримує передові програми ШІ.

Узагальнення тексту. Цей тип задачі зводиться до процесу використання моделей глибокого та машинного навчання для синтезу великих текстів у їхні найважливіші частини. Його можна застосовувати до статичних, вже існуючих текстів, таких як наукові статті або новини.

Машинний переклад: це складна проблема, яка ускладнюється тим фактом, що існують тисячі мов, і кожна з них може мати дуже різні граматичні правила. Один із підходів полягає в тому, щоб перетворити формальні правила граматики для однієї мови, наприклад англійської, у структуру, що не залежить від мови, а потім перекласти її шляхом перетворення назад на іншу мову.

Тематичне моделювання. Його завдання - захоплювати семантичну інформацію за межами окремих слів; виявляти приховані теми чи теми в документах; відповідно анотувати документи; використовувати анотації для управління, узагальнення, пошуку та рекомендування вмісту. Часто використовується для виявлення прихованих семантичних структур в тілі тексту та дозволяє ефективно аналізувати великі обсяги текстів, об’єднуючи документи в кластери.

Кластеризація текстів. У сфері обробки природної мови (NLP) кластеризація тексту є фундаментальною та універсальною технікою, яка відіграє ключову роль у різних програмах, таких як пошук інформації, системи рекомендацій, організація вмісту та аналіз настроїв. Кластеризація тексту – це процес групування подібних документів або фрагментів тексту в кластери або категорії. Ця техніка дозволяє нам виявляти приховані закономірності, отримувати цінну інформацію та оптимізувати великі обсяги неструктурованих текстових даних. Найпоширенішими методоми кластеризації є кластеризація k-середніх, ієрархічна.

Глибоке навчання для аналізу тексту. Моделі глибинного навчання, такі як рекурентні мережі чи особливо трансформери значно поліпшили аналіз тексту. Нейронні мережі можуть застосовуватись до різних завдань обробки тексту, включаючи класифікацію, переклад, аналіз сентименту, розпізнавання іменованих сутностей та багато інших. Що і робить їх універсальним інструментом для обробки текстової інформації.

Зв’язування іменованих сутностей (NED): даний процес розширює можливості розпізнавання іменованих сутностей, включаючи не лише їх ідентифікацію, але й прив'язку до зовнішніх баз знань. Це сприяє усуненню неоднозначності сутностей і покращує результати аналізу тексту. Наприклад, у реченні "Стів Джобс був ключовим для успіху Apple" концепція стосується бренду Apple, а не фрукта. Це реалізується за допомогою прив’язки сутностей, але вимагає бази знань про сутності, яка міститиме всі згадки про сутності в тексті.

Етична сторона в аналізі текстів. Етичність в аналізі текстів є надзвичайно важливою, оскільки вона визначає взаємодію між аналітиками та даними, з якими вони працюють. Забезпечення конфіденційності, захист від упередженості та збереження справедливості має вирішальне значення для забезпечення довіри до результатів аналізу. Правильне врахування етичних аспектів також допомагає уникнути потенційних негативних наслідків, таких як

недопустиме використання аналітичної інформації або порушення приватності осіб. Тому важливо, щоб аналітики дотримувалися високих стандартів етики в усіх аспектах своєї роботи з текстовими даними. Ці методи виступають фундаментальними складовими для практичного застосування великих мовних моделей у текстовому аналізі.

1.3 Сфери застосування великих мовних моделей

Великі мовні моделі впроваджуються у різні сфери та галузі, відкриваючи нові можливості для аналізу текстів В даному розділі буде розглянуто різноманітні варіанти застосувань, де ці моделі продемонстрували вражаючу продуктивність.

Роздрібна торгівля та електронна комерція. Зазвичай LLM додаються до чат-ботів, що покращують взаємодію з клієнтами завдяки природній і персоналізованій комунікації. Аналізуючи дані клієнтів, великі мовні моделі надають індивідуальні пропозиції щодо товарів, що збільшує можливості для додаткових і перехресних продажів.

Освіта. Студенти часто використовують LLM для повторення знань з різних дисциплін. Моделі можна налаштувати для створення індивідуальних тестів та контрольних робіт. За таким підходом студенти покращують свої оцінки на тестах на 62%, згідно з даними Knewton [3]. Також ці моделі можуть відстежувати академічні результати студентів. Аналізуючи їхні оцінки, вони виявляють потенційно невстигаючих учнів і сповіщають викладачів для швидкого втручання.

Фінанси. Великі мовні моделі виконують роль фінансових консультантів, допомагаючи клієнтам приймати обґрунтовані рішення. Вони надають рекомендації щодо інвестицій, страхування та пенсійних планів. Майже 60% клієнтів Bank of America використовують продукти LLM для цих цілей. Також LLM допомагають фінансовим установам виявляти різні види шахрайських дій.

Аналізуючи історію платежів та шаблони транзакцій, вони знаходять аномалії у використанні кредитних карток та операціях, миттєво сповіщаючи власників рахунків.

Охорона здоров’я. Великі мовні моделі допомагають медичним фахівцям аналізувати дані пацієнтів. Це призводить до більш точних діагнозів та кращих результатів лікування. Вони досягають точності на рівні 83,3% шляхом аналізу історичних даних та схожих випадків [4]. Рішення, що використовують ці моделі, допомагають медичним організаціям у зборі, зберіганні та доступі до даних пацієнтів. Великі мовні моделі також аналізують цю інформацію, структурують та категоризують її. З цим підходом організації економлять час і зусилля на документацію.

Маркетинг. Маркетингові агентства використовують великі мовні моделі для створення персоналізованого контенту на основі поведінки, інтересів та фону кожної особи. Це використовується для створення настроєних електронних розсилок та рекомендацій продуктів. Підприємства, які використовують великі мовні моделі, спостерігають вищі показники конверсії та взаємодії з клієнтами. LLM також допомагають оцінити ефективність маркетингових кампаній. З їх допомогою маркетологи отримують відомості про думки та задоволеність клієнтів. Це дозволяє компаніям визначити найкращі стратегії для різних демографічних груп та покращити їх ефективність.

Пошукові системи. Великі мовні моделі застосовуються в пошукових системах для покращення якості результатів пошуку. Завдяки здатності краще розуміти складні запити та враховувати контекст, вони дозволяють системам надавати більш точну, змістовну та зрозумілу інформацію. Завдяки цьому, пошукові системи можуть зменшити кількість нерелевантних результатів і покращити користувацький досвід, надаючи більш змістовні та відповідні відповіді. Моделі також можуть розпізнавати синоніми і враховувати неявні наміри користувачів, роблячи пошук ще більш ефективним і природним.

Переклад. Мовні моделі значно покращили точність машинного

перекладу, завдяки здатності враховувати контекст, стилістику та культурні особливості тексту. Вони забезпечують точність перекладу на рівні людського, особливо для поширених мовних пар. Ці моделі використовуються для автоматичного перекладу документів, веб-сторінок та мультимедійного контенту, допомагаючи компаніям і організаціям розширювати свою аудиторію..

Творчість. За допомогою LLM можна генерувати творчі тексти, такі як оповідання, поезію та різноманітні історії. Можна запозичувати варіанти сюжетів, персонажів та образів, що в подальшому можуть стати вихідною точкою для творчого розвитку. Створення інтерактивних історій, де читачі взаємодіють із текстом та впливають на подальший розвиток сюжету. Це надає новий рівень залучення для читачів.

2 АНАЛІЗ ТЕХНОЛОГІЙ ТА ПІДХОДІВ ДЛЯ РЕАЛІЗАЦІЇ МОДЕЛІ

Моделі машинного навчання діляться на два основні типи: генеративні та дискримінативні. Генеративні моделі вивчають ймовірнісний розподіл даних та мають здатність генерувати нові зразки даних на основі заданого розподілу. Вони застосовуються для ряду задач де потрібно аналізувати структуру даних, виявляти приховані закономірності чи виконувати трансформації: кластеризація, оцінка щільності та зменшення розмірності. Дискримінативні моделі, в свою чергу, використовуються для передбачення виходу або класу на основі набору ознак. Даний тип моделей навчається оптимізовувати функцію різниці між передбаченим та істинним значенням. Прикладом таких моделей є дерева рішень, логістична регресія. Підтипом генеративних моделей є мовні моделі, які вивчають статистичні властивості природної мови та генерують текст. Зазвичай вони використовуються для задач класифікації текстів, машинного перекладу, та генерації. Моделі цього типу обчислюють ймовірність для послідовності слів і на основі цієї ймовірності можуть передбачити ймовірність наступного слова. Якщо послідовность високоякісна, то згенероване речення буде логічне та природне. Дані моделі навчаються на великих обсягах тексту, а потім використовують цей контекст для передбачення наступного слова на основі нового вводу.

2.1 Рекурентні нейронні мережі

Людська думка є безперервним процесом, який ґрунтується на попередньому досвіді та інформації. Під час читання тексту кожне наступне слово сприймається у контексті попередніх, формуючи цілісне розуміння. Людина не починає мислити "з нуля" після кожного слова – її думки мають тривалість і послідовність.

На відміну від цього, традиційні нейронні мережі не здатні враховувати контекст попередніх даних. Це суттєве обмеження для задач, де важливо зберігати пам’ять про послідовність подій. Наприклад, при класифікації сцен у фільмі модель повинна аналізувати не лише поточний момент, але й враховувати попередні події для кращого розуміння розвитку сюжету. Традиційні нейронні мережі не мають механізмів для такого довготривалого контекстуального аналізу.

Рекурентні нейронні мережі, рисунок 2.1, пропонують ефективне рішення цієї проблеми. Їхня унікальна структура з циклічними зв'язками дозволяє зберігати інформацію про попередні стани, що забезпечує врахування контексту під час наступних обчислень. Завдяки цьому РНМ здатні аналізувати послідовності даних, такі як текст, відео чи часові ряди, використовуючи знання про попередні події для прийняття рішень у поточний момент.

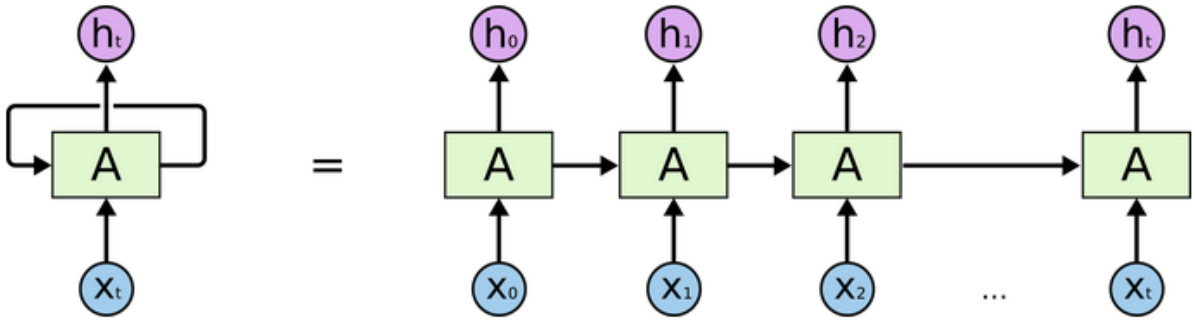

Рисунок 2.1 – Рекурентна мережа в розгорнутому вигляді

Такий розгорнутий вигляд мережі показує, що рекурентні нейронні мережі є релевантними в роботі з послідовними даними. Вони представляють собою природну архітектуру нейромереж для обробки таких типів даних. RNN складається з групи нейронів, що взаємодіють через зворотні зв’язки. Кожен нейрон приймає на вхід сигнал та інформацію про стан мережі з попереднього кроку, після чого передає свій вихідний сигнал до наступного нейрону. За

допомогою даних зв'язків RNN мають можливість зберігати інформацію про попередні стани мережі та враховувати її для обробки наступних даних.

RNN має один з найчастіших варіантів застосування, як лінгвістична модель генерації тексту . Мережа навчається прогнозувати наступне слово, беручи до уваги попередні слова. Тобто знання про текст в момент часу t складається із знання в попередній момент часу та з інформації про нове прочитане слово. Вони сумуються та виходить новий прихований стан H_t, формула 2.1. Через деяке домноження на матриці в стані H_t агрегуються всі слова, прочитані до цього. Проблема із затуханням градієнта, є однією з ключових труднощів, з якими стикаються при навчанні рекурентних нейронних мереж [5].

Особливістю рекурентних нейронних мереж є застосування одних і тих самих параметрів на всих рівнях в мережі. У мережах прямого поширення нейрони у різних шарах мають свій унікальний набір ваг, які визначають зв’язки з іншими нейронами. Натомість у рекурентних нейронних мережах один і той самий набір ваг використовується повторно на кожному кроці часу, що забезпечує збереження контексту для роботи з послідовними даними. Коригування ваг у РНМ відбувається шляхом орієнтованого на роботу з послідовностями алгоритму зворотного поширення помилки в часі.

$$H_t = \sigma\left(W_{xh}X_t + W_{hh}H_{t-1}\right), \tag{2.1}$$

де X_t – вхідний вектор слова;

W_{xh} – матриця вагів між вхідним та прихованим шаром;

H_{t-1} – вектор стану з попереднього кроку;

W_{hh} – матриця вагів для звязків між станами;

σ – функція активації softmax.

Далі йде обрахування вектору, формула 2.2, що відповідає певному слову, та порівняння за допомогою функції втрат з наступним правильним словом, щоб в подальшому максимізувати його ймовірність.

$$O_t = f_o\left(W_{ho} H_t\right), \tag{2.2}$$

де O_t – прогноз прихованого шару;

O_{ho} – матриця вагів між прихованим та вихідним шаром;

H_t – вихідний вектор;

На кожному етапі послідовності модель отримує на вхід правильне слово, рисунок 2.2, разом із прихованим станом, який зберігає інформацію про попередні слова. Використовуючи ці дані, мережа обчислює розподіл ймовірностей для наступного слова, щоб оцінити втрату для наступного елемента послідовності. Потім модель переходить до наступного слова, ігноруючи свій прогноз, і натомість подається правильне наступне слово разом із попередньою історією. Такий підхід, коли модель завжди отримує правильну послідовність для прогнозування наступного слова, називається навчанням з примусом учителя. Параметри мережі оптимізуються для мінімізації значення втрат крос – ентропії, формула 2.3, у всій навчальній послідовності за допомогою градієнтного спуску.

$$L = -\sum_{i=1}^{N} y_i \log(j_i), \tag{2.3}$$

де N – кількість класів;

y – істинне значення;

j – значення прогнозоване моделлю.

Таким чином, на кожен вихід мережі застосовується функція втрат, яка потребує, щоб мережа видавала значення в певний момент часу відповідно.

Вихід мережі на останніх станах, залежить від всіх попередніх параметрів та станів - їх виходів, тому що, щоб порахувати H_t, потрібно знати вихід H_{t-1} та входів відповідно. Тому при зворотньому розповсюдженні помилки, формула 2.4, функція втрат в конкретний момент часу дає інформацію, про те, що оновити в попередні моменти часу.

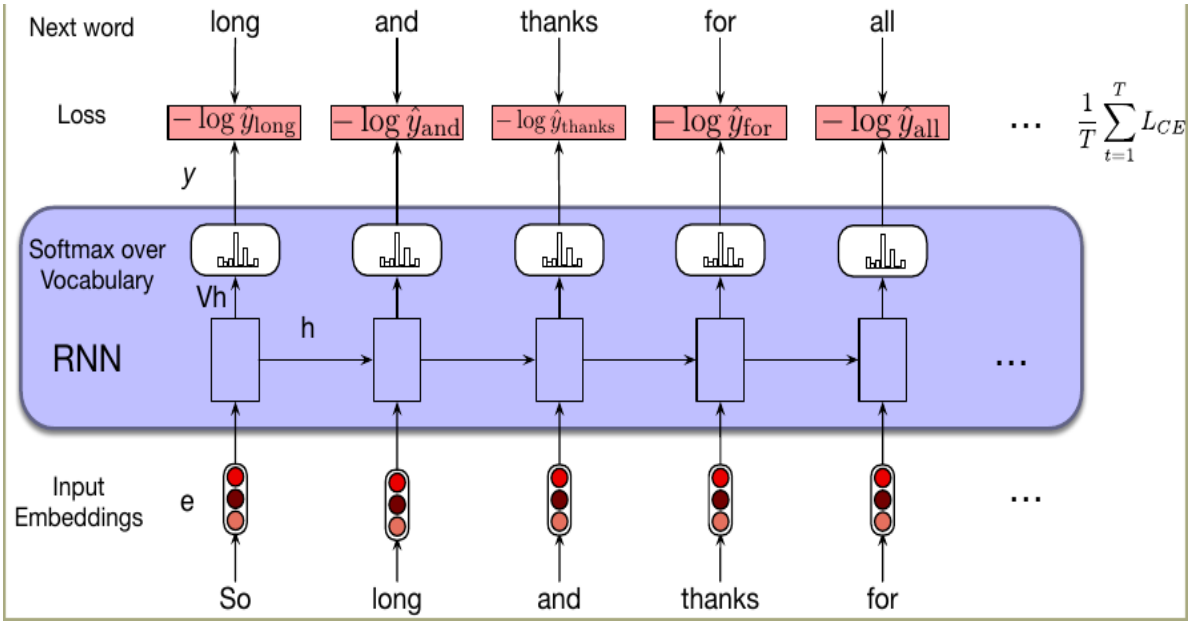

Рисунок 2.2 – Процес тренування RNN

Основним елементом цього процесу є градієнт, який використовується для обчислення похідних для матриць параметрів, формула 2.4.

$$\frac{\partial L}{\partial W} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial W}, \tag{2.4}$$

де W – матриця ваг;

L – загальне значення втрат для всієї послідовності даних;

L_i – значення втрат на окремому часовому кроці.

Ці параметри оновлюються з урахуванням коефіцієнта швидкості

навчання, за правилом [6] наведеним у формулі 2.5:

$$W = W - \alpha \frac{\partial L}{\partial W}, \tag{2.5}$$

де, α - швидкість навчання.

Для задач із довгими вхідними послідовностями, градієнт повинен пройти через велику кількість перетворень всі попередні блоки і через це з великим текстом градієнт ближче до початку сітки стане малим та почне поступово затухати, так як на етапі зворотного поширення помилки градієнти піддаються багаторазовому множенню, залежно від довжини послідовності. Це призводить до того, що ваги в мережі будуть оновлюватися дуже повільно або взагалі перестануть. Інформація про те, яким чином потрібно змінити ваги на початку, щоб помилка на кінцевих станах зменшилась практично не дійде до них. Даний чинник значно знижує ефективність навчання RNN має назву проблема зникаючого градієнта. В завданнях, де прогнозується наступне слово, на основі попередніх, де важливою є лише невелика частина попередньої інформації. Наприклад, в реченні "Хмари розташовані у небі", немає потреби у додатковому контексті – очевидно, що наступним словом буде "небо". У таких випадках, коли проміжок між релевантною інформацією та місцем, де вона потрібна, невеликий, даний тип мереж показує себе досить добре. Проте є випадки, коли потрібен більш глибокий контекст. Уявімо, що треба передбачити останнє слово у фразі "Він народився в Японії… Його улюблена страва - суші". Нещодавня інформація підказує, що наступним словом, ймовірно, буде назва страви, але для уточнення, яка саме страва, потрібен контекст про Японію, що може міститися дуже далеко в тексті.

Тож, цей підхід погано працює з дуже довгими текстами, так як крім проблеми затухання градієнта, в один, навіть досить великий вектор складно помістити велику кількість інформації.

2.2 Двонаправлені та багатошарові RNN

Рекурентні нейронні мережі є досить гнучкими. Завдяки поєднанню прямого поширення в розгорнутих обчислювальних графах із використанням векторів як вхідних і вихідних даних, такі мережі можуть слугувати модулями, які можна комбінувати різними способами. Рекурентні нейронні мережі стандартного типу використовують контекст лише з лівої сторони, тобто попередніх елементів послідовності для передбачень у конкретний момент часу. Але у ряді задач доступна вся послідовність даних, тож можливим є використання слова з контексту, що знаходиться праворуч. Для цього застосовують двонаправлені RNN - дві окремі RNN, одна з яких обробляє послідовність зліва направо, а інша — справа наліво, після чого їхні представлення об'єднуються у єдиний вектор.

У RNN, що працюють зліва направо, прихований стан у певний момент часу представляє все, що мережа знає про послідовність на цей момент. Цей стан є функцією від вхідних даних і відображає контекст мережі зліва від поточного моменту часу. Щоб врахувати контекст праворуч від поточного вводу, можна навчити RNN на оберненій послідовності. У цьому випадку прихований стан у момент часу буде представляти інформацію про послідовність від цього моменту до кінця послідовності. Двонаправлена RNN об'єднує дві незалежні RNN: одну, що обробляє вхід з початку до кінця, і другу, що обробляє вхід із кінця до початку, процес зображено на рисунку 2.3.

Генерація тексту часто вимагає врахування граматичних і семантичних зв’язків між словами, які можуть залежати від контексту як до, так і після поточного слова. Двонаправлені моделі допомагають уникнути втрати ключової інформації, особливо в довгих послідовностях, поєднуючи інформацію з обох боків. Дана проблема притаманна звичайним RNN через обмеження в обробці, лише в одному напрямку.

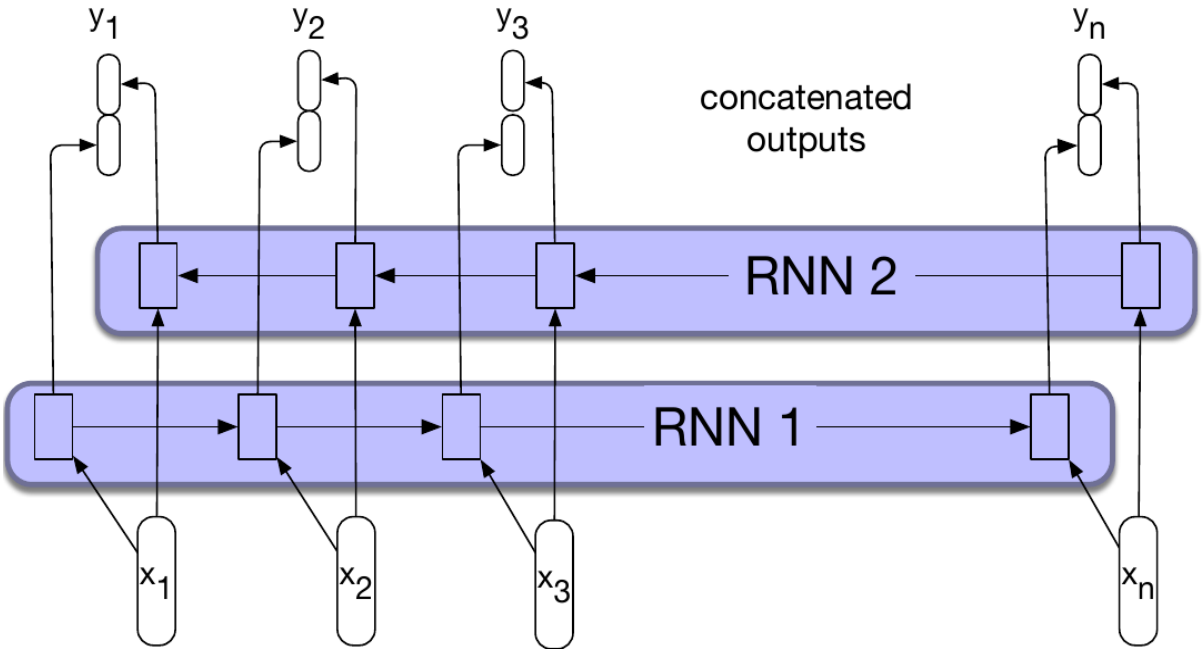

Рисунок 2.3 - Двонаправлена RNN

Представлення, обчислені цими мережами, конкатенуються у вектор, наведений у формулі 2.6, який враховує як лівий, так і правий контексти для кожного моменту часу:

$$H_t = H_{tf} + H_{tb}, \tag{2.6}$$

де H_{tf} - вихідні вектори стандартної RNN;

H_{tb} - вихідні вектори RNN із зворотнім проходом.

Двонаправлені RNN також ефективні для класифікації послідовностей. Для класифікації послідовностей звичайними рекурентними мережами використовується фінальний прихований стан RNN як вхід для наступного класифікатора. Проблема такого підходу полягає в тому, що фінальний стан зазвичай відображає більше інформації про кінець послідовності, ніж про її початок. Двонаправлені RNN вирішують цю проблему, поєднуючи фінальні приховані стани прямого та зворотного проходів і використовуючи це об'єднане

представлення для подальшої обробки .

У випадку генерації тексту в реальному часі, коли доступні лише попередні дані, наприклад, у чатах, двонаправлені RNN менш ефективні, оскільки не можуть використовувати майбутній контекст. Так як для кожного входу модель виконує дві незалежні серії обчислень, які потім поєднуються, обробка в двох напрямах подвоює кількость обчислень порівняно з односпрямованою моделлю. Також двонаправлена модель потребує більше пам’яті для збереження параметрів і більше часу на їх оновлення під час тренування.

Багатошарові RNN. У звичайних RNN вхідними даними є послідовності векторів, наприклад, ембеддинги слів або символів, а виходи використовуються для передбачення слів, міток чи інших характеристик послідовності. Проте немає жодних обмежень щодо використання всієї послідовності виходів одного шару як вхідної послідовності для наступного шару.

Багатошарова рекурентна нейромережа складається з кількох шарів, де вихідні дані попереднього шару слугують вхідними даними для наступного шару. Дана архітектура зазвичай за якістю перевершує одношарові моделі. Це пояснюється тим, що різні шари дозволяють формувати представлення на різних рівнях абстракції. Наприклад, на ранніх стадіях обробки зображення зорова система людини ідентифікує такі основні характеристики, як краї, які потім об’єднуються, щоб виявити більш складні структури. Подібним чином перший рівень багатошарової RNN створює прості представлення, які потім обробляються глибшими шарами для більш складного аналізу.

Кількість шарів у моделі залежить від завдання та характеристик набору даних. Однак, збільшення кількості шарів значно призводить до зростання обчислювальних витрат та споживання ресурсів для навчання багатошарових рекурентних нейромереж. Однак здатність краще розуміти та ефективно розпізнавати складні залежності виправдовує витрати на виконання багатьох завдань.

2.3 Мережі довготривалої короткострокової пам’яті

Мережі довготривалої короткострокової пам’яті (LSTM) — це тип рекурентних нейронних мереж, які можуть ефективно вивчати довготривалі залежності. Вперше вони були представлені Гохрейтером і Шмідгубером у 1997 році [7], а згодом вдосконалені та популяризовані багатьма іншими дослідниками. LSTM добре справляються з різноманітними завданнями і на даний час широко використовуються. Вони були створені для вирішення проблем довготривалої залежності та проблеми зникаючих градієнтів, які трапляються в стандартних RNN, де зберігання інформації протягом тривалого часу є складним завданням. Так як, хоч мережа має доступ до всієї попередньої послідовності, інформація, закодована в прихованих станах, зазвичай є локальною і більш релевантною для найближчих частин вхідної послідовності. Наприклад, присвоїти високу ймовірність слову "був", яке йде після "фільм", досить просто, адже слово ‘фільм’ створює локальний контекст для узгодження однини. Але правильно оцінити ймовірність слова "були" набагато складніше, тому що множина ‘сцени’ знаходиться далеко, а ближче розташоване слово "фільм" в однині, яке може заплутати модель. Ідеально, якщо мережа здатна зберігати віддалену інформацію про множину ‘сцени’ до моменту, коли вона стане потрібною, водночас обробляючи проміжну частину послідовності коректно.

Рекурентні нейронні мережі мають циклічну структуру з повторюваних ланцюгів елементів. У стандартній RNN цей повторюваний елемент має досить просту структуру, як правило, один шар із функцією активації. LSTM мають таку ж ланцюгову структуру, але сама архітектура рекурентного модуля відрізняється. Додається контекст та механізм воріт, рисунок 2.4.

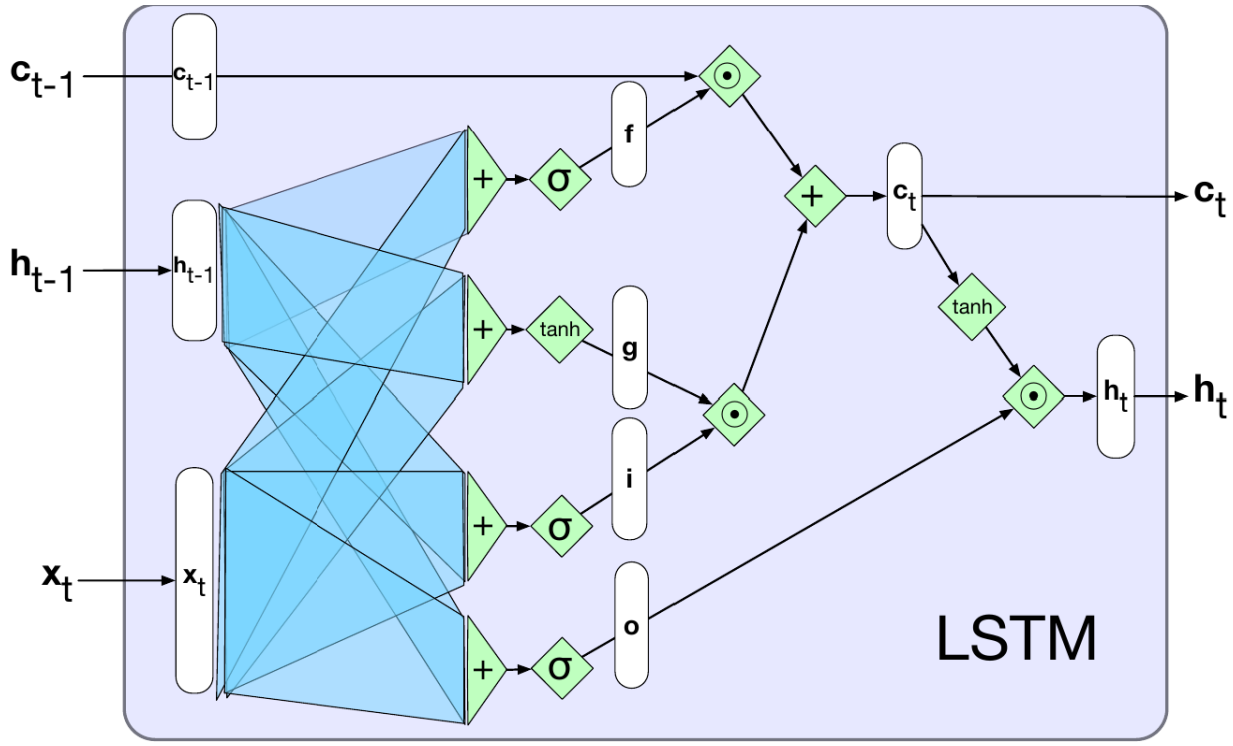

Рисунок 2.4 – Будова одиничного модуля LSTM

LSTM розділяє проблему управління контекстом на дві підзадачі: видалення зайвої інформації з контексту та додавання важливої інформації для майбутнього прийняття рішень. Основна ідея полягає в тому, щоб навчити модель керувати цим контекстом, а не жорстко прописувати якісь правила в архітектуру. Для цього LSTM спочатку додає до звичайного рекурентного шару спеціальний контекстний шар і використовує спеціальні нейронні вузли з механізмом воріт, для керування потоком інформації. Ворота в мережах цього типу реалізуються за допомогою додаткових ваг, що послідовно обробляють вхідні дані, попередній прихований та контекстний шари.

Усі ворота в LSTM працюють за однаковим принципом. Спочатку отримуються значення, що проходять через шар прямого поширення. Далі ці значення пропускаються через сигмоїдальну активаційну функцію, а потім покомпонентне множення з шаром, що регулюється. Сигмоїдальна функція використовується завдяки її здатності приводити вихідні значення до діапазону від 0 до 1. Таким чином цей процес схожий на застосування своєрідної бінарної

маски: якщо значення близьке до 1, відповідна інформація зберігається майже без змін, а якщо значення близьке до 0, ця інформація майже повністю видаляється.

Ворота забування (forget gate), відповідають за те, щоб прибирати з контексту інформацію, що більше не потрібна, принцип функціонування наведений в формулі 2.7. Для цього береться зважена сума прихованого шару з попереднього стану та поточного входу, потім вона пропускається через сигмоїдальну функцію активації. В результаті формується своєрідна маска, що визначає, які частини інформації потрібно залишити, а які - видалити чи зменшити рівень важливості відповідно до ступеня нерелевантності інформації. Потім дана маска покомпонентно множиться на контекстний вектор. Такий процес, дозволяє обнулити зайву інформацію, залишаючи лише те, що може знадобитися далі [7].

Наприклад, якщо модель зберігає інформацію про час дієслова у попередньому контексті, ворота забуття допомагають позбутися цієї інформації, коли з'являється нова частина речення, що потребує іншого часу. Таким чином, модель не використовуватиме дієслова в неправильному часі, і зможе правильно вибрати форму дієслова для нової ситуації.

$$\begin{aligned} f_t &= \sigma(U_f h_{t-1} + W_f x_t) \\ k_t &= c_{t-1} \odot f_t \end{aligned}, \tag{2.7}$$

де U_f – ваги, для прихованого стану на попередньому кроці;

h_{t-1} – прихований стан на попередньому кроці часу;

W_f – ваги, для поточного входу;

x_t – поточний вхід;

σ – сигмоїдна активаційна функція;

c_{t-1} – контекстний вектор на попередньому кроці часу.

Наступним кроком є обчислення інформації, яку потрібно отримати з попереднього прихованого стану та поточного входу, формула 2.8. Для цього застосовується функція гіперболічного тангенса, утворюється вектор нових можливих значень. Це стандартна дія, яка виконується загалом у всіх рекурентних мережах для отримання нової інформації.

$$g_t = \tanh(U_g h_{t-1} + W_f x_t), \tag{2.8}$$

де U_g - ваги, для прихованого стану на попередньому кроці;

h_{t-1} - прихований стан на попередньому кроці часу;

W_f - ваги, для поточного входу;

x_t - поточний вхід;

tanh - функція активації.

Наступним кроком йде створення маски input gate layer для add gate(воріт додавання), щоб вибрати інформацію, яку слід додати до поточного контексту, формула 2.9. Сигмоїдний шар, визначає, яким чином значення будуть проходити процедуру оновлення.

$$i_t = \sigma(U_i h_{t-1} + W_i x_t), \tag{2.9}$$

де U_i – ваги, для прихованого стану на попередньому кроці;

h_{t-1} – прихований стан на попередньому кроці часу;

W_i – ваги, для поточного входу;

x_t – поточний вхід;

σ – сигмоїдна активаційна функція.

Далі йде застосування маски i_t до значень g_t, залишаючи лише ті частини, які мають вагоме значення для оновлення контексту. Результат є новою інформацією, що буде додана до поточного стану комірки, формула 2.10.

Потім ця інформація додається до модифікованого контекстного вектора, створюючи новий контекстний вектор.

У прикладі з мовною моделлю, наведеному вище, потрібно до стану комірки, якщо з'являється нове дієслово, додати інформацію про його час для заміни старої інформації про нього.

$$\begin{aligned} j_t &= g_t \odot i_t \\ c_t &= j_t + k_t \end{aligned}, \tag{2.10}$$

де g_t – інформація з попереднього прихованого стану та поточного входу;

i_t – маска, створена add gate;

j_t – нова інформація, що буде додана до поточного стану комірки.

Ворота виходу відповідають за те, яка інформація потрібна для формування поточного прихованого стану використовуючи оновлений контекст (попередній прихований стан та минулий контекст). Цей тип воріт формує маску, щоб вирішити, що має бути використано з оновленого контексту. Контекст проходить через гіперболічний тангенс, щоб обмежити значення в діапазоні від -1 до 1. Ця трансформована інформація множиться покомпонентно на маску, що дозволяє отримати поточний прихований стан .

$$\begin{aligned} o_t &= \sigma(U_o h_{t-1} + W_o x_t) \\ h_t &= o_t \odot \tanh(c_t) \end{aligned}, \tag{2.11}$$

де U_o – ваги, для прихованого стану на попередньому кроці;

h_{t-1} – прихований стан на попередньому кроці часу;

W_o – ваги, для поточного входу;

x_t – поточний вхід;

o_t – маска воріт виходу;

c_t – оновлений контекстний вектор.

На виході LSTM формує новий прихований стан і оновлений контекст, що використовуються на наступних кроках або як остаточний результат моделі. За рахунок додавання контексту та чотирьох воріт LSTM краще вивчає довгі закономірності, протягує інформацію з початку тексту в кінець. При зворотньому розповсюдженні помилки, за рахунок тотожних зв'язків градієнт може протікати відразу в попередній стан без особливих змін, що допомагає йому краще зберігатись, менше затухати по мірі руху по шарах.

Таким чином, LSTM дозволяє моделі ефективно зберігати далеку інформацію, необхідну для ухвалення складних рішень, долаючи обмеження звичайних RNN. Дані мережі значно покращили здатності RNN у роботі з довготривалими залежностями і демонструють відмінні результати у багатьох задачах. Попри те, що їх математична частина значно складніша за базові мережі, ця складність інкапсульована всередині обчислювальних блоків, що дозволяє зберігати модульність і легко експериментувати з різними архітектурами. Єдина додаткова зовнішня складність у порівнянні з базовим рекурентним блоком - це додавання контекстного вектора як вхідного і вихідного сигналу. Модульність блоків LSTM є ключовою причиною їхньої потужності та широкого застосування. Вони легко інтегруються в різні архітектури нейронних мереж. Як і звичайні RNN, багатошарові мережі з такими блоками можна перетворити на глибокі мережі прямого поширення і навчати їх за допомогою стандартного методу зворотного поширення помилки. Тому на практиці LSTM стали стандартним вибором для сучасних систем, які працюють із рекурентними мережами, замінивши звичайні RNN. Наступним кроком у дослідженні RNN є використання механізмів уваги, які обіцяють ще більше покращити результати.

Основна ідея уваги полягає в тому, що замість створення єдиного прихованого стану для вхідної послідовності, енкодер генерує прихований стан на кожному кроці, до якого звертається декодер. Проте використання всіх

станів одночасно створило б величезний вхід для декодера, тому потрібен механізм для пріоритизації, які стани використовувати. Увага дозволяє декодеру призначати різну кількість ваги або "уваги" кожному з станів енкодера на кожному кроці декодування.

Зосереджуючись на тому, які вхідні токени є найбільш релевантними на кожному кроці, моделі, засновані на увазі, можуть навчатися нетривіальним вирівнюванням між словами в згенерованому тексті та словами у вихідній послідовності. Декодер може правильно визначати релевантність слів у контексті всього тексту.

2.4 Архітектура трансформера

Хоча увага і дозволила рекурентним моделям створювати набагато кращі результати у генерації тексту, все ж залишалася суттєва проблема при використанні рекурентних моделей для енкодера і декодера: обчислення були за своєю природою послідовними і не могли бути паралелізовані по всій вхідній послідовності. З появою трансформера було представлено нову парадигму моделювання: відмова від рекурентності взагалі і покладання виключно на особливу форму уваги, яка називається самоувага [8].

Трансформер — це модель глибинного навчання, яка використовує особливий механізм самоуваги для обробки вхідних даних, зокрема послідовностей тексту. Завдяки цьому механізму, трансформери здатні навчатися взаємозв'язкам між токенами в послідовності без необхідності обробляти дані послідовно, як це роблять рекурентні нейронні мережі (RNN). Що в свою чергу дає змогу значно прискорити процес навчання та обробки даних. Трансформер базується на архітектурі енкодера-декодера, яка широко використовується для завдань, таких як машинний переклад, де послідовність слів перекладається з однієї мови на іншу. Ця архітектура складається з двох компонентів. енкодера, що конвертує вхідну послідовність токенів у

послідовність векторів вбудовування, які часто називають прихованим станом або контекстом, та декодера, що використовує прихований стан енкодера для ітеративного генерування вихідної послідовності токенів, по одному токену за раз.

Архітектура трансформера була спочатку розроблена для завдань обробки послідовностей, проте згодом її енкодер та декодер почали застосовуватись як окремі моделі. Існує велика кількість варіацій трансформерів, однак більшість із них належать до трьох основних типів.

Тільки енкодер перетворює вхідну текстову послідовність на числове представлення, яке ефективно використовується для завдань, таких як класифікація текстів або розпізнавання іменованих сутностей. Моделі, як BERT та його різні варіанти, належать до цієї архітектури. У цьому випадку представлення для кожного токена враховує як лівий контекст (до токена), так і правий (після токена). Такий підхід називається називається двосторонньою увагою.

Тільки декодер, отримавши початкову підказку, наприклад, "Сьогодні я планую...", завершить послідовність, по черзі прогнозуючи найбільш ймовірне наступне слово. Моделі, як GPT, належать до цієї категорії. Представлення для кожного токена в цій архітектурі обчислюється лише на основі попереднього контексту. Такий підхід називається каузальною або авторегресивною увагою.

Архітектура енкодер-декодер використовується для моделювання складних перетворень однієї текстової послідовності в іншу, що робить її ідеальною для завдань, як-от машинний переклад або реферування. Окрім трансформерної архітектури, яка поєднує енкодер і декодер, рисунок 2.5, до цієї групи також входять моделі, такі як BART і T5.

На практиці межа між застосуваннями архітектур тільки енкодера і тільки декодера часто є умовною. Наприклад, моделі тільки декодера, такі як GPT, можна налаштувати для виконання завдань, які традиційно відносяться до обробки послідовностей, як-от переклад. Так само, моделі тільки енкодера,

наприклад BERT, можуть бути використані для завдань реферування, які зазвичай асоціюються з архітектурами енкодер-декодер або тільки декодер.

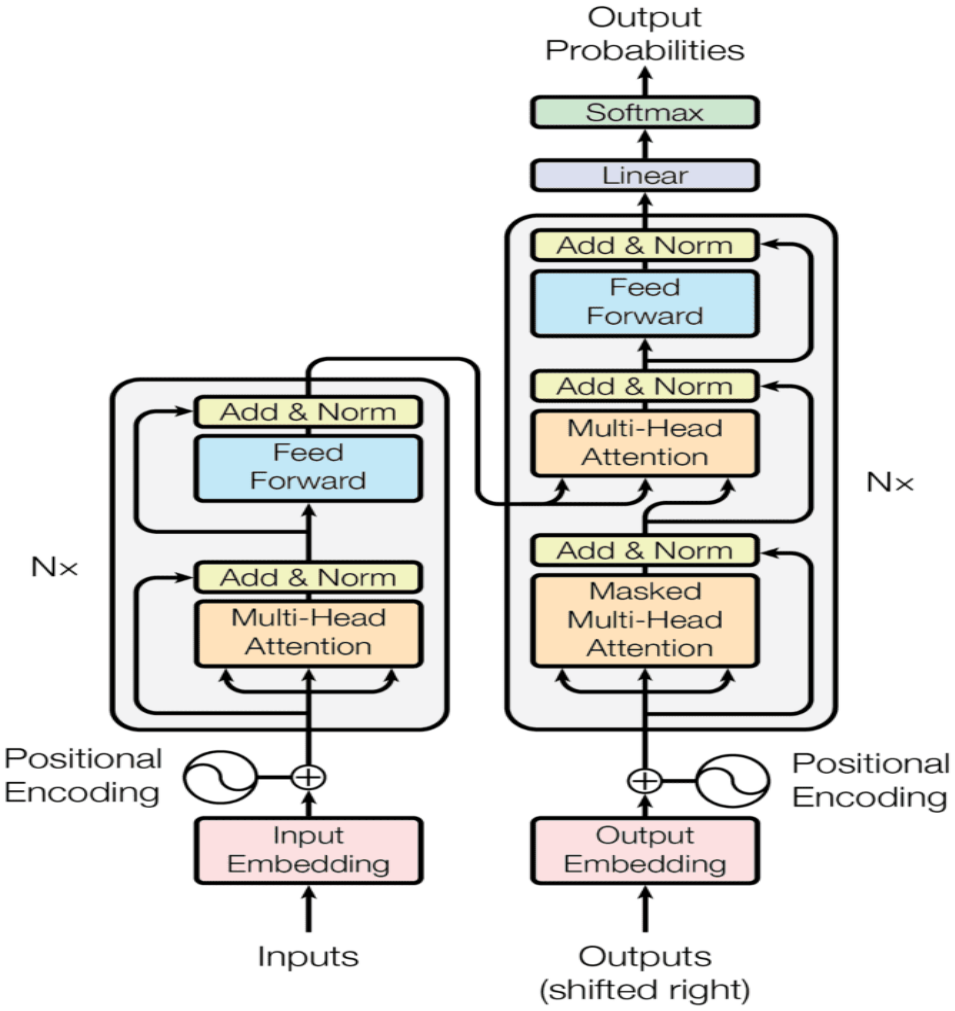

Рисунок 2.5 – Архітектура трансформера

Блок енкодера в трансформері складається з багатьох шарів енкодерів, розташованих один над одним. Кожен шар отримує послідовність вбудовувань і пропускає їх через шар самоуваги з багатьма головами (multi-head attention) та повнозв’язаний шар прямого поширення (feed-forward).

Вихідні вбудовування кожного шару енкодера мають той самий розмір, що й вхідні. Головне завдання стека енкодерів полягає в оновленні вхідних вбудовувань, створюючи репрезентації, які втілюють контекстуальну

інформацію послідовності. Наприклад, слово "python" перетворюється, набуваючи значення, ближче до програмування та віддаляючись від змії, якщо воно з’являється поруч із такими словами, як "coding" чи "language". Кожен із цих підшарів також використовує пропускні з'єднання та нормалізацію шару(layer normalization), що є стандартними прийомами для ефективного навчання глибоких нейронних мереж.

Ідея самоуваги полягає в тому, щоб дозволити увазі працювати на всіх станах в одному шарі нейронної мережі. У терміні "self" у самоувазі підкреслює, що ваги розраховуються для всіх прихованих станів, які належать до одного блоку, наприклад, станів енкодера. У протилежність цьому, увага в рекурентних моделях базується на визначенні значущості кожного прихованого стану енкодера відносно прихованого стану декодера на конкретному етапі роботи. Тобто, архітектура із самоувагою здатна навчитись набагато швидше, ніж рекурентні моделі.

Задаючи послідовність ембедінгів токенів, механізм самоуваги генерує нову послідовність ембедінгів, формула 2.12.

$$\sum_{y=1}^{n} W_{ji} x_j \,, \tag{2.12}$$

де W_{ji} - ваги уваги;

x_i – векторні представлення токенів, кожен x_i представляється лінійною комбінацією всіх x_i

Ваги уваги нормалізуються таким чином, що їх сума по осі j дорівнюватиме одиниці. Приклад усереднення: побачивши слово "bank", можна подумати про фінансову установу, але якщо додати більше контексту, як-от "he sat on the bank of the river", стає зрозуміло, що "bank" в цьому випадку означає берег річки. Подібним чином можна створити представлення для "bank", яке враховує цей контекст, комбінуючи всі токен-ембедінги в різних пропорціях,

призначаючи більшу вагу W_{ji} токен-ембедінгам для "sat" і "river". Ембедінги, створені таким чином, називаються контекстуалізованими ембедінгами.

Існує декілька способів реалізації шару самоуваги, але найпоширенішим є масштабована скалярна увага, представлена на рисунку 2.4.

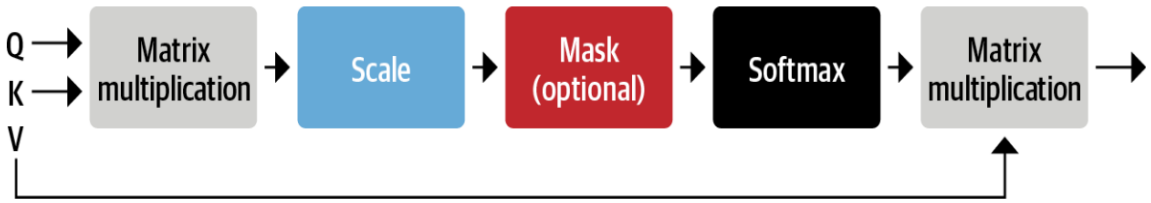

Рисунок 2.6 – Операції масштабованої скалярної уваги

Основні етапи реалізації цього механізму включають проєктування кожного токен-ембедінга у три вектори: запит, ключ і значення. Потім обчислюються оцінки уваги, де визначається, наскільки вектор запиту співвідноситься з вектором ключа за допомогою функції подібності. Для масштабованої скалярної уваги ця функція є скалярним добутком, який обчислюється через матричне множення ембедінгів. Запити та ключі з високою подібністю мають більший скалярний добуток, а з меншою — менший. Результатом є матриця оцінок уваги розміром $n \times n$ для послідовності з n токенів. [8].

Обчислення ваг уваги. що скалярні добутки можуть мати значно великі значення, що негативно впливає на стабільність навчання. Для того, щоб зменшити це негативне вплив, оцінки уваги спочатку множаться на масштабувальний коефіцієнт, щоб зменшити їхню дисперсію і зробити їх більш керованими. Після цього оцінки нормалізуються через застосування функції софтмакс, яка перетворює їх таким чином, щоб сума значень в кожному стовпці дорівнювала одиниці. Завдяки цьому ми отримуємо матрицю розміром $n \times n$, що містить ваги уваги W_{ji}, які відображають взаємозв'язок між кожними двома

токенами. Останнім етапом є множення отриманих ваг уваги на відповідні вектори значень, процес наведено в формулі 2.13, дає оновлені токени ембедінги для подальшої обробки. Цей процес дозволяє моделі враховувати контекст кожного слова для отримання більш точних і контекстуально релевантних представлень.

$$\sum_{j=1}^{n} W_{ji} v_j , \tag{2.13}$$

де, W_{ji} - ваги уваги;

v_j – вектор значень.

У механізмі самоуваги застосовуються три окремі лінійні перетворення для кожного вбудовування, щоб створити вектори запиту, ключа та значення. Кожне з цих перетворень проєктує вбудовування, і для кожного з них існує набір параметрів, які оптимізуються під час навчання. Це дає змогу шару самоуваги акцентувати увагу на різних семантичних характеристиках послідовності.

Також виявляється корисним мати кілька наборів лінійних проекцій, кожен з яких представляє окрему голову уваги. Це важливо, оскільки одна голова softmax зазвичай фокусується на одному аспекті схожості. Множинність голів дозволяє моделі одночасно акцентувати увагу на різних аспектах. Наприклад, одна голова може аналізувати зв'язок між підметом і дієсловом, а інша – звертати увагу на сусідні прикметники.

У енкодері та декодері шар прямого поширення представляє собою двошаровий повнозв'язний нейронний шар. Однак є одна особливість: замість того, щоб обробляти всю послідовність вбудовувань як єдиний вектор, кожне вбудовування обробляється окремо. Через це такий шар часто називають позиційно-залежним. Зазвичай розмір прихованого шару в чотири рази перевищує розмір вбудовувань, а для активації часто застосовується функція

GELU. Саме в цьому шарі відбувається більша частина запам'ятовування. Основна різниця між декодером і енкодером полягає в тому, що декодер включає два види уваги.

Маскований багатоголовий шар самоуваги забезпечує, щоб кожен новий токен, що генерується на певному кроці, базувався лише на попередніх результатах і поточному токені, що прогнозується. Без цього механізму декодер міг би просто копіювати цільовий текст під час навчання, що призвело б до неправильних або неадекватних результатів. Маскування вхідних даних робить задачу більш складною та не дає можливості для тривіальних рішень. Шар уваги енкодера-декодера виконує багатоголову увагу, обробляючи ключі та значення, отримані від виходу енкодера, з використанням поточних проміжних представлень декодера як запитів [8]. Це дає можливість шару уваги вивчати взаємозв'язки між токенами з двох окремих послідовностей. У кожному блоці декодер отримує доступ до ключів та значень, що генеруються енкодером.

3 ОСОБЛИВОСТІ НАВЧАННЯ ТА ГЕНЕРАЦІЇ В LLM

3.1 Тонке налаштування моделей

Великі мовні моделі під час попереднього навчання обробляють великі корпуси текстів на різноманітні тематики, але часто з’являється потреба застосувати модель в певному домені для нового, більш спеціалізованого завдання. LLM, що тренуються на широкому спектрі даних показують гарну продуктивність у загальних задачах. Проте їх продуктивність знижується в нішевих доменах, де потрібні специфічні знання. Донавчання моделі на вузькоспеціалізованих даних дає можливість адаптувати модель до роботи із специфічною термінологією, стилістикою та певним форматом даних, що характерні для конкретної галузі [9]. Разом з цим донавчання сприяє підвищенню точності та деталізації відповідей для специфічних задач. До прикладу, потрібна мовна модель, що спеціалізується на медичних чи юридичних текстах. У таких випадках модель продовжує тренування на відповідних даних в новій доменній області. Даний процес, у якому береться вже попередньо навчена модель і виконуються додаткові кроки навчання на нових даних, називається тонким налаштуванням або fine-tuning. Усі види тонкого налаштування спрямовані на подальшу адаптацію попередньо навченої моделі до нових даних, але різниця між ними полягає в тому, які саме та яка кількість параметрів моделі змінюються під час даного процесу.

Продовжене попереднє навчання – найпростіший тип тонкого налаштування, він полягає в оновленні параметрів всієї моделі на нових доменних даних, але використовуючи ті ж самі підходи, що й під час її початкового навчання: прогнозування слів і таку ж функцію втрат на основі перехресної ентропії. Через те, що нові дані, буквально, додаються до кінця початкового набору даних, цей метод має відповідну назву [10].

Оновлення всіх параметрів це досить затратне та повільне дійство. Тому існує оптимізаційний підхід, суть якого полягає в тому, що частина параметрів залишається незмінною, їх заморожують, а тренуванню на нових даних підлягає лише обраний деяким способом набір параметрів. Такий підхід відомий як параметрично ефективне тонке налаштування. Існують ідейно різні параметрично ефективні методи тонкого налаштування, в залежності від конкретних завдань. Наприклад, потрібно адаптувати мовну модель, як класифікатор, для цього до верхнього шару моделі додають новий компонент – так звану класифікаційну голову. Ця додаткова структура отримує на вхід вектор вбудовування, продукований верхнім шаром трансформера, та виконує класифікацію. При цьому, в даному методі всі ваги попередньо навченої моделі заморожуються, а тренується тільки додана голова для класифікації на нових розмічених відповідно до класу даних. Зазвичай усе, що відбувається після попереднього навчання, об’єднують під загальним терміном «посттренування»;

3.2 Низькорангова адаптація

Адаптація великих мовних моделей є вельми проблематичною, особливо коли мова йде про оновлення величезної кількості параметрів. Кожен прохід градієнтного спуску повинен враховувати численні великі шари моделі, що робить процес дуже вимогливим до обчислювальних ресурсів, обсягу пам’яті та часу. Задля подолання цих труднощів були створені більш ресурсоощадні методи, що дають змогу адаптувати модель без необхідності коригувати всі її параметри. Ці підходи називаються ефективними з точки зору параметрів, оскільки вони зосереджуються лише на оновленні певних ключових параметрів. Наприклад, деякі параметри залишаються незмінними, тоді як модифікація торкається лише тих, що мають найбільше значення для адаптації до нових даних.

Один з таких найбільш застосовуваніших методів є LoRA (Low-Rank

Adaptation). Трансформери містять багато шарів, в яких виконуються операції множення матриць, наприклад, шари для обчислення запиту, ключа, значення, виходу у механізмі уваги. Основна ідея LoRA, рисунок 3.1, полягає в тому, щоб замість оновлення всіх цих шарів під час тонкого налаштування, механізм LoRA заморожує їх, а натомість оновлює їхню низькорангову апроксимацію, невеликі додаткові матриці, що мають в рази менше параметрів.

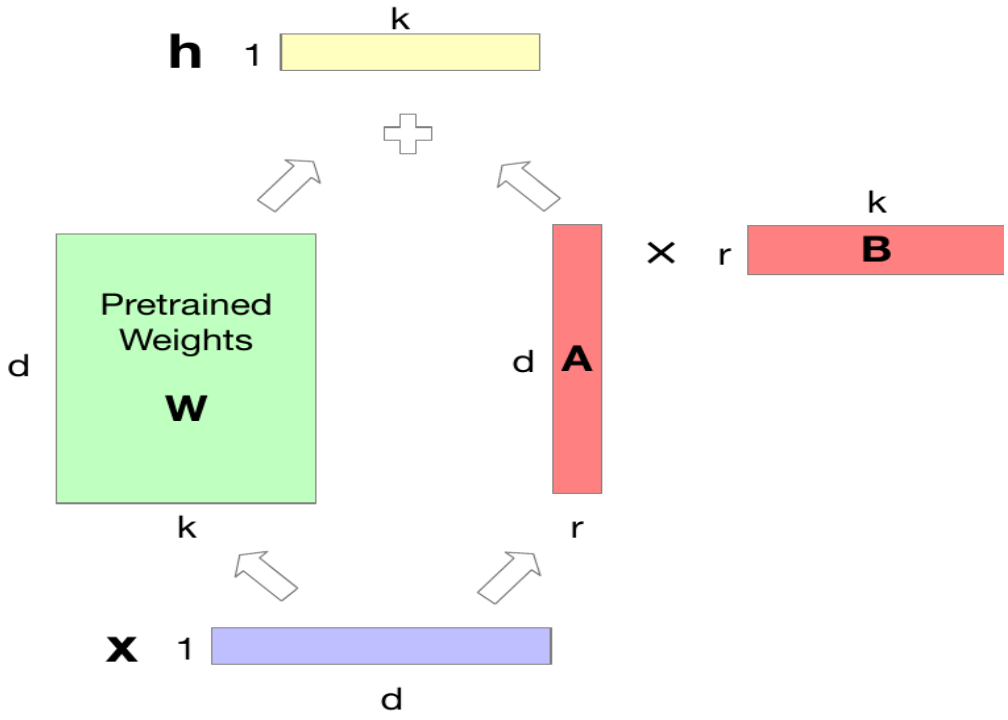

Рисунок 3.1 – Функціонування низькорангової адаптації

Під час файнтюнінгу зазвичай оновлюють матрицю вагів W розміром $N \times d$, через градієнтний спуск. У стандартному випадку дана матриця має оновлення ΔW з тим самим розміром $N \times d$. Це сприяє зміні всіх попередньо навчених параметрів після кожного кроку градієнтного спуску. У методі низькорангової адаптації матриця W залишається незмінною, замість неї оновлюється її низькорангова декомпозиція. Цей підхід передбачає створення двох матриць A та B, де A має розмір $N \times r$, а B розміром $r \times d$. Значення r – гіперпараметру, що визначає ранг апроксимації повинно бути меншим d та N.

Фактично даний спосіб базується на припущенні, що зміни до W мають бути ефективно представлені в просторі меншої розмірності. Матриці B та A фактично визначають проекцію даних у зменшений підпростір та відповідають за подальше відновлення цих даних у простір початкової матриці W [11].

В процесі роботи методу оновлюються лише параметри матриць A і B. Таким чином, замість оновлення W шляхом додавання ΔW, використовується низькорангова апроксимація у вигляді $W \oplus BA$. У новому підході прямий прохід, який зазвичай виглядає як xW, замінюється на формулу 3.1

$$h = xW \oplus xAB, \tag{3.1}$$

де W –заморожена матриця основних ваг,

B, A– адаптаційні матриці, які навчаються.

На цьому етапі BA додається до W динамічно для обчислення результатів. Розрахунок функції втрат включає вплив як замороженої матриці W, так і адаптаційних B та A. Однак градієнти обчислюються тільки для параметрів B та A, адже вони є єдиними змінними, які модель оптимізує під час тренування. Після завершення навчання виконується процес, відомий як злиття. На цьому етапі адаптаційна матриця BA додається до основної матриці ваг W. Результат цього додавання зберігається в W, і більше немає потреби зберігати або використовувати окремо B та A. Після цього модель використовує вже оновлену W під час прогнозування, і всі обчислення стають такими ж, як у стандартній моделі без LoRA, але з урахуванням адаптаційних змін, внесених під час тренування.

Таким чином, LoRA має низку переваг. Вона значно зменшує вимоги до апаратних ресурсів, тому що градієнти потрібно обраховувати лише для невеличкої частини параметрів. Оновлення ваг легко додаються до попередньо навчених, адже BA співпадає за розмірністю з W. Це забезпечує відсутність додаткових витрат часу під час інференсу моделі. LoRA дозволяє створювати

спеціалізовані модулі для різних завдань, які можна легко підключати чи відключати від W . Існує багато варіантів LoRA, але у своїй стандартній версії вона застосовується лише до шарів в обчисленні уваги.

3.3 Токенізація в великих мовних моделях

Токенізація — це процес перетворення тексту на послідовність елементів, які можуть бути словами або їх частинами. Токени є найменшими одиницями тексту, що мають значення і можуть бути оброблені мовною моделлю. Наприклад, речення "Як справи ? " перетворюється на токени ["Як", "справи", "?"]. Під час навчання та прогнозування LLM кожному токену присвоюється числовий ідентифікатор, що відповідає певному рядку у таблиці векторних представлень токенів, які використовуються моделлю для подальшої обробки.

Кодування пар байтів найбільш широко використовуваний спосіб токенізації у великих мовних моделях. На відміну від способів, де токени визначаються як окремі слова, розділені пробілами, або за допомогою більш складніших алгоритмів розбиття тексту, BPE пропонує інший підхід. Замість того, щоб вручну задавати правила токенізації, даний метод використовує сам текст, для автоматичного визначення, які семантичні одиниці найкраще підходять для представлення токенів – визначає які будуть словами, частинами слів чи символами.

Зазвичай алгоритми обробки тексту вивчають певні мовні закономірності на основі навчального корпусу даних, де вони аналізують структуру, частотність слів та інші характеристики мови. Потім ці здобуті знання використовуються для обробки нового тексту, тестових чи даних з реального світу. Поява в нових даних слова, яке не було в навчальних даних може привести до некоректної обробки та хибних результатів відповідно.

Якщо навчальний набір містить слова на кшталт "strong", "stronger" і "weak", але не включає форму "weaker", модель може не змогти коректно

обробити це слово, оскільки воно не зустрічалося під час навчання. Для вирішення цієї проблеми сучасні токенізатори автоматично розбивають слова на менші одиниці — субслова. Субслова, зазвичай, представляють собою довільні підрядки або деякі смислові одиниці, такі як морфеми — найменші одиниці мови, які мають значення. Наприклад, слово "predefined" складається з трьох морфем: "pre" (означає "до"), "define" (дієслово) і "ed" (позначає минулий час). У сучасних методах токенізації переважна більшість токенів представляє собою повні слова, але також деякі токени є окремими частинами слів, що мають високу частотність, наприклад, закінчення "ed". Тому, якщо зустрічається невідоме слово, наприклад "unhappier", токенізатор може розділити його на послідовність відомих субсловних одиниць, таких як "un" і "happier". Також, коли це необхідно, токенізатор може навіть розбити слово на окремі літери, що дозволяє обробляти навіть ті слова, які раніше не зустрічалися в навчальних даних.

Більша частина методів токенізації складається з двох основних етапів: формування токенів та їх сегментація. На першому етапі алгоритм формування аналізує необроблений текст із навчального корпусу (який інколи вже поділений на слова, наприклад, за пробілами) і створює словник токенів — набір елементів, які будуть використовуватися для сегментації тексту. На другому етапі, алгоритм сегментації токенів бере необроблене речення з тестового набору та розділяє його на токени, використовуючи раніше створений словник.

Алгоритм кодування пар байтів, який продемонстровано в таблиці 3.1, працює за наступним принципом: спочатку створюється словник, який містить лише окремі символи. Потім алгоритм аналізує текст із навчального корпусу, шукає дві літери, які найчастіше зустрічаються разом, наприклад, "A" і "B", і додає їх поєднання "AB" до словника. У текстових даних ці пари літер замінюються новим символом "AB". Даний процес повторюється багаторазово, поступово формуючи складніші та довші комбінації символів. Кількість таких

злиттів визначається параметром k, який вказує, скільки нових символів потрібно створити. Після завершення роботи алгоритму виходить словник, що складається зі стартових символів та k нових комбінацій. BPE функціонує зазвичай в межах кожного окремого слова, не об’єднуючи символи з різних слів. Тому на початковому етапі вхідний текст розбивається на окремі слова через пробіли, утворюючи список рядків, де кожен рядок складається із символів одного слова та спеціального маркера, що позначає його кінець.

Таблиця 3.1 – Приклад роботи алгоритму Byte pair encoding

Крок	Злиття	Частота	Словник	Корпус
0	-	-	, a, e, g, h, i, l, m, n, o, r, s, t, v, w	m o o n, m o o n l i g h t, r i v e r…
1	r i	8	, a, e, g, h, i, l, m, n, o, r, s, t, v, w, ri	m o o n, m o o n l i g h t, **ri** v e r, s t r o n g e r, **ri** v a l
2	o o	7	, a, e, g, h, i, l, m, n, o, r, s, t, v, w, ri, oo	m **oo** n, m **oo** n l i g h t, ri v e r, s t r o n g e r, ri v a l
3	ri v	8	, a, e, g, h, i, l, m, n, o, r, s, t, v, w, ri, oo, riv	m **oo** n, m **oo** n l i g h t, **riv** e r, s t r o n g e r, **riv** a l
n	rive r	6	, a, e, g, h, i, l, m, n, o, r, s, t, v, w, ri, oo, riv, oon, rive, river	moon, moon l i g h t, river, s t r o n g er, rive a l

Навчальний датасет складається із 18 токенів слів, кожне зі своїми частотами. Слово "moon" зустрічається 5 разів, "moonlight" - 2, "river" - 6, "stronger" - 3, "rival" - 2. Початковий словник містить 12 літер. Алгоритм BPE спочатку обчислює частоту з якою зустрічаються всі сусідні пар символів у

корпусі. Найпоширенішою парою символів є "r i", оскільки вона зустрічається у словах "river" (6 разів) та "rival" (2 рази), що в сумі становить 8 входжень. Після цього символи об’єднуються, утворюючи новий символ "er", і обчислення виконується знову. Система "вивчає", що для закінчення слова "er" повинен бути окремий токен. Коли словник вже сформовано, алгоритм використовує сегментатор токенів для обробки тестових даних. Він працює на основі об’єднань, які були "вивчені" під час тренування, алгоритм діє жадібно, тобто застосовує правила в тому порядку, в якому вони були вивчені.

Спочатку кожне слово в текстовому корпусі розбивається на окремі символи. На наступному кроці перше правило заміняє всі входження "r i v e r" на "r i v er", друге правило — заміняє "r i v er" на "riv er", і так далі. Як наслідок, якщо тестові дані містять рядок "r i v e r", він буде токенізований як повне слово. Однак нове або невідоме слово, наприклад, "s t r o n g e r", буде розбито вже на два токени: "s t r o n g" та "er".

В задачах реального світу BPE застосовується для великих корпусів даних, виконуються тисячі об'єднань пар символів. Як результат, більшість слів будуть представлені як окремі токени, і лише рідкісні чи невідомі слова будуть розбиті на менші частини.

3.4 Вибір наступного слова під час генерації в LLM

Ключовим аспектом механізму генерації для великих мовних моделей є вибір наступного слова, який залежить від ймовірностей, які модель прогнозує наступним можливим токенам [12]. Процес визначення слова відповідно до його ймовірності виданою моделлю, називається декодуванням. Коли декодування здійснюється поступово, зліва направо та кожне нове слово генерується на основі попередніх, тоді такий підхід вважають авторегресивною генерацією або генерацією в причинно-наслідковій мовній моделі. Насправді, авторегресивна модель передбачає значення на етапі t, на основі лінійної

залежності від значень на попередніх етапах. Мовні моделі, в свою чергу, не є лінійними, через наявність численних шарів з нелінійними перетвореннями. Проте цей процес все одно прийнято називати авторегресивною генерацією, тому, що кожне згенероване слово залежить від попереднього, вибраного мережею на попередньому кроці. Також, існують інші мовні моделі, такі як масковані, вони не причинно-наслідкові, оскільки для передбачення слова враховують як попередні, так і наступні слова.

На кожному кроці слово обирається з розподілу ймовірностей, заданого моделлю в відповідному контексті. У результаті слова з високою ймовірністю в цьому контексті мають більший шанс бути згенерованими, тоді як малоймовірні слова обираються рідше. Так відбувається генерація тексту в уніграмній мовній моделі, багаторазове випадкове вибирання слів відповідно до їхніх ймовірностей, доки не досягнеться заздалегідь визначена довжина чи певний токен кінця речення. LLM при генерації передбачає подібний підхід, але на кожному етапі вибирається слово згідно з його ймовірністю, яка, в свою чергу, визначається попереднім вибором. В якості архітектури моделі для предбачень цих ймовірностей використовується трансформер.

На практиці, метод звичайної випадкової вибірки виявляється не надто ефективним. Основний недолік полягає в тому, що, попри генерацію даним підходом логічних та високоймовірних слів, у нижній частині розподілу залишається чимало малоймовірних. Хоча ймовірність кожного з них окремо невелика, їх сукупний внесок у розподіл виявляється значним. Цей фактор призводить до частого вибору таких слів, що створюють алогічні фрази.

Через недоліки властиві випадковій вибірці зазвичай застосовують інші підходи, для мінімізації ймовірністі генерації слів з дуже низькою імовірністю. Такі підходи дозволяють налаштовувати параметри для досягнення балансу між двома ключовими аспектами: якістю та різнообразністю. Методи, що фокусуються на виборі найбільш ймовірних слів, зазвичай створюють тексти, які сприймаються людьми як логічні, точні та зв’язні, але водночас можуть

бути занадто передбачуваними й повторюваними. Підходи ж, що, навпаки, враховують і середні за ймовірністю варіанти, часто генерують більш різноманітний і творчий текст, хоча він схильний до втрати зв'язності, і загалом мати гіршу якість.

Метод вибору k найкращих є деяким вдосконаленням жадібного декодування. Відмінність полягає в тому, що обирається k найбільш ймовірних слів, далі вони нормалізуються за ймовірностями та випадковим чином обираються відповідно до нормалізованої ймовірності. Тоді як в жадібному декодуванні просто обирається одне найбільш ймовірне слово. У випадку, коли значення k перевищує одиницю, з'являється можливість вибору відносно менш ймовірних слів, на противагу вибору найбільш ймовірних, що сприяє підвищенню різноманітності тексту, зберігаючи його якісний рівень. При значенні $k = 1$, цей метод збігається з тим самим жадібним декодуванням.

Одним з недоліків методу найкращих k є його залежність від сталого значення, бо розподіл ймовірностей слів може змінюватися залежно від контексту. Наприклад, якщо обрати $k = 30$, у деяких випадках ці 30 слів матимуть високу ймовірність і охоплюватимуть більшу частину ймовірнісної маси, проте в інших ситуаціях розподіл може бути більш рівномірним, і ці 30 слів охоплять лиш невелику частку.

Іншим підходом є вибірка за ймовірнісним порогом, найкращих p. У даному підході акцент робиться не на виборі фіксованої кількості слів, як у найкращих k, а на збереженні сукупної частки ймовірносної маси, що дорівнює p. Основна ідея залишається незмінною — відсікати малоймовірні варіанти, проте використання ймовірності замість фіксованої кількості слів дозволяє адаптивно змінювати обсяг вибірки залежно від конкретного контексту, що сприяє забезпеченню більшої гнучкості.

Метод найкращих p, показаний в формулі 3.2, визначає найменший набір токенів, загальна ймовірність яких перевищує заданий поріг p.

$$\sum_{w \in V(p)} P(w) \geq p, \tag{3.2}$$

де $P(w)$ – ймовірність того, що слово w буде наступним;

V – словник ймовірних токенів;

p – значення порогу.

У методі вибірки за температурою кількість можливих варіантів не зменшується, а змінюється сам розподіл ймовірностей. Концепція вибірки за температурою базується на законах статистичної термодинаміки. Коли температура низька система, найімовірніше, зосередиться лише на станах, які мають більшу енергетичну ефективність, натомість, коли температура висока, система демонструє значну гнучкість і здатність досліджувати широкий спектр станів. У вибірці з низькою температурою підвищується ймовірність найбільш ймовірних слів і знижується менш ймовірних.

Цей метод реалізується, просто розділяючи логіт на параметр температури t перед його нормалізацією через softmax. У вибірці з низькою температурою значення t знаходиться в інтервалі $(0;1]$. Таким чином, логіти перед потраплянням до софтмакс, спочатку діляться на параметр температури, і вже потім обраховується функція для нормалізації, формула 3.3.

$$y = \sigma(\frac{u}{t}), \tag{3.3}$$

де σ – функція активації софтмакс;

u – значення логіту;

t – значення температури.

При наближенні значення температури t до 1, зміни в розподілі ймовірностей залишаються мінімальними. Проте, коли t знижується, модель

починає акцентувати увагу на найвищих значеннях, що передаються до функції нормалізації софтмакс, що володіє такою особливістю, яка підвищує високі значення, наближаючи їх до одиниці, в той час, зменшує малі значення, наближаючи їх до нуля. Таким чином, при низьких температурах, розподіл стає жорсткішим: ймовірності найімовірніших слів суттєво зростають, а ймовірності малоймовірних слів значно знижуються. У крайніх випадках, коли t прямує до нуля, вибір фактично обмежується найімовірнішим словом, оскільки його ймовірність стає практично рівною одиниці. В деяких задачах необхідно уникати надмірної концентрації уваги на окремих словах і надавати більшу рівномірність розподілу. Такий ефект досягається шляхом збільшення температури, що сприяє зниженню різниці між ймовірностями слів. У такому випадку навіть менш ймовірні слова мають шанс бути обраними, параметр температури дозволяє гнучко регулювати баланс між точністю і різноманітністю та творчістю у процесі генерації тексту.

4 РЕАЛІЗАЦІЯ РІШЕННЯ ГЕНЕРАЦІЇ ПОЕЗІЇ ТА АНАЛІЗ ЙОГО ЯКОСТІ

4.1 Застосовані технології та інструменти

Для налаштування моделі генерації поезії було застосовано бібліотеки transformers та PEFT, середовище Kaggle для навчання.

В якості набору даних використовувались антології української поезії та пісні з Kaggle датасету genius-song. Для обробки датасету була вибрана бібліотека NumPy та Pandas.

Для реалізації веб-додатку, через який буде взаємодіяти користувач з моделлю, використовується фреймворк Flask для обробки запитів та взаємодії з моделлю, який працює в середовищі Kaggle. Для інтеграції з веб-додатком було обрано технологію ngrok, яка дозволяє створити з'єднання між сервером та клієнтським інтерфейсом, що реалізований за допомогою JavaScript.

Kaggle – це спеціальне середовище для аналітики та машинного навчання, яке дозволяє працювати в хмарі без необхідності встановлювати програмне забезпечення локально. Надає можливість запускати Python-код, аналізувати дані, будувати графіки та тестувати моделі машинного навчання, що особливо корисно при роботі з великими даними, як у випадку з моїм проєктом генерації поезії. Перевагою Kaggle є безкоштовні хмарні обчислення, операції виконуються на серверах Kaggle, що зменшує навантаження на локальний комп’ютер і дозволяє працювати з великими датасетами.

Оскільки навчання нейронних мереж і виконання скриптів це ресурсомісткий процес, використання Kaggle Notebooks є зручним рішенням для дослідників, інженерів і спеціалістів з даних, які шукають простий спосіб оптимізувати свої робочі процеси без фінансових витрат.

Flask – компактний та гнучкий веб-фреймворк. Він надає основні інструменти для створення веб-застосунків чи API, надаючи програмісту

максимальну свободу в реалізації функціоналу. Початково створений командою Pocoo, фреймворк пізніше перейшов під управління Pallets Project, яка відповідає за його розвиток і оновлення. Завдяки активній підтримці спільноти та численним онлайн-ресурсам, розробники мають можливість швидко знайти відповіді на свої запитання.

Flask відзначається гнучкістю, що дозволяє підключати лише потрібні компоненти, спрощуючи налаштування, розгортання і контроль роботи API, незалежно від масштабу чи особливостей проєкту. Flask базується на двох основних елементах: Werkzeug, що відповідає за маршрутизацію, підтримку інтерфейсу WSGI та Jinja2, що виконує функцію шаблонізатора [13]. Незважаючи на відсутність вбудованих засобів для роботи з базами даних або автентифікації, цей фреймворк дозволяє легко додавати такі можливості завдяки широкому вибору розширень. Для невеликих застосунків весь код можна розмістити в декілька десядків рядків. Головна ідея Flask полягає в наданні мінімального, але гнучкого каркасу, який можна адаптувати до потреб проєкту за допомогою додаткових інструментів і бібліотек. Flask має модульну архітектуру, яка дозволяє розробникам додавати необхідну функціональність через зовнішні бібліотеки, уникаючи перевантаження непотрібними функціями.

В даному проєкті Flask обробляє HTTP-запити, приймаючи введені користувачем дані – початок поезії, і передає їх до моделі генерації поезії. Отримавши результат обробки даних від моделі, Flask формує відповідь у вигляді JSON, що зручно для подальшого використання у клієнтському інтерфейсі. Flask легко налаштувати для інтеграції моделлю та інструментами, такими як Ngrok, для надання публічного доступу до сервера.

4.2 Підготовка даних

Для навчання моделі для генерації поезії було зібрано відповідний набір даних, який складається з текстів українських поезій та їх анотацій.

Враховуючи досліджені джерела та наявні антології, було прийнято рішення використовувати два томи української поезії, а також інші збірки, такі як "Сади Адоніса", "З під евкаліптів", "Гербарій" та українські пісні з Kaggle датасету "genius-song". Загальний розмір набору даних складає 4800 об’єктів, причому більша частина даних - пісні.

Датасет включає в себе різні типи поезії – від коротких віршів до більших поетичних творів, по типу “Гайдамаки”, Тараса Шевченка, приклад зображено на рисунку 4.1. Якщо поетичний твір є занадто довгим для ефективного навчання моделі, він розбивається на менші частини. В умовах обмежених ресурсів відеокарти, розбиття на частини дозволяє моделі краще обробляти великі обсяги тексту, оскільки занадто довгі твори можуть мати складні структури, які важко піддаються навчанню. Перевантаження вхідної пам'яті моделі, призводить до того, що контекст не встигає обробити всю необхідну інформацію. Обрізка тексту до 256 токенів, рисунок 4.2, показала оптимальний варіант між достатністю контексту та обсягом, який модель може ефективно обробити. Трансформери використовують механізм уваги, який має складність $O(n^2)$, де n — кількість токенів у послідовності. Обсяг пам'яті, необхідний для збереження всіх зв'язків між токенами, швидко зростає. Зокрема, для 512 токенів пам'ять зростає в 4 рази порівняно з 256 токенами, як результат пам’яті відеокарти не вистачає. Також розбиття на менші частини дозволяє зосередитись на кожному уривку окремо, покращити точність моделі у відтворенні відповідних запиту та граматично правильних результатів.


```
{
    "index": 10,
    "excerpt": "Що вона потім робила, куди пішла,\nз хусткою в рукаві й каблучкою на мізинці,\nколи її темне вікно
    роз’їдала імла,",
    "poem": "\nяк іржа роз’їдає старі есмінці?\nКурила бельгійський тютюн,\nміцніший, ніж зазвичай.\nСварилася з
    поліцейськими - п’яна і грізна.\nЛюбила сухе вино, пила у фаст-фудах чай -\nіндійський, мов океан,\nчорний, як
    власна білизна.\nЙ пила за те, що ніхто не зможе її знайти,\nза те, що проймається спокоєм\nдуші золота матерія.
    \nЛежала на теплому спальнику -\nоголена, мов дроти,\nтиха, наче вода, сонна, ніби артерія.\n \nЩо по ній
    залишилось? Якісь борги,\nякі я поволі сплачував, книги і мапи,\nякісь випадкові друзі, якісь вороги,\nяких я
    насправді не знав,\nхоч насправді мав би.\nЛишилися речі в її шухляді, ніби в журбі,\nкалендарик із її місячними,
```


Рисунок 4.1 – Приклад об’єкта з датасету

В полі excerpt міститься початок поетичного твору випадкової довжини. Використання різних уривків із поетичних текстів у якості початкового контексту дозволяє моделі навчитися розпізнавати широкий спектр тем, стилів, цим самим покращуючи її здатність до якісної генерації.

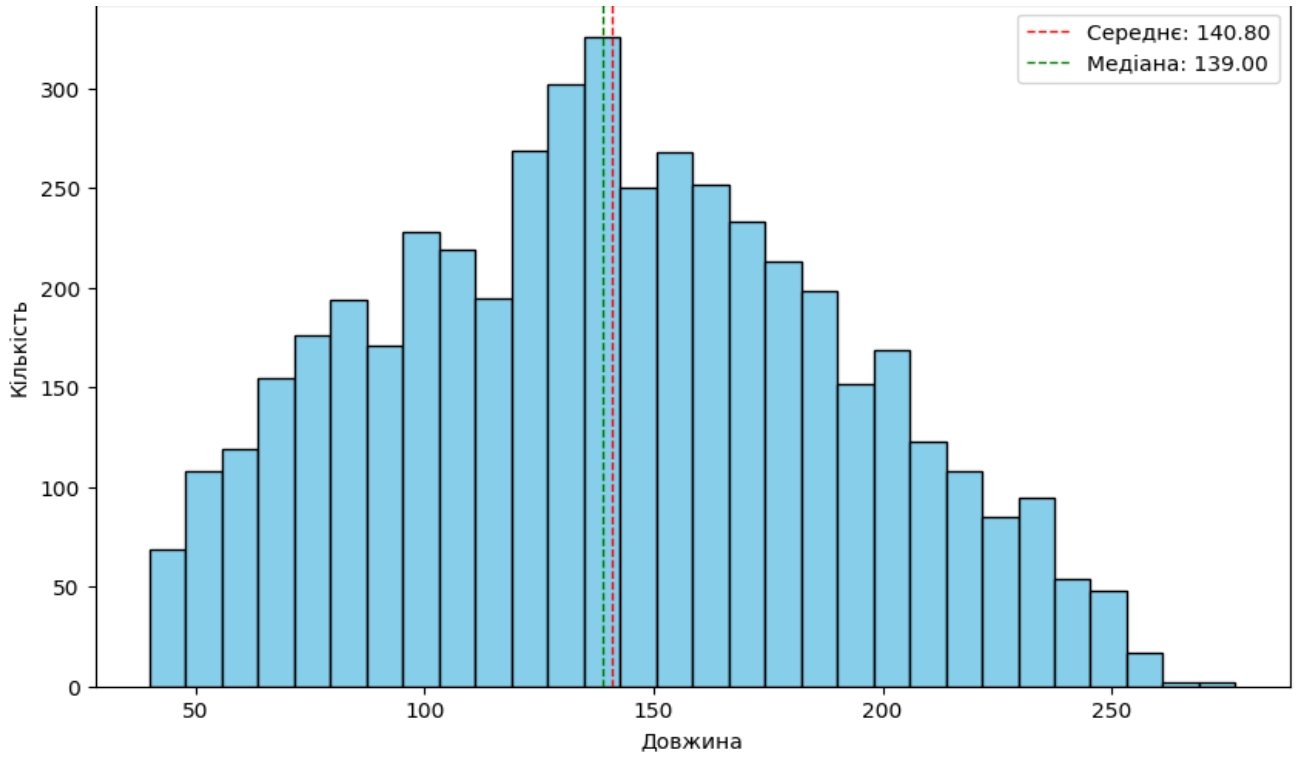

Рисунок 4.2 – Розподіл довжин віршів

Для забезпечення чистоти та релевантності тексту у поезії для подальшого навчання моделі, було проведено кілька етапів попередньої обробки. Відфільтровано тексти, що були надто короткими, менші за 40 слів, дублікати, початкові вигуки, непотрібні символи та слова з окремих віршів, такі як позначення куплетів, слова на кшталт "приспів" та інші маркери, що використовуються для структуризації пісні, але не несуть значення для генерації поезії.

4.3 Характеристики моделі та її навчання

У даній роботі для задачі генерації поезії було використано модель Gemma 2B, яка є типовим представником SOTA генеративних мовних моделей. Gemma 2B базується на архітектурі декодера трансформера, подібно до інших передових моделей, таких як LLaMA 3B. Обидві моделі демонструють високі показники у задачах генерації і мають оптимальний розмір для використання на безкоштовних обчислювальних ресурсах Kaggle. Подальше донавчання базується на використанні бібліотек PEFT і Transformers, що дозволяє застосовувати ефективні методи адаптації, такі як LoRA. Налаштування виконується за допомогою модуля SFTTrainer із бібліотеки TRL (Transformers Reinforcement Learning). Для оптимізації використання памяті застосовується метод 4-бітної квантизації для базової мовної моделі QLoRA (Quantized Low-Rank Adaptation). Параметри конфігурації визначаються через клас BitsAndBytesConfig, лістинг 4.1.

Лістинг 4.1 – Параметри класу *BitsAndBytesConfig*

```
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch_dtype,
    bnb_4bit_use_double_quant=True,
)
```

Оптимізована базова модель Gemma-2b має наступні характеристики. Розмірність вхідного ембединг вектора становить 256 тисяч токенів кожен з яких 2048 елементів. У моделі передбачено 18 шарів GemmaDecoderLayer. Кожен шар складається з механізму самоуваги GemmaAttention, проєкції запитів (q_proj), ключів (k_proj), значень (v_proj) якої виконуються у форматі 4-бітної квантизації та мають розмірність вхідних параметрів 2048 та 256 на виході. MLP блоку GemmaMLP, який містить три проєкції: gate_proj, up_proj, down_proj, що працюють у 4-бітному форматі також. В ролі функції активації

застосовано PytorchGELUTanh, вона комбінує властивості гіперболічного тангенсу та GELU для кращої нелінійності. Середньоквадратичної нормалізації GemmaRMSNorm. Вхідна, після уваги та загальна, кожна з яких зі значенням $\varepsilon = 1e-6$, зменшує вплив різниці в масштабах значень.

Модуль lm_head - це останній шар в архітектурі трансформера, який відповідає за перетворення вихідних прихованих станів моделі в ймовірності для кожного токену в словнику.

Для пришвидшення навчання застосовується адаптер, що зробить процес більш швидким і економним за використанням пам'яті. Адаптер створюється для конкретних модулів трансформера(уваги та багатошарового персептрону) з урахуванням специфіки завдання, лістинг 4.2. Було проведено експеременти з двома помірними значеннями рангу 8 та 16, таблиця 4.1. Ранг відповідає за розмірність простору, на який проектуються ваги кожного модуля.

Таблиця 4.1 – Залежність кількості параметрів відносно рангу

Ранг	Тренувальні параметри	Всього параметрів	Відсоток тренувальних параметрів	Кількість параметрів на вірш
8	10 383 360	2 624 725 248	0,3956	2403
16	20 766 720	2 635 108 608	0,7881	4806

Параметр адаптеру, lora_alpha, масштабний коефіцієнт, який контролює, наскільки сильно адаптери впливають на модель, порівняно з її основними вагами. Значення 16 було обрано експериментальним шляхом. Параметр lora_dropout відповідає за ймовірність обнулення певних ваг адаптера під час тренування для запобігання перенавчанню.

Лістинг 4.2 – Параметри класу LoraConfig

```
lora_config = LoraConfig(
    r=16,
```

```
    lora_alpha=16,
    target_modules=modules,
    lora_dropout=0.1,
    bias="none",
    task_type="CAUSAL_LM"
)
```

З міркувань економії памяті, яку використовує модель, було застосовано метод градієнтного накопичення, лістинг 4.3, який дозволяє ефективно замінювати великий розмір пакету(batch). Суть даного методу полягає в обчисленні та накопиченні градієнтів по маленьких батчах, замість обробки всіх прикладів в одному великому батчі. Коли кількість кроків досягає заданого параметра gradient_accumulation_steps – відбувається оновлення ваг моделі. Швидкість навчання $4e-5$ була підібрана експерементальним шляхом, при значенях на порядок нижчих відбулось падіння валідаційних втрат до нуля. Також із застосуванням warmup_steps, перші 100 кроків швидкість навчання буде зростати від 0 до $4e-5$, що дозволяє моделі "пристосуватись" до задачі без різких коливань, зменшить ймовірність того, що модель почне тренуватися з великою помилкою. В якості оптимізатора використано AdamW у квантованому 8-бітному форматі з підтримкою розподілених сторінок пам'яті для зниження використання пам'яті та покращення швидкості обчислень.

Лістинг 4.3 – Параметри класу TrainingArguments

```
args=transformers.TrainingArguments(
        per_device_train_batch_size=1,
        gradient_accumulation_steps=8,
        num_train_epochs=2,
        learning_rate=4e-5,
        logging_steps=1,
        output_dir="outputs",
        optim="paged_adamw_8bit",
        save_strategy="epoch",
        eval_strategy="steps",
        warmup_steps=100,

    )
```

Перевіримо міркування про те, що модель, яка вже проходила тонке

налаштування на наборі даних певної мови – має більше знань про неї, було взято модель akiulian/uagemma-2b-test [14]. Попереднє тонке налаштування даної моделі проводилось на комбінованому наборі даних, що складається з 13 063 інструкцій, які включали 10 000 рядків набору даних UAlpaca та 3 063 рядків з набору даних ЗНО, призначених для адаптації моделі до формату з вибором відповідей, поширеного в українському національному іспиті. Повторне тонке налаштування проведено на наборі віршів. На рисунку 4.3 продемонстровано графік функції втрат на тренувальному та валідаційному наборі даних в залежності від різних значень рангу та швидкості навчання.

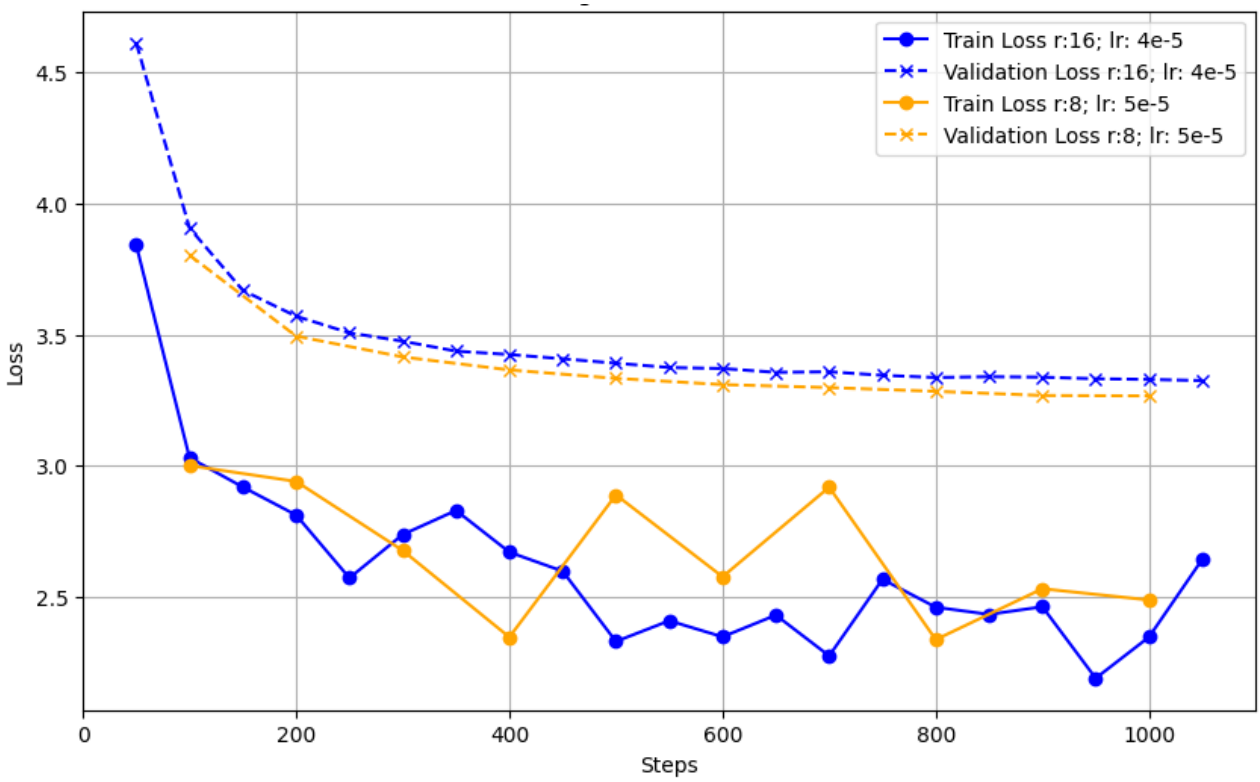

Рисунок 4.3 – Графік функції втрат на тренуванні та валідації при різних параметрах рангу та швидкості навчання

При використанні меншого значення рангу валідаційна помилка зменшується, але не значно. Різниця між значеннями рангів 8 та 16 виявляється мінімальною відносно валідаційних втрат. Для генерації було підібрано

параметри, що запобігають утворенню надмірно повторюваного або зациклено тексту, водночас забезпечуючи баланс між креативністю і осмисленістю: значення температури 0.85, найкращих k - 50. Модель відбирає 50 токенів та надає їм виразності за допомогої температури. В умовах обмежених тренувальних даних важливо застосовувати техніку cherry picking – процес вибору найкращих результатів серез згенерованих моделлю, ігноруючи менш вдалі варіанти. Приклад згенерованого вірша даною моделлю зображено на рисунку 4.4.

Чому мені сниться, як знову і знову
Гуляєм з тобою по рідному Львову
Там пахне весною і сонце сідає
На березі річки, якої немає
І дивляться леви на тебе ласкаво
І варять бажання із запахом кави
Наче мовчання до ранку і снів.

Я не знаю, що, может, вибачати за це!
Але все ж прощайся, я не з тобою, я,
Я не знаю, що не бачу — тільки квіти,
Звичайно, про щось, мов, минають ночі.

Гори світять в колі і зорі прокидають у небо
І очей у мене нема вже.
Де будеш? Де йдеш?
Де сльози відрікаються і надії падіють.

У темряві, де суцвіття вмирає,
Мій улюблюй рідний край.

Рисунок 4.4 – Приклад генерації

Було проведено навчання базової моделі gemma-2b, графік функції втрат при різних параметрах навчання зображено на рисунку 4.5. Протестовано такі ранги та швидкості навчання, як і в попередньому тонкому налаштуванні.

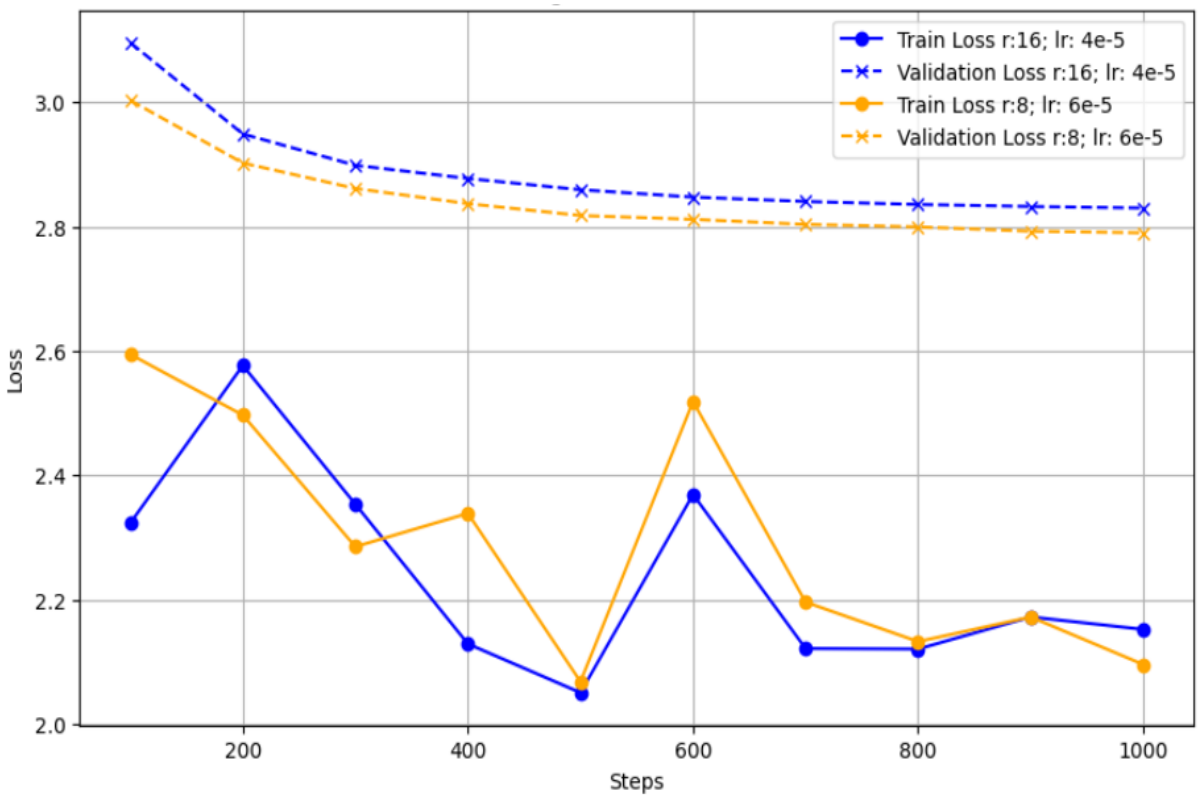

Рисунок 4.5 – Графік функції втрат на тренуванні та валідації при різних параметрах

Різниця між значеннями рангів 8 та 16 є незначною щодо валідаційних втрат. В свою чергу, показники втрат як на валідації так і під час тренування зменшились в абсолютному вимірі на 0.7 одиниць відповідно, порівняно з попереднім тонким налаштуванням вже налаштованої моделі на інших українських даних. Це може свідчити, про те що модель akiulian/uagemma-2b-test була перенавчена на конкретному датасеті, це погіршило здатність генералізації на нових даних – віршах. Коли модель налаштовується на специфічному наборі даних, вона може забути загальні патерни, які були навчальні під час базового тренування. Було відібрано методом cherry-picking вірш за таким самим запитом, як і для попередньої моделі, рисунок 4.6.

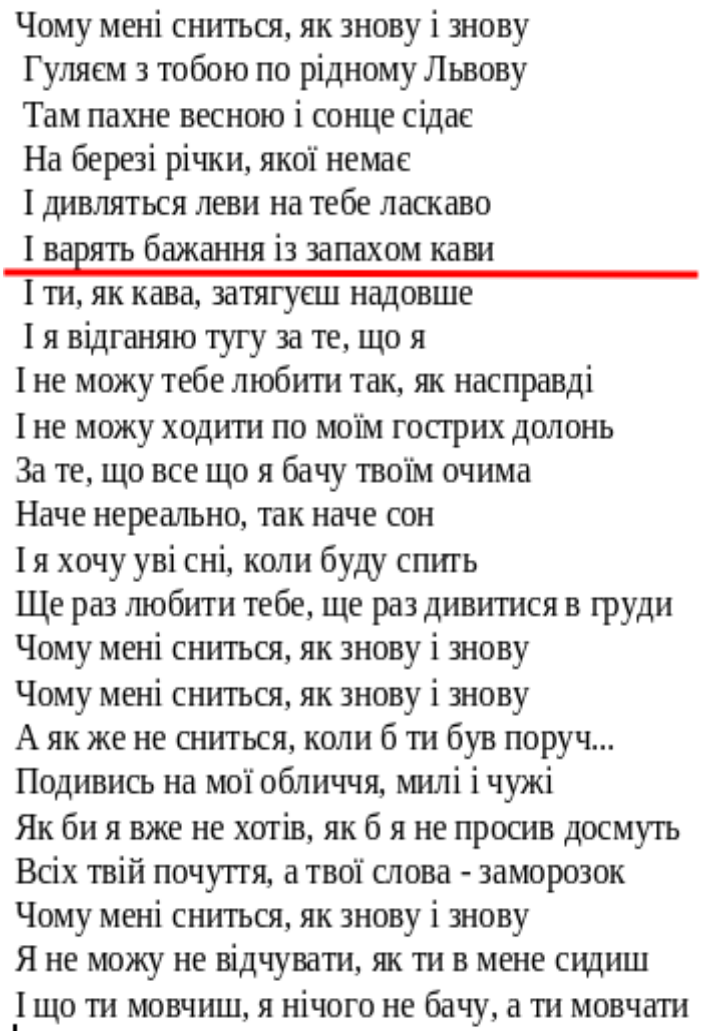
Чому мені сниться, як знову і знову
Гуляєм з тобою по рідному Львову
Там пахне весною і сонце сідає
На березі річки, якої немає
І дивляться леви на тебе ласкаво
І варять бажання із запахом кави
І ти, як кава, затягуєш надовше
І я відганяю тугу за те, що я
І не можу тебе любити так, як насправді
І не можу ходити по моїм гострих долонь
За те, що все що я бачу твоїм очима
Наче нереально, так наче сон
І я хочу уві сні, коли буду спить
Ще раз любити тебе, ще раз дивитися в груди
Чому мені сниться, як знову і знову
Чому мені сниться, як знову і знову
А як же не сниться, коли б ти був поруч...
Подивись на мої обличчя, милі і чужі
Як би я вже не хотів, як б я не просив досмуть
Всіх твій почуття, а твої слова - заморозок
Чому мені сниться, як знову і знову
Я не можу не відчувати, як ти в мене сидиш
І що ти мовчиш, я нічого не бачу, а ти мовчати

Рисунок 4.6 – Приклад генерації вірша налаштованою базовою моделлю

Модель з тонким налаштуванням akiulian/uagemma-2b-test має креативний підхід в плані образності тексту, але вона схильна до досить нелогічних поєднань, граматичних помилок та перевантаженості образами, це робить текст важким для сприйняття. Базова модель генерує поезії, що виглядають більш зрозуміло і структуровано, з меншим хаосом в образах і більш чітким вираженням емоцій, проте інколи вона не дає повної розв’язки. Наприклад, в кінці генерації трапляються неповні фрази, значення яких важко інтерпретувати.

4.4 Виділення та оцінка метрик якості генерації поезії

Для оцінки якості згенерованої поезії було обрано метрики, що визначають, якою повинна бути гарна поезія [15], а також проведено їх аналіз.

Різноманіття лексики . Дана метрика оцінює співвідношення TTR (Type-Token Ratio), тобто кількість унікальних токенів відносно загальної кількості токенів у кожному тексті. Високе співвідношення свідчить про багатство словникового запасу, різноманітність рим і варіацій. Водночас надмірна лексична різноманітність не завжди є ознакою високої якості поезії, оскільки важливими є також цілісність змісту, гармонійність і стилістична відповідність тексту. Розрахунок TTR приведено в таблиці 4.2.

Таблиця 4.2 – Порівняння TTR на згенерованих віршах, та віршах з датасету.

Набір даних	Кількість віршів	Різноманіття лексики			
		μ	σ	Max	Min
Тренувальний	356	81.13%	7.93%	100%	51.8%
Gemma-2b	356	88.09%	5.28%	97.89%	65.14%
Gema-2b cherry picking	7	78.32%	10.90%	97.83%	58.36%

Середнє значення TTR на згенерованому наборі даних без використання техніки cherry picking, вище, аніж на тренувальному, свідчить про те, що модель в цілому схильна до різноманітності в генерації. Проте вірші відібрані за допомогою cherry picking мають майже таке ж середнє значення TTR. Стандартне відхилення є меншим на віршах згенерованих без використання cherry picking, що вказує на меншу стабільність різноманіття лексики, в той же

час на відібраних віршах стандартне відхилення, навпаки трішки більше, це свідчить про гарну стабільність в триманні рівня різномаїття. Максимальне значення TTR має схожі показники, в той час, як мінімальне є найвищим на згенерованих віршах без використання cherry picking.

Змістова і тематична цілісність. Високоякісна поезія характеризується логічним розвитком сюжету, де кожен рядок гармонійно підтримує і розвиває основну ідею, створюючи зв'язний текст. Однак, у згенерованих поемах часто спостерігається зміна тематики між рядками, використання невизначених займенників і несумісних іменників, що негативно впливає на цілісність твору. Іменники вважаються основними маркерами сутностей і концептів, слугують ефективним індикатором теми поезії, оскільки вони фокусують увагу на ключових аспектах змісту, зменшуючи вплив другорядних елементів мови [16].

Щоб визначити тематичну узгодженість у поемі, було проведено вимірювання семантичної схожості іменників у поезіях, що здійснювалося через різницю відстаней між їх векторними представленнями.

Середнє значення цих відстаней характеризує рівень розсіювання іменників щодо основної тематики: чим більше значення, тим сильніше слова віддаляються від центральної теми. Стандартне відхилення використовується для аналізу узгодженості іменників: низьке значення вказує на їхню тематичну схожість, тоді як високе може свідчити про наявність кількох різних тем у тексті через значну віддаленість окремих іменників.

Виявлення іменників у поезії проводилось за допомогою української моделі uk_core_news_sm з бібліотеки SpaCy. Векторні представлення для кожного іменника отримувалися за допомогою моделі cbow.uk.300.bin. Ця модель, створена за допомогою бібліотеки FastText і навчена за підходом CBOW(Continuous Bag of Words), для побудови векторних представлень слів. Навчання проводилось на наборі даних UberText2.0. Вона генерує 300-вимірні вектори для слів української мови. Схожість між векторами іменників вимірювалась за допомогою методу косинусної подібності.

На рисунку 4.6, зліва, середнє значення відстані між іменниками та центроїдом іменників у поемах датасету дещо більше, ніж у згенерованих поемах, тоді як стандартне відхилення для поезій датасету є меншим. Це вказує на те, що в поемах датасету іменники, хоча й розташовані далі від центроїда в середньому порівняно із згенерованими, все ж більш однорідніше групуються навколо нього. Тоді як, згенеровані поеми мають більшу варіативність у відстанях від центроїда, що свідчить про наявність різноманітніших образів чи ідей що не завжди логічно зв'язуються в єдину гармонійну тему. З підходом cherry picking для вибору кращих згенерованих поезій середнє значення відстані до центроїду та стандартне відхилення залишаються на рівні з поемами з датасету. Тобто, обираючи лише найбільш якісні згенеровані поеми, ми отримуємо поезії з узгодженістю тем на рівні датасету.

Також, для обчислення оцінки безперервності теми в згенерованих та поезіях з датасету, було розраховано середнє значення попарних відстаней між усіма іменниками в поезії. Результати цих обчислень представлені на діаграмі на рисунку 4.6. Поезія з датасету в цілому демонструє більшу середню попарну схожість векторних представлень іменників, що вказує на те, що іменники в цих текстах є більш семантично подібними між собою. Це свідчить про високий рівень узгодженості теми в поемах. Натомість, згенеровані поезії мають більші середні попарні відстані, що вказує на деяку слабшу семантичну сумісність між іменниками, що свідчить про проблеми з безперервністю теми, де окремі елементи змісту не завжди гармонійно поєднуються.

Проте, при виборі поезій за допомогою підходу cherry-picking, де були відібрані найбільш вдалі згенеровані тексти, з точки зору користувача, середня попарна схожість векторних представлень іменників наближається до рівня поезій з датасету.

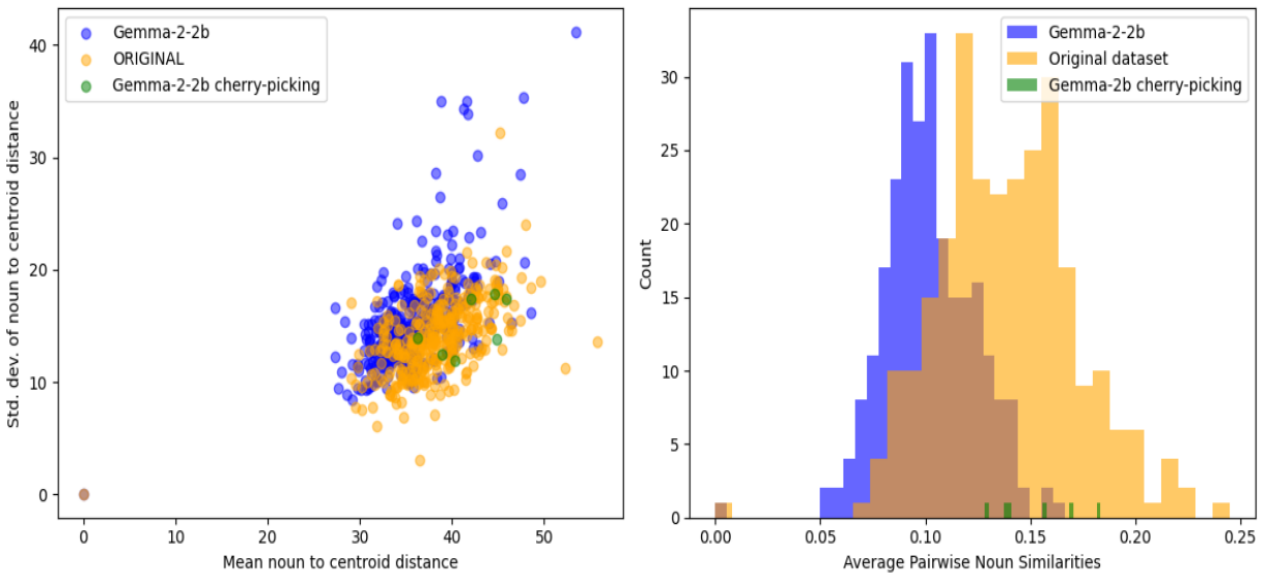

Рисунок 4.6 – Стандартне відхилення, середня відстань до центроїду та попарна схожість іменників у згенерованих та віршах з датасету

4.5 Реалізація додатку

Був розроблений простий веб додаток для демонстрації роботи моделі генерації поезії, схема якого зображена на рисунку 4.7. Такий додаток включає можливість приймати текстові запити від користувача, взаємодіяти з моделлю для створення поетичних творів і відображати згенеровані результати. У додатку також була передбачена опція збереження створених текстів для подальшого їх аналізу.

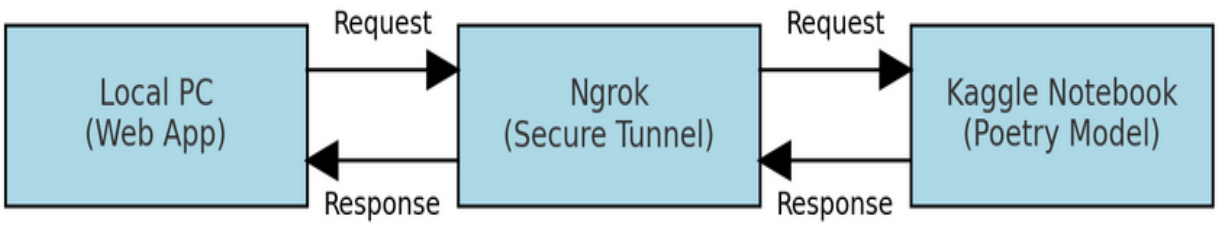

Рисунок 4.7 – Схема роботи додатку

Веб-застосунок складається з однієї сторінки, яка включає текстове поле для введення запиту, початку вірша, та прихованого блоку для відображення згенерованого тексту, який з'являється після отримання відповіді від сервера.

Клієнтська частина застосунку відповідає за надсилання запиту на сервер через API, лістинг 4.3. А також обробку відповіді сервера та її виведення у зручному для користувача вигляді, а також обробку помилок у разі проблем із запитом.

Лістинг 4.3 – Запит *fetch()* для отримання згенерованого тексту

```
fetch('https://2bc8-104-199-115-161.ngrok-free.app/generate', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ prompt: prompt })
    })
    .then(response => {
        if (!response.ok) {
            throw new Error('Network response was not ok');
        }
        return response.json();
    })
    .then(data => {
        // Виводимо відповідь на сторінці
        const resultDiv = document.getElementById('result');
        const responseText = document.getElementById('response-
text');
        resultDiv.style.display = 'block';
        responseText.textContent = data.response;
    })
    .catch(error => {
        console.error('Error:', error);
        alert('Виникла помилка при генерації тексту.' +
error.message);
    });
```

Завдяки своїй легкості у використанні, гнучкості в налаштуванні та можливості швидкої інтеграції зі сторонніми бібліотеками, для розробки серверної частини системи було обрано фреймворк Flask. Цей мікрофреймворк надає всі необхідні інструменти для обробки HTTP-запитів, керування маршрутами та інтеграції з іншими компонентами додатка. В лістингу 4.4

показано процес обробки запиту, що відповідає за генерацію поезії.

Лістинг 4.4 – Процес обробки запиту

```
@app.route('/generate', methods=['POST'])
def generate():
    data = request.get_json()
    prompt = data.get('prompt', 'No prompt provided')

    device = model.device
    input_ids = tokenizer(prompt, return_tensors="pt").input_ids
    input_ids = input_ids.to(device)
    output = model.generate(input_ids, max_new_tokens=250,
num_return_sequences=1, temperature=0.7, top_k=40, do_sample=True)
    generated_text = tokenizer.decode(output[0],
skip_special_tokens=True)

    return jsonify({"response": f"{output}"})
```

Для забезпечення доступу до серверної частини, яка виконується в Kaggle ноутбуці використовується сервіс ngrok, лістинг 4.5. Він створює безпечний тунель, для доступу до локального сервера Flask із зовнішнього інтернету. Ngrok дозволяє обійти обмеження доступу до сервера, розгорнутого у середовищі Kaggle. Локальний API ngrok повертає дані про створені тунелі у форматі JSON. Далі надається інформація про публічний URL, через який можна отримати доступ ззовні до Flask-додатка.

Лістинг 4.5 – Налаштування ngrok

```
# Запускаємо ngrok через subprocess
ngrok_process = subprocess.Popen(['ngrok', 'http', '5000'])

# Чекаємо кілька секунд, щоб ngrok налаштував тунель
import time
time.sleep(5)  # Затримка для ініціалізації ngrok

# Отримуємо публічний URL ngrok
url = subprocess.check_output(['curl',
'http://localhost:4040/api/tunnels'])
```

4.6 Тестування додатку

Процес роботи веб-застосунку складається з кількох етапів. Спершу користувачу потрібно ввести текстовий запит у форму на головній сторінці та натискає кнопку "Згенерувати", рисунок 4.8. Генерувати потрібно до тих пір, доки не знайдеться гарний результат.

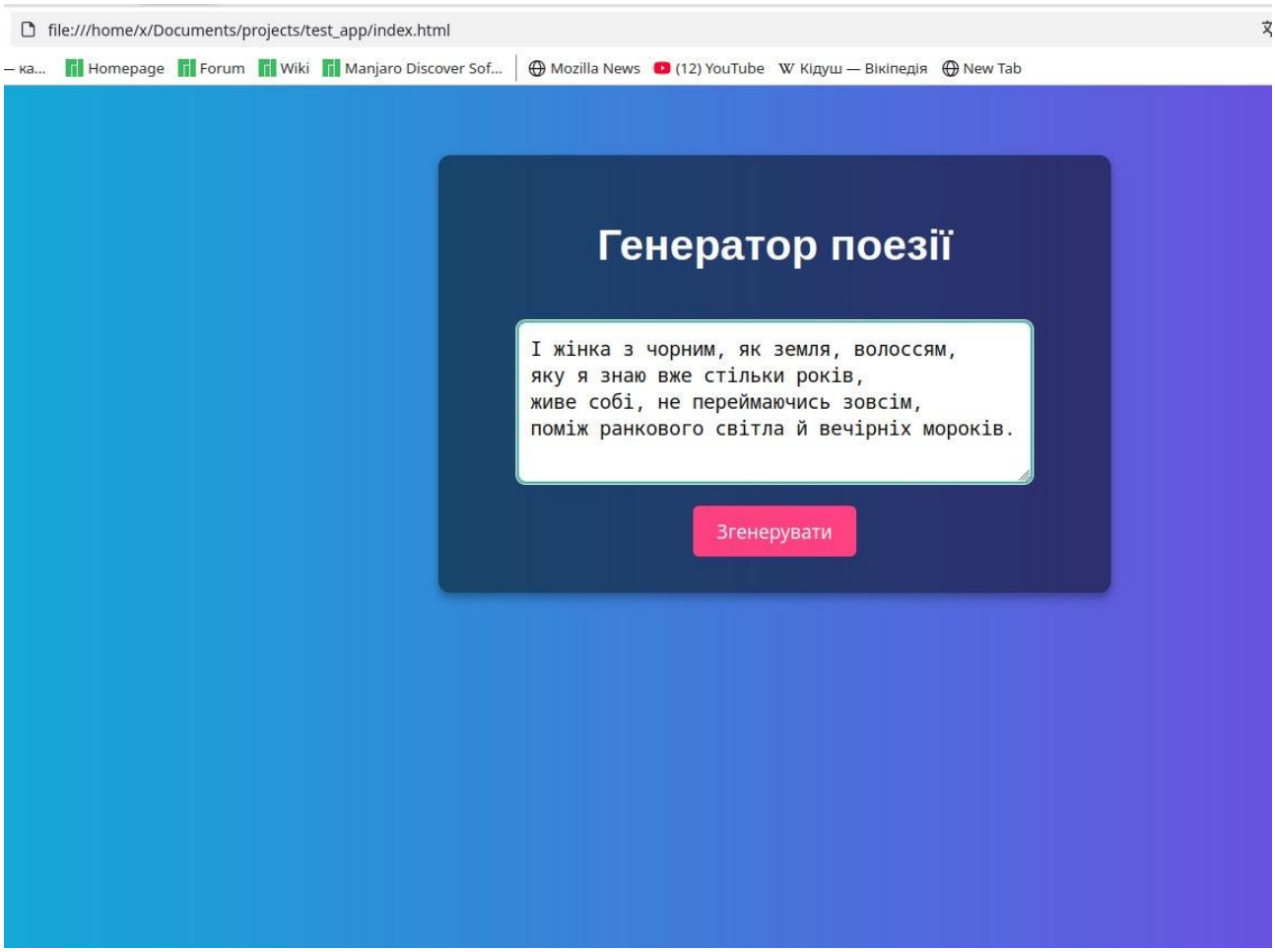

Рисунок 4.8 – Введення рядків, за якими буде проходити генерація

Запит передається на сервер, де модель генерації текстів його обробляє. Після цього, у випадку успішного виконання, оброблений результат надсилається назад до клієнта й відображається на екрані, рисунок 4.9.

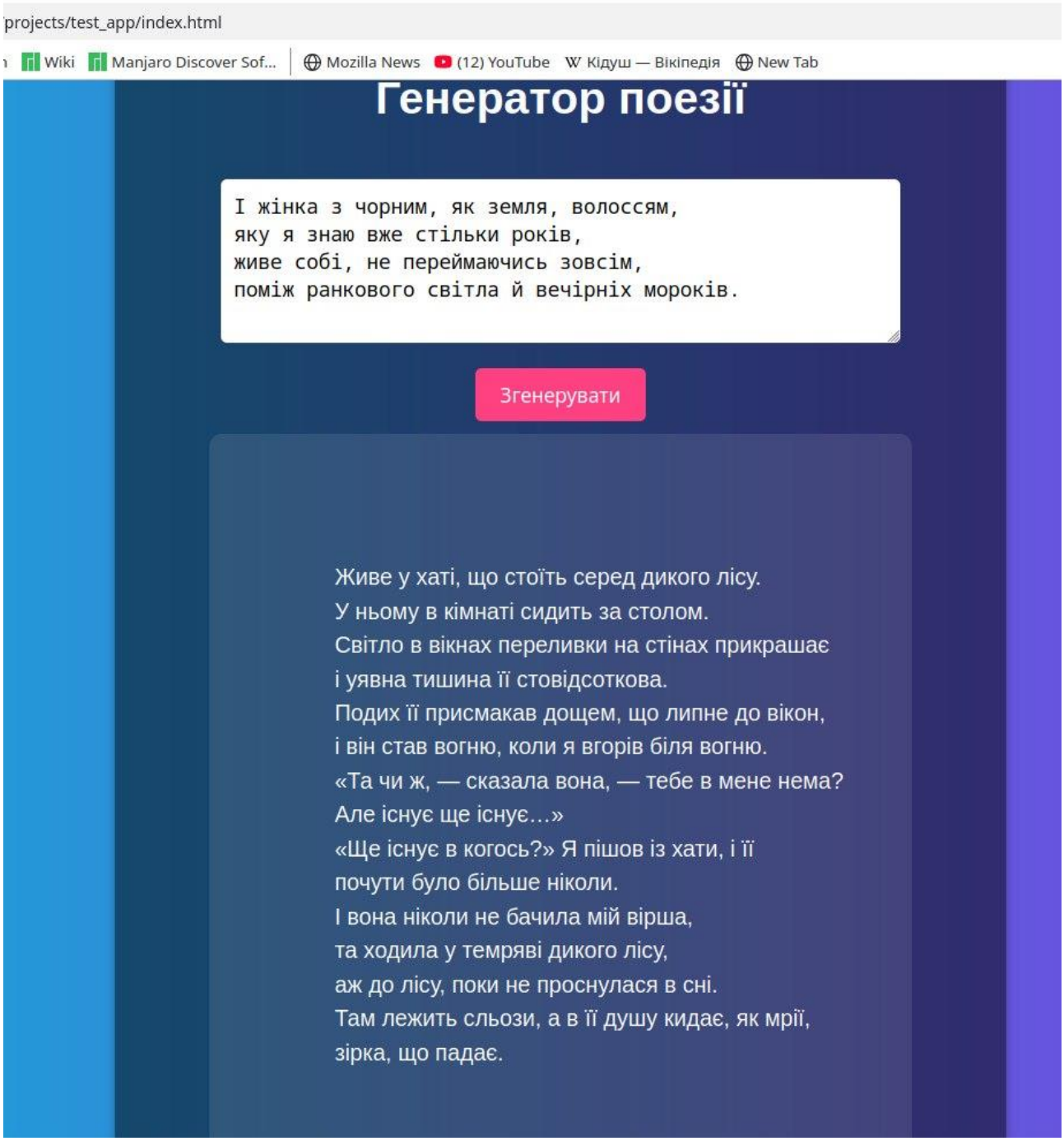

Рисунок 4.9 – Отриманий результат генерації

ВИСНОВКИ

В ході виконання роботи, було проведено огляд найбільш поширених способів генерації тексту, зокрема поезії, методів на основі рекурентних нейронних мереж та трансформерів. Були оцінені їхні переваги та недоліки. Основна проблема рекурентних мереж — це затухання градієнтів під час роботи з довгими послідовностями. LSTM частково вирішує цю проблему завдяки механізму довготривалої пам'яті, що дозволяє зберігати контекст, який стандартні RNN втрачають через цю проблему. Однак навіть LSTM в порівнянні з трансформерними архітектурами демонструють обмежену ефективність при дуже довгих контекстах, через саму рекурентність як таку. Сигнал, що проходить через численні часові кроки, так чи інакше згасає, механізм воріт має обмежену здатність зберігати всю необхідну інформацію.

Трансформери, завдяки механізму уваги, долають ці недоліки, дозволяючи ефективно обробляти залежності між будь-якими елементами послідовності, незалежно від їхньої віддаленості один від одного, відповідно мають перевагу в кращій здатності утримувати контекст.Це досить вирішальний фактор для задач генерації творів, де важлива логічна послідовність думок на певних часових проміжках.

Врахувавши переваги та недоліки, вибір зупинився на донавчанні SOTA моделі генерації тексту Gemma 2B. Вона базується на архітектурі декодера трансформера, завдяки чому може обробляти довгі текстові послідовності та має значний обсяг контексту, що дає змогу зберігати тематику й стилістичні особливості тексту, що є ключовим для генерації зв’язної й структурованої поетичної форми. А також має оптимальний розмір для використання на безкоштовних обчислювальних ресурсах таких як Kaggle.

Як початкові точки для донавчання на віршах було обрано базову модель Gemma 2b та попередньо натреновану Gemma 2b на українських текстах , а

саме UAlpaca та наборі даних ЗНО.

Результатом донавчання цих варіантів Gemma 2B стали моделі, що продовжують генерувати поезії, задані першими рядками. Проте, донавчання попередньо навченої мало гірші результати як на валідаційній функції втрат, так і на якості генерації, вірші виглядають більш структуровано, з меншим хаосом в образах. Це пояснюється тим, що під час попереднього донавчання, модель могла забути загальні патерни, які були навчальні під час базового тренування, збільшивши акцент на цільові дані. Тому для подальшої роботи було обрано варіант з базовою моделю.

Модель демонструє здатність створювати подальший сюжет з тими об’єктами, що згадуються у вхідних рядках, проте часто генеруються неповні фрази, значення яких важко інтерпретувати. З цих причин інколи проявляється слабка звязність твору. Запропонований ряд метрик, щодо оцінки якості згенерованих віршів, показав, що порівнянні з набором оригінальних поем, деякій частці згенерованих притаманна менша попарна схожість та більше стандартне відхилення іменників – частин мови, для позначення основних сутностей та концепцій. Проте, у відібраних користувачем віршах дані метрики відповідали рівню поезій з датасету, проте проглядається слабка звязність, хоч слова чи фрази вписуються в межі основної теми.

Для демонстрації роботи моделі було розроблено веб додаток, для реалізації якого обрано фреймворк Flask, який працює на платформі Kaggle, а також використовуються тунель ngrok, що дозволяє обійти обмеження доступу до сервера, розгорнутого у сердовищі Kaggle.

Результати свідчать про необхідність подальших експерементів для вдосконалення системи генерації поезії.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I. Attention Is All You Need. arXiv:1706.03762, 2017. URL: https://arxiv.org/abs/1706.03762

2. Гущин І., Сич Д. Аналіз впливу попередньої обробки тексту на результати текстової класифікації. Молодий Вчений, 2018, № 10. С. 34–40.

3. MMD Alumni. Knewton Personalizes Learning with the Power of AI. Harvard University, 2021. URL: https://d3.harvard.edu/platform-digit/submission/knewton-personalizes-learning-with-the-power-of-ai/

4. Jin, M., Shu, D., & Yu, Q. Health-LLM: Personalized Retrieval-Augmented Disease Prediction System. arXiv 2402.000746v7, 2024. URL: https://arxiv.org/abs/2402.000746

5. Goodfellow, I., Bengio, Y., & Courville, A. Deep Learning. MIT Press, 2016.

6. Руденко, О. Г., & Бодянський, Є. В. Штучні нейронні мережі. Харків: Компанія СМІТ, 2006. 117 с.

7. Jurafsky, D., & Martin, J. H. Speech and Language Processing (3rd ed., draft). January 12, 2025.

8. Tunstall, V., Leandro, L., & Wolf, T. Natural Language Processing with Transformers (Vol. 2). 2022. 96-134 с.

9. Писаренко С. В. Метод ідентифікації схожих людей неінтрузивним способом - розпізнаванням вуха та обличчя. Здобутки та досягнення прикладних та фундаментальних наук XXI століття. VI Міжнародна наукова конференція, 8 грудня 2023 р. – Черкаси - С. 264-266.

10. Alammar, J. Hands-On Large Language Models: Language Understanding and Generation. O’Reilly Media, 2024. ISBN 9781098150969.

11. Hu, E. J., Shen, D., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, L., Wang, S.,

& Chen, W. LoRA: Low-Rank Adaptation of Large Language Models. arXiv. URL: https://arxiv.org/abs/2106.09685

12. Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C., Kandola, J., Hofmann, T., Poggio, T., & Shawe-Taylor, J. A Neural Probabilistic Language Model. Technical Report, 2003.

13. Grinberg, M. Flask Web Development. First Edition. O'Reilly Media, Inc., 2014.

14. Kiulian, A., Polishko, A., Khandoga, M., Chubych, O., Connor, J., Ravishankar, R., & Shirawalmath, A. From Bytes to Borsch: Fine-Tuning Gemma and Mistral for the Ukrainian Language Representation. arXiv. URL: https://doi.org/10.48550/arXiv.2404.09138

15. Lo, K., & Ariss, K. GPoeT-2: A GPT-2 Based Poem Generator. arXiv 2205.08847, 2022. URL: https://arxiv.org/abs/2205.08847

16. Радомська, Л. А., & Азарова, Л. Є. Семантичні відношення в термінологічних іменниках: монографія. Вінниця: ВНТУ, 2016.