

Міністерство освіти і науки України  
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук  
(повна назва)

Кафедра Системотехніки  
(повна назва)

## КВАЛІФІКАЦІЙНА РОБОТА Пояснювальна записка

другий (магістерський)  
(освітньо-кваліфікаційний рівень)

Розробка системи обробки природної мови в інтерактивних діалогах та сюжетах комп'ютерних ігор на основі нейронних мереж  
(тема роботи)

Виконав: студент IV курсу, групи ITПм-22-2  
спеціальність 122 – «Комп'ютерні науки»  
тип програми освітньо-професійна  
освітня програма Інформаційні технології  
проективання

(шифр і назва спеціальності)

Ігошин Д.В.

(прізвище, ініціали)

Керівник доц. Решетнік В.М.  
(прізвище, ініціали)

Допускається до захисту

Зав. кафедри СТ

(підпис)

проф. Гребеннік І.В.

(прізвище, ініціали)

2024 р.

Я, як студент ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

«16» січня 2024 р.

Ігошин Д.В.

Кваліфікаційна робота формлена у відповідності до вимог діючих стандартів та методичних вказівок.

Матеріали кваліфікаційної роботи не містять відомостей, що заборонені для опублікування у відкритих виданнях.

Попередній захист проведено 16.01.24.

Керівник кваліфікаційної роботи



Решетнік В.М.

Факультет комп'ютерних наук

Кафедра системотехніки

Рівень вищої освіти другий (магістерський)

Спеціальність 122 «Комп'ютерні науки»

Тип програми освітньо-професійна

Освітня програма Інформаційні технології проектування

ЗАТВЕРДЖУЮ:

Зав. кафедри СТ

проф. Гребеннік І.В.

" " 2024 р.

## ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Студентові Ігошину Данилу Валерійовичу

(прізвище, ім'я, по батькові)

1. **Тема роботи** Розробка системи обробки природної мови в інтерактивних діалогах та сюжетах комп'ютерних ігор на основі нейронних мереж  
затверджена наказом по університету від " 20 " листопада 2023р. № 1373СТ

2. **Термін подання студентом роботи** 18 січня 2024 р.

3. **Вихідні дані до роботи (проекту)** Перелік використовуваних програмних засобів: ОС Microsoft Windows v.10. Технічне забезпечення: IBM-сумісний ПК з процесором Core i5 10 TH GEN. Інтегроване середовище розробки PyCharm Community Edition 2022.3.3 та Google Cloud

4. **Зміст пояснювальної записки (перелік питань, що потрібно розробити)**
- 4.1 Вступ. 4.2 Аналіз проблемної області. 4.2.1. Аналіз предметної області, яка визначає розробку та використання нейронних мереж(НМ) з обробкою природної мови. 4.2.1.1. Загальні відомості про нейронні мережі з можливістю обробки природної мови (NLP). 4.2.2. Аналіз існуючих НМ генеративного типу для створення моделі інтерактивних діалогів та сюжетів. 4.2.3. Аналіз існуючих рішень у сфері інтерактивних сюжетів у іграх. 4.2.4. Порівняння різних моделей та підходів для створення моделі нейронної мережі інтерактивних сюжетів та діалогів. 4.2.5. Визначення та принцип функціонування моделі «Трансформера». 4.2.6. Аналіз основних викликів та проблем при розробці системи інтерактивних сюжетів. 4.2.7. Постановка завдання. 4.3. Опис прийнятих проектних рішень 4.3.1. Обґрунтування мови програмування для розробки нейронної мережі. 4.3.2. Стек бібліотек для розробки та навчання нейронної мережі. 4.3.3. Обґрунтування архітектури для розробки нейронної мережі. 4.3.4. Обґрунтування етапів розробки нейронної мережі. 4.3.5. Розробка спрощеної моделі 4.3.6. Розробка з використанням переднавченої моделі. 4.3.7. Розробка моделі нейронної мережі з використанням можливостей хмарного сервісу Google Cloud. 4.3.7.1. Обґрунтування використання хмарного сервісу Google Cloud 4.3.7.2. Налаштування сервісу для використання та розробки моделі 4.3.8. Розробка інтерфейсу для взаємодії з нейронною мережею. 4.3.9. Тестування якості згенерованого тексту системи інтерактивних сюжетів та діалогів. 4.3.10. Висновки по результатам розробки. 4.4. Оптимізація та вдосконалення моделі НМ. 4.4.1. Аналіз результатів навчання та ефективності моделі. 4.4.2. Оцінка якості моделі інтерактивних сюжетів та діалогів. 4.4.2.1. Розрахунок перплексії моделі. 4.4.2.2. Розрахунок BLEU. 4.4.2.3. Візуалізація attention maps (карти уваги). 4.4.2.4. Тестування оцінки ефективності використання обчислювальних ресурсів. 4.4.3. Аналіз методів оптимізації гіперпараметрів нейронної мережі на основі архітектури трансформера. 4.4.4. Оптимізація гіперпараметрів та моніторинг ефективності моделі нм. 4.4.5. Висновки за результатами оптимізації гіперпараметрів та вдосконалення моделі нейронної мережі 4.5. Висновки. 4.6. Перелік джерел посилання.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, плакатів) 5.1 Постановка задачі (1 аркуш формату А4).5.2 Діаграма архітектури desktop застосунку зі спрощеною моделлю нейронної мережі 5.3 Діаграма архітектури desktop застосунку з використанням переднавченої мережі «GPT-2» та використанням хмарного середовища Google Cloud Console

6. Дата видачі завдання: 20.10.2024 р.

### КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів атестаційної роботи	Термін виконання етапів роботи	Примітка
1.	<i>Отримання завдання атестаційної роботи</i>	20.10.2023	Виконано
2.	<i>Аналіз завдання, літератури та аналогів з теми атестаційної роботи</i>	20.10 - 1.11.2023	Виконано
3.	<i>Проектування та уточнення вимог до моделі нейронної мережі, етапів розробки та навчання</i>	1.11 - 10.11.2023	Виконано
4.	<i>Вибір середовища розробки програми, стеку технологій та архітектури моделі</i>	10.11 - 13.11.2023	Виконано
5.	<i>Розробка та навчання нейронної мережі</i>	13.11 – 29.11.2023	Виконано
6.	<i>Аналіз ефективності та методів оптимізації гіперпараметрів</i>	29.11 – 06.12.2023	Виконано
7.	<i>Оптимізація та вдосконалення нейронної мережі, моніторинг ефективності та порівняння результатів</i>	06.12 – 28.12.2023	Виконано
8.	<i>Оформлення пояснювальної записки та програмної документації</i>	28.12 – 08.01.2024	Виконано
9.	<i>Представлення на рецензування</i>	13.01.2024	Виконано
10.	<i>Попередній захист</i>	15.01.2024	Виконано
11.	<i>Представлення атестаційної роботи в ДЕК</i>	18.01.2024	Виконано

Студент \_\_\_\_\_

Ігошин Д.В. \_\_\_\_\_

(підпис)

Керівник роботи \_\_\_\_\_



доцент Решетнік В.М. \_\_\_\_\_

## РЕФЕРАТ

Кваліфікаційна робота містить: 133 с., 10 табл., 41 рис., 3 додатка, 55 джерел інформації, 13 лістингів.

НЕЙРОННА МЕРЕЖА, ІНТЕРАКТИВНИЙ СЮЖЕТ, ГЕНЕРАЦІЯ ТЕКСТУ, ОБРОБКА ПРИРОДНОЇ МОВИ, ТРАНСФОРМЕР, GPT, HUGGING FACE, TKINTER, PYTHON, GOOGLE CLOUD, NLP, API, КОПМ'ЮТЕРНІ ІГРИ

Об'єктом розробки є процес генерації інтерактивних сюжетів та діалогів за допомогою нейронних мереж.

Предмет розробки – інформаційні технології та програмні методи для створення нейронної мережі з метою генерування інтерактивних сюжетів.

Мета роботи – розробка нейронної мережі для генерації інтерактивних сюжетів та діалогів.

Методи розробки – метод аналізу літератури, огляд реалізованих систем з підтримкою NLP та нейронних мереж у відеоіграх, метод аналізу нормативно-правової документації та етнічних норм, методи оцінювання ефективності та методи оптимізації та вдосконалення.

В результаті роботи проаналізовано предметну область, обрано архітектуру моделі нейронної мережі, розроблено та навчено нейронну мережу з використанням бібліотеки Hugging Face та хмарних технологій, оптимізовано за обраними та обґрунтованими критеріями, порівняно ефективність та зроблені висновки. Ця робота пов'язана з тезами на тему «розробка системи обробки природної мови в інтерактивних діалогах та сюжетах комп'ютерних ігор на основі нейронних мереж» [1].

Галузь застосування – використання у ігровій індустрії для генерації інтерактивних сюжетів та діалогів.

## ABSTRACT

The qualifying paper includes: 133 pages, 10 tables, 41 figures, 3 appendices, 55 information sources, 13 listings.

NEURAL NETWORK, INTERACTIVE PLOT, TEXT GENERATION, NATURAL LANGUAGE PROCESSING, TRANSFORMER, GPT, HUGGING FACE, TKINTER, PYTHON, GOOGLE CLOUD, NLP, API, COMPUTER GAMES

The development object is the process of generating interactive plots and dialogs using neural networks.

The development subject includes information technologies and software methods for creating a neural network aimed at generating interactive plots.

The work's goal is to develop a neural network for generating interactive plots and dialogs.

Development methods include literature analysis, review of implemented systems supporting NLP and neural networks in video games, analysis of regulatory documentation and ethnic norms, evaluation and optimization methods.

As a result of the work, the subject area was analyzed, the architecture of the neural network model was chosen, and a neural network was developed and trained using the Hugging Face library and cloud technologies. It was optimized according to selected and justified criteria, the effectiveness was compared, and conclusions were drawn.

This work is associated with the theses on the topic "development of natural language processing system in interactive dialogs and plots of computer games based on neural networks" [1].

Application field: Usage in the gaming industry for generating interactive plots and dialogs.

## ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, ОДИНИЦЬ І ТЕРМІНІВ.....	9
ВСТУП.....	10
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	12
1.1 Аналіз предметної області, яка визначає розробку та використання нейронних мереж (НМ) з обробкою природної мови .....	12
1.1.1 Загальні відомості про нейронні мережі з можливістю обробки природної мови (NLP) .....	13
1.2 Аналіз існуючих НМ генеративного типу для створення моделі інтерактивних діалогів та сюжетів .....	15
1.3 Аналіз існуючих рішень у сфері інтерактивних сюжетів у іграх.....	17
1.4 Порівняння різних моделей та підходів для створення моделі нейронної мережі інтерактивних сюжетів та діалогів .....	19
1.5 Визначення та принцип функціонування моделі «Трансформеру».....	26
1.6 Аналіз основних викликів та проблем при розробці системи інтерактивних сюжетів .....	28
1.7 Постановка завдання .....	29
2 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ .....	31
2.1 Обґрунтування мови програмування для розробки нейронної мережі	31
2.2 Стек бібліотек для розробки та навчання нейронної мережі.....	32
2.3 Обґрунтування архітектури для розробки нейронної мережі.....	33
2.4 Обґрунтування етапів розробки нейронної мережі .....	34
2.5 Розробка спрощеної моделі .....	35
2.6 Розробки з використанням переднавченої моделі .....	43
2.7 Розробка моделі нейронної мережі з використанням можливостей хмарного сервісу Google Cloud.....	47
2.7.1 Обґрунтування використання хмарного сервісу Google Cloud .....	48
2.7.2 Налаштування сервісу для використання та розробка моделі .....	49
2.8 Розробка інтерфейсу для взаємодії з нейронною мережею .....	56
2.8.1 Аналіз користувацьких потреб .....	56
2.8.2 Прототип інтерфейсу та кольорова палітра .....	56
2.8.3 Розробка інтерфейсу .....	58

2.9	Тестування якості згенерованого тексту системи інтерактивних сюжетів та діалогів.....	60
2.9.1	Критерії тестування якості згенерованого тексту .....	60
2.9.2	Тестування якості згенерованого тексту .....	62
2.10	Висновки по результатам розробки.....	64
3	Оптимізація та вдосконалення моделі НМ.....	65
3.1	Аналіз результатів навчання та ефективності моделі.....	65
3.2	Оцінка якості моделі інтерактивних сюжетів та діалогів .....	68
3.2.1	Розрахунок перплексії моделі.....	68
3.2.2	Розрахунок BLEU.....	71
3.2.3	Візуалізація attention maps (карти уваги).....	73
3.2.4	Тестування оцінки ефективності використання обчислювальних ресурсів.....	76
3.3	Аналіз методів оптимізації гіперпараметрів нейронної мережі на основі архітектури трансформера .....	77
3.3.1	Аналіз методів для оптимізації гіперпараметрів .....	77
3.3.2	Регуляція та уникнення перенавчання.....	79
3.3.3	Навчання на нових наборах даних .....	80
3.4	Оптимізація гіперпараметрів та моніторинг ефективності моделі нейронної мережі.....	81
3.4.1	Результати моніторингу ефективності після проведення оптимізації гіперпараметрів .....	86
3.5	Висновки за результатами оптимізації гіперпараметрів та вдосконалення моделі нейронної мережі .....	92
	ВИСНОВКИ.....	94
	ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	96
	Додаток А Посібник користувача.....	101
	Додаток Б Текст програми .....	109
	Додаток В Графічний матеріал кваліфікаційної роботи .....	125

## ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, ОДИНИЦЬ І ТЕРМІНІВ

API – програмний інтерфейс застосування (application programming interface)

BERT – двонаправлений кодувальник-розкодувальник трансформера (bidirectional encoder representations from transformers)

CNN – згорткова нейронна мережа (convolutional neural network)

GPT – генеративна переднавчена трансформерна модель (generative pre-trained transformer)

GRU – блок гейтів, що періодично поновлюються (gated recurrent unit)

GUI – графічний інтерфейс користувача (graphical user interface)

Hugging Face – бібліотека для роботи з переднавченими моделями NLP

LSTM – довга короткострокова пам'ять (long short-term memory)

NLP – обробка природної мови (natural language processing)

ReLU – випрямляюча лінійна функція активації (rectified linear unit)

RNN – рекурентна нейронна мережа (recurrent neural network)

НМ – нейронна мережа

ШІ – штучний інтелект

## ВСТУП

Розвиток технологій штучного інтелекту відкриває нові горизонти для людства. З'являються нові нейронні мережі (НМ), які здатні аналізувати зображення, генерувати текст та виконувати класифікацію. Нейронні мережі використовуються в маркетингу, медицині, кіно та ігровій індустрії, тощо. Використання алгоритмів для створення цікавих сюжетів та діалогів у комп'ютерних іграх є інноваційним та здатним перевернути індустрію розробки ігор.

Актуальність теми розробки нейронної мережі для генерації інтерактивних сюжетів та діалогів обумовлена наступними чинниками:

- використання нейронних мереж спрощує розробку діалогів та сюжетів, дозволяє створювати динамічний контент та надає можливість грі адаптуватись до дій користувача. Розробка такої спеціалізованої моделі є важкою та актуальною задачею;

- існуючі нейронні мережі не в повній мірі задовольняють виконання завдання створення інтерактивних сюжетів та діалогів, які були б логічні та не втрачали структури від дій користувача. Розвиток такої моделі може значно прискорити розробку ігор та покращити якість контенту;

- спеціалізована нейронна мережа для генерації інтерактивних сюжетів та діалогів має великий потенціал комерційного застосування у сферах освіти, маркетингу, кіно. Нейронні мережі для створення сюжетів та діалогів можуть стати актуальним продуктом.

Отже, виходячи з аналізу чинників, тема є актуальною в контексті розробки штучного інтелекту та сучасних тенденцій розвитку. Актуальність теми підтверджують тенденції попиту на розробку та використання нейронних мереж на даний час.

Об'єктом розробки є процес генерація сюжетів та діалогів з використання нейронної мережі.

Предмет розробки – це методи, алгоритми розробки та навчання моделі нейронної мережі, яка здатна генерувати інтерактивні сюжети та діалоги.

Метою роботи є розробка спеціалізованої моделі нейронної мережі на основі архітектури трансформера та переднавченої моделі «GPT-2» для генерації інтерактивних сюжетів та діалогів.

Для досягнення цілей розробки, поставлені наступні завдання:

- проаналізувати предметну область, яка регламентує розробку нейронних мереж. Визначити існуючі методи та підходи, проаналізувати типи архітектури штучних мереж, які підходять для генерації тексту та інтерактивної взаємодії з користувачем;

- обґрунтувати обрану архітектуру та етапи розробки моделі;

- виконати комплексну розробку та навчання моделі;

- проаналізувати потреби користувача та розробити інтерфейс для взаємодії з НМ;

- виконати тестування якості згенерованого контексту, проаналізувати якість моделі за допомогою методів перплексії, BLEU, карт уваги (attention maps) та використання обчислювальних ресурсів;

- провести аналіз можливих методів оптимізації, обґрунтування обраного методу та виконати оптимізацію та вдосконалення;

- проаналізувати результати отримані до виконання оптимізації та після, зробити висновки та надати рекомендації.

Результати досліджень розробки можуть знайти використання у створенні більш досконалих нейронних мереж для генерації інтерактивних сюжетів та діалогів у таких сферах, як маркетинг, ігрова індустрія та кіноіндустрія.

## 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області, яка визначає розробку та використання нейронних мереж (НМ) з обробкою природної мови

На даний час у світі набувають великого розвитку та фінансування різні за призначення та дії нейронні мережі. Зараз, як ніколи, штучний інтелект стає доступним як для великих корпорацій з суперкомп'ютерами, так і для звичайних людей. Вже сьогодні існують такі системи, в які вбудовано штучний інтелект – це чат-боти різних установ, голосові помічники, генератори зображень та тексту та класифікатори. Ігрова індустрія розвивається доволі стрімко, та генерація випадкових рівнів за певними правилами – це звичайне діло. Такі ігри називаються рогаликами. Але використання саме нейронних мереж для створення рівнів, сюжетів чи діалогів ще не є розповсюдженим явищем.

Існують такі експериментальні ігри, які реалізують ідею додавання нейронної мережі до гри, але вони не в повній мірі відображають ідею генерації та створення інтерактивності. Наступні представлені ігри включають деякий функціонал:

– ai Dungeon – це текстова гра, яка використовує модель «GPT» для створення пригод у жанрі настільної гри «підземелля та дракони, D&D». В даній грі користувачі можуть взаємодіяти зі штучним інтелектом (ШІ) та вибирати з дій, які він пропонує;

– rpgm week є експериментальною грою в якій НМ дозволяє персонажам взаємодіяти у грі між собою та приймати рішення, засновані на їх характерах та стосунках один з одним. Але користувач тут не може грати, а лише створювати персонажів та їх характер та дивитися за їх діями у грі.

Ідея кваліфікаційної роботи полягає у розробці спеціалізованої моделі НМ, яку можна використовувати за допомогою API або окремо для розроблення сюжету та продуманих інтерактивних діалогів. Отже, обрана тема є досить актуальною та інноваційною, зокрема для країн СНГ.

### 1.1.1 Загальні відомості про нейронні мережі з можливістю обробки природної мови (NLP)

Нейронні мережі в теперішній час займають великий сегмент у комп'ютерних науках. Вони мають швидку динаміку розвитку, але є складними в розробці та потребують великої кількості знань та обчислювальних ресурсів. Нейронна мережа – це комп'ютерна система, яка моделює структуру та функції взаємодії біологічних нейронів людського мозку. Вона складається зі штучних нейронів, які взаємодіють між собою через зв'язки, аналогічні нашим синапсам у мозку. Штучні мережі навчаються та регулюються з урахуванням соціальних та етичних норм.

Нейрони в штучних мережах, хоча і схожі на людські, але не є подібними. Штучні нейрони – це математичні обчислювальні одиниці, які можуть приймати сигнали, обробляти їх та видавати вихідні значення. Їх використовують для вирішення багатьох задач машинного навчання шляхом навчання на великих об'ємах даних (датасетах).

Вчені та розробники використовують спрощену модель людського нейрона, щоб створювати штучні мережі та використовувати їх для обробки великої кількості даних. Основна ідея полягає у передачі сигналів, обчислених ваг та функцій активації, але це все відбувається у математичному просторі. Цей простір не має схожості з біологічними процесами людини [2].

Людський нейрон виконує багато біологічних функцій та взаємодій, які дуже важно відобразити у штучному нейроні. Рисунок 1.1 ілюструє будову штучного нейрона, який складається з кількох компонентів:

- входи – це те, через що нейрон отримує вхідні дані від інших нейронів у мережі;
- ваги, кожна стрілка на схемі має вагу, яка визначає, наскільки сильно впливає вхід на вихід нейрона;
- сума: ваги множаться на вхідні дані, а вже потім підсумовуються;

- функція активації: отримана сума подається на функцію активації, яка визначає, що буде на виході нейрону;
- вихід: вихід нейрону передається до інших нейронів мережі.

В даному прикладі у нейрона є три входи, два ваги та функція активації, яка ще називається логічною функцією. Вона повертає значення від 0 до 1, що можна інтерпретувати як ймовірність того, що нейрон активується [3].

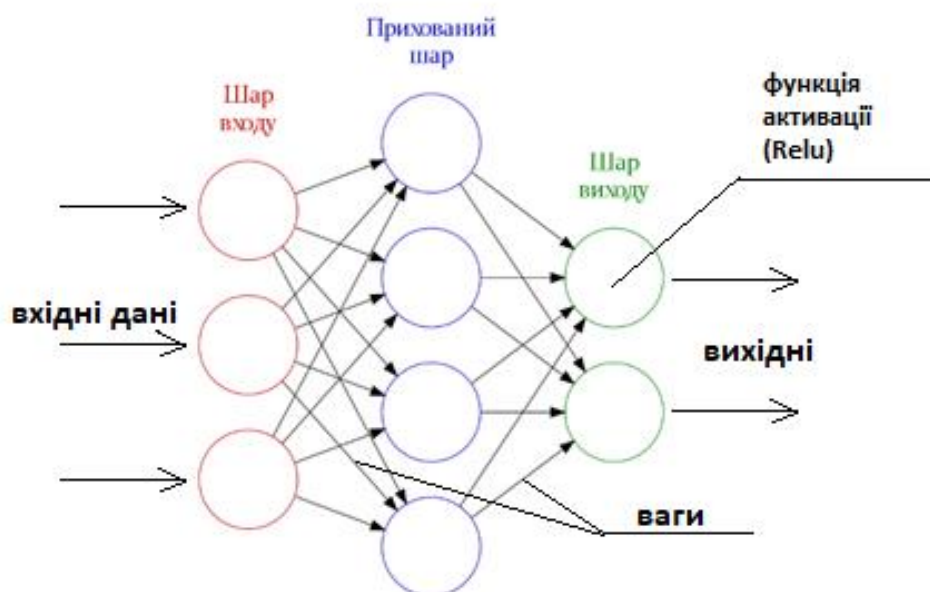


Рисунок 1.1 – Будова штучного нейрону в НМ

Важливий компонент у нейронній мережі для взаємодії з користувачем на його рідній мові є обробка природної мови (NLP). Обробка природної мови – це галузь штучного інтелекту, яка використовується для розробки алгоритмів та моделей для обробки, які здатні до аналізу, розуміння людської мови у текстовій чи аудіовізуальній формі [4]. Основна мета обробки природної мови – це створення систем, які здатні взаємодіяти з людиною на природній для неї мові. Також у перелік завдань входить: автоматичний переклад, генерація тексту, сентиментальний аналіз та багато іншого. Для створення моделі, яка може генерувати інтерактивний текст спираючись на дії користувача необхідно

розібратися у нейронних мережах генеративного типу. Таких мереж є велика кількість, вони мають різне застосування, переваги та недоліки.

## 1.2 Аналіз існуючих НМ генеративного типу для створення моделі інтерактивних діалогів та сюжетів

Використання нейронних мереж, які використовують обробку природної мови розширило можливості взаємодії людини зі штучними мережами. Зокрема, глибокі нейронні мережі, є ключовою технологією, яка сприяє покращенню якості та продуктивності систем NLP. Вони знаходять широке застосування в різних сферах нашого життя. Перелік аспектів штучних мереж, які активно використовуються, включає в себе [5]:

- глибокі нейронні мережі, такі як рекурентні (RNN) та згорткові (CNN), дозволяють моделювати складні залежності між словами та фразами у тексті. Вони мають можливості виявлення синтаксичних та семантичних структур у тексті, що є важливим у створенні інтерактивності;

- дозволяють виявляти емоційне забарвлення тексту, що важливо для створення яскравих героїв та ігрових сюжетів, які взаємодіють з гравцями відповідно до настрою;

- можуть використовуватися для вивчення поведінки гравців та адаптувати ігровий сценарій або діалог відповідно до поведінки, уподобань та стилю гри користувачів;

- можна використовувати для створення більш природних та контекстна-залежних діалогів між гравцем та віртуальним персонажем, що в свою чергу підвищить рівень взаємодії та зацікавленості;

- використовують для розпізнавання логічних зв'язків, що може бути корисним при створенні складних ігрових завдань та головоломок.

Однією з найбільш продуктивних та досконалих розробок на сучасному етапі, яка використовує NLP, є GPT-3.5, яку розробила компанія OpenAI. Ця нейронна мережа складається з 175 мільярдів параметрів та здатна генерувати

природні тексти, відповіді та діалоги. Проте вона є комерційною, і доступ до неї є обмеженим. Застосування GPT-3.5 представлено на рисунку 1.2.

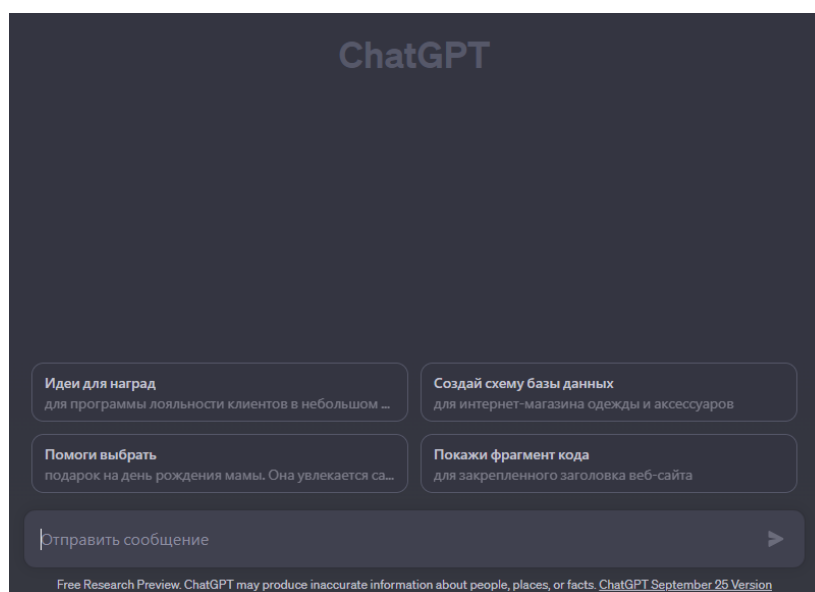


Рисунок 1.2 – Застосунок для взаємодії з Chat GPT-3.5

Dialogflow від Google і Language Understanding (LUIS) від Microsoft – це платформи, які дозволяють розробляти чат-ботів та системи обробки природної мови для взаємодії з користувачами. Вони надають інструменти для розпізнавання мови, аналізу і реагування на користувацькі запити [6].

Amazon Lex – платформа, інтегрована з сервісами AWS (Amazon Web Services), надає можливості для розробки чат-ботів з голосовою взаємодією та систем обробки природної мови [7].

Wit.ai, що належить Facebook, дозволяє створювати додатки та ігри з використанням мовних запитів та є безкоштовною [8].

Rasa – відкрита платформа для розробки чат-ботів та систем обробки мови. Вона надає можливість структурно за допомогою правил створювати розумних чат-ботів [9].

Ці платформи та програмні рішення допомагають розробникам впроваджувати NLP-функціонал у власних інтерактивних додатках. Вони забезпечують різні рівні складності, інтеграцію з голосовими та текстовими інтерфейсами, а також можливість навчання моделей для кращого розпізнавання

мови. У таблиці 1.1 представлено порівняння різних програмних рішень та платформ обробки природної мови. Порівняльну таблицю складено з використанням інформації з сайті технологій та відсортовано за параметрами.

Таблиця 1.1 – Порівняння програмних рішень та платформ з обробкою природної мови

Критерій	OpenAI GPT-3	Dialogflow та LUIS	Amazon Lex	Wit.ai	Rasa
Підтримка мов	Багато	Багато	Обмежена	Багато	Багато
Величина моделі	175 млрд параметрів	Залежить від плану	Не вказана	Не вказана	Власна модель
Гнучкість	Обмежена	Задовольняє	Задовольняє	Задовольняє	Задовольняє
Інтеграція з AWS / Azure	Ні	Так	Так	Ні	Ні
Вартість	Висока	Плата за використання	Плата за використання	Безкоштовно	Безкоштовно

### 1.3 Аналіз існуючих рішень у сфері інтерактивних сюжетів у іграх

Ігрова індустрія стрімко розвивається, вносячи інновації у галузь ігрового досвіду. Однією із цих інновацій є системи обробки природної мови, що дають гравцям можливість впливати на розвиток подій у грі. Аналіз спрямований на вивчення існуючих систем, їх переваг та обмежень, особливо у контексті створення моделі інтерактивних сюжетів та діалогів.

Відеоігри, які реалізують глобальну ідею інтерактивності діалогів та сюжетів не багато. На сьогоднішній день можна виділити таких представників:

– ai Dungeon – це гра у текстовому форматі, яка використовує GPT-3 для створення захопливих пригод. Користувачі можуть вводити свої дії та взаємодіяти з створеним світом, який генерує штучний інтелект. Перед початком створення інтерактивного сюжету, веб-застосунок пропонує обрати тип світу (фентезі, фантастика і тд.), вибрати параметри для початкового запиту, можна переглянути на рисунку 1.3. Надалі, після зроблених виборів про світ, роль, расу та введення

імені генерується історія, яка виведена на рисунку 1.4. Таким чином, можна продовжувати історію, вибираючи з запропонованих варіантів. Але недоліком є те, що не можна писати свої варіанти [10].



Рисунок 1.3 – Вибір параметрів для початкового запиту (промту)

Промт (prompt) – це набір текстових характеристик, котрі допоможуть нейронній мережі видати потрібний результат. Наприклад: «Згенеруй мені 12 ідей для назви шоколадного магазину, де продаються цукерки» [11].

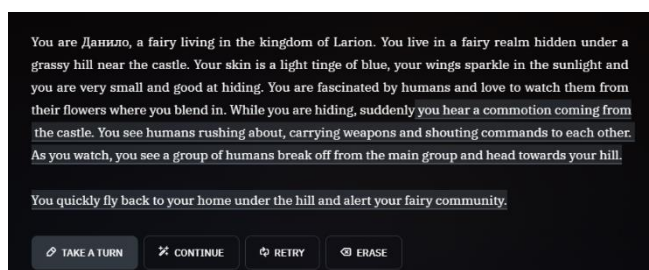


Рисунок 1.4 – Згенерована історія

– influence – це гра, в якій гравець бере на себе роль політика, який намагається отримати владу. Гра використовує нейронні мережі для створення діалогів, які реагують на дії гравця;

– tell Me Why – це гра, в якій гравець розслідує таємницю в маленькому містечку. Гра використовує NLP-технології для створення сюжету, який є більш правдоподібним та цікавим;

– life is Strange – гра, в якій гравець бере на себе роль дівчини, яка може подорожувати в часі. Гра використовує нейронні мережі для створення діалогів, які реагують на дії гравця [12].

Ігри з використанням нейронних мереж та обробкою природної мови мають високий потенціал розвитку. Такі ігри є більш реалістичними, захопливими та інтерактивними. Ймовірно у майбутньому ігри з використанням нейронних мереж стануть популярними. Ці ігри пропонують гравцям унікальний досвід, які відразу доразу буде різним в залежності від дій гравця. У таблиці 1.2 представлені переваги ігор з використанням нейронних мереж з обробкою природної мови.

Таблиця 1.2 – Переваги ігор з інтерактивними історіями на основі нейронних мереж та NLP

Перевага	Опис
Реалізм	Технології машинного навчання дозволяють створювати діалоги та сюжети, які є більш реалістичними та цікавими.
Інтерактивність	Гравці можуть приймати рішення, які впливають на розвиток сюжету, що робить ігри більш інтерактивними.
Особистісне занурення	Технології машинного навчання дозволяють створювати ігри, які можуть бути більш особисто захоплюючими для гравців.

#### 1.4 Порівняння різних моделей та підходів для створення моделі нейронної мережі інтерактивних сюжетів та діалогів

Завдання розробки нейронної мережі з обробкою природної мови у контексті інтерактивних діалогів та сюжетів є складним завданням. Спочатку необхідно розібратися з традиційними методами обробки природної мови за допомогою: правил, структурних мовних аналізаторів, стемінгу та лематизації та інших.

Правила (в контексті мовознавства) – це програмні засоби, які використовуються для розділення (розбору) мовних конструкцій з метою визначення семантики та граматики. Аналіз виконується на рівні слів, фраз та

речень, між якими виявляють зв'язки та визначають граматичну роль і семантичні властивості.

Стемінг – це процес відокремлення кореня (стеми) слова шляхом відокремлення афіксів, суфіксів та закінчень. Основна мета стемінгу полягає в полегшенні пошуку інших слів з однаковим коренем. Цей процес використовується для покращення аналізу, а також індексації інформації [13].

Лематизація – це процес відокремлення леми слова, або його нормалізованої словникової форми. Лема є базовою формою слова, і вона враховує морфологічні, граматичні та семантичні відмінності між словами [13].

На даний час розроблено та вдосконалено багато видів архітектури нейронних мереж, які підходять для виконання завдання генерації інтерактивного тексту:

1. Повністю зв'язані нейронні мережі (Fully Connected Neural Networks) – це тип штучної мережі, який є базовим. Такі мережі використовуються для аналізу послідовностей слів та символів. Однак, незважаючи на перевагу аналізу, вони можуть бути неефективними для вирішення завдань NLP, оскільки не враховують контексту та емоційного забарвлення. Рисунок 1.5 показує схему моделі "Fully Connected Neural Networks". На схемі зображено нейронну мережу з двома прихованими шарами та складається з наступних компонентів:

- вхідний шар (input layer), через цей шар нейронна мережа приймає вхідні дані, які представлені вектором  $x_0$ ;
- приховані шари обробляють вхідні дані за допомогою функції активації  $\sigma$ ;
- вихідний шар генерує вихідні дані, які представлені вектором  $y$ .

Зважені зв'язки між нейронами визначаються матрицею  $W$ , матриця має розмір  $(n + m) \times m$ , де  $n$  – кількість нейронів у вхідному шарі,  $m$  – кількість нейронів у прихованому шарі. Навчання мережі полягає в тому, щоб знайти значення матриці  $W$ , які мінімізують помилку між вхідними даними та бажаними результатами [14].

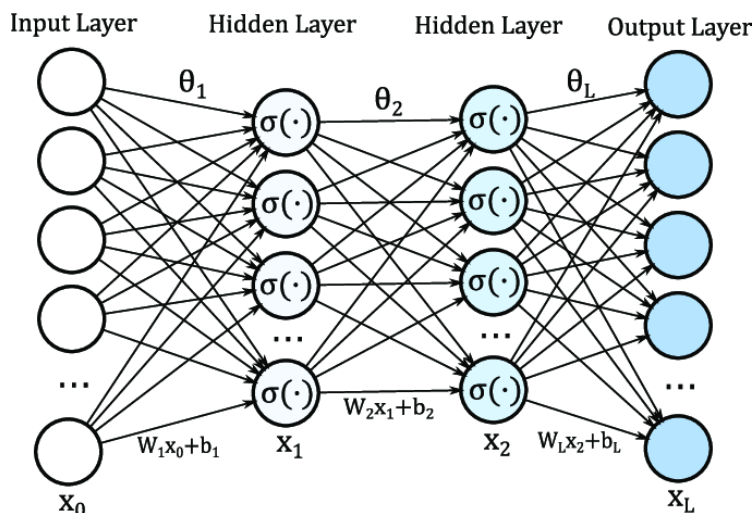


Рисунок 1.5 – Модель «Fully Connected Neural Networks»

2. Згорткові нейронні мережі (Convolutional Neural Networks, CNN) – це такий тип мереж, який використовується для обробки тексту на рівні фраз та слів. Мережа здатна виявляти залежності в тексті, відношення між словами в одному реченні, однак не здатні враховувати глобальний контекст та причинно-наслідковий зв'язок. Рисунок 1.6 ілюструє модель «CNN». На цій схемі мережа складається з наступних компонентів:

- вхідний шар для прийняття даних, в даному разі це зображення (Input image);
- згортковий шар виконує операцію згортки до кожного осередку вхідних даних (Convolution). Згортковий шар використовується для вилучення ознак з вхідних даних;
- операція згортання (pooling) об'єднує осередки вхідних даних у більші блоки. Операції згортання використовуються для зменшення розміру вихідних даних згорткових операцій;
- повторний рівень складається з чергування згорткових операцій і операцій згортання;
- вихідний шар генерує вихідні дані, які представлені вектором у [14, 15].

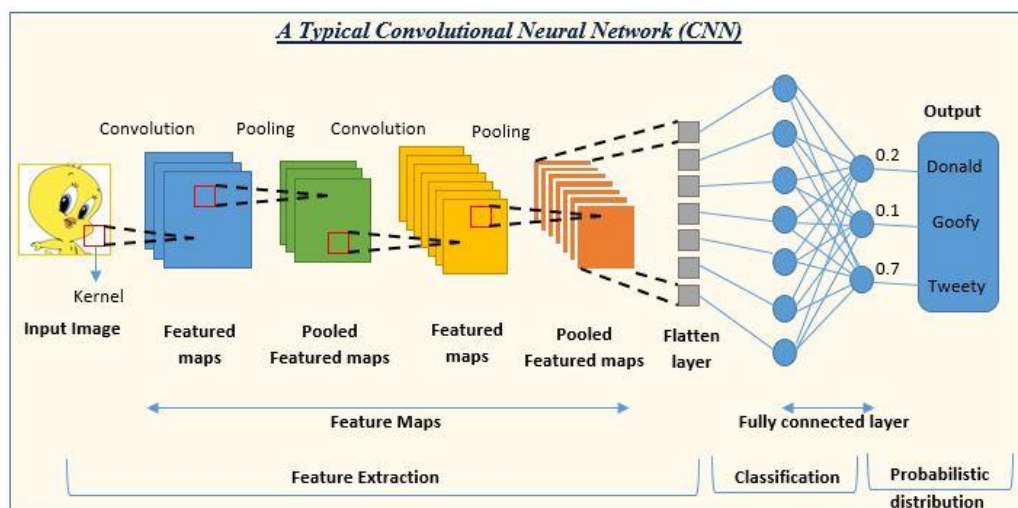


Рисунок 1.6 – Модель «Convolutional Neural Networks»

3. Рекурентні нейронні мережі (Recurrent Neural Networks, RNN) – тип мереж, які враховують послідовність даних та здатні до збереження станів для розуміння контексту. Добре підходять для генерації тексту, але можуть мати проблеми з довготривалими залежностями через проблему зникаючих/зростаючих градієнтів. На рисунку 1.7 зображено модель з рекурентним зв'язком. Рекурентна нейронна мережа (RNN) складається з наступних компонентів:

- вхідний шар (input layer) приймає вхідні дані, які представлені вектором  $x$ ;
- прихований шар (hidden layers) обробляє вхідні дані послідовно за допомогою функції активації  $\sigma$ . Це означає, що вхідні дані з попереднього часу впливають на обробку вхідних даних у поточний час;
- вихідний шар (output layer) генерує вихідні дані, які представлені вектором  $y$  [14, 15].

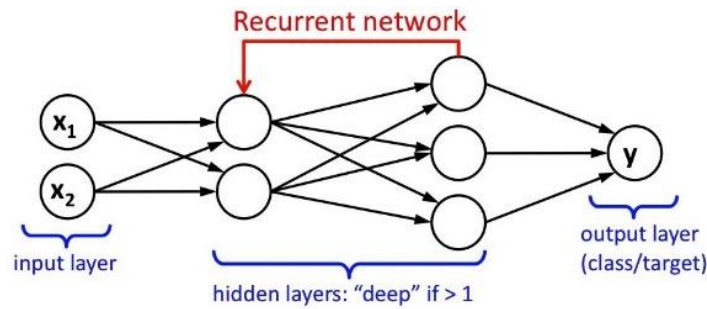


Рисунок 1.7 – Модель «Recurrent Neural Networks, RNN»

4. Мережа довгої короткострокової пам'яті (Long Short-Term Memory, LSTM) – це покращена версія RNN, яку вирішує проблему зникаючих/зростаючих градієнтів. Штучна мережа здатна враховувати довготривалі залежності у тексті та широко використовується у завданнях обробки природної мови. На рисунку 1.8 наведено модель LSTM. Модель LSTM складається з трьох основних компонентів:

- вхідний шлюз (Input Gate) визначає, які вхідні дані будуть дозволені до обробки. Якщо значення вхідного шлюзу менше 0, то вхідні дані будуть заблоковані;

- шлюз забуття (Forgotten Gate) визначає, які вхідні дані будуть забуті. Шлюз забуття приймає вхідні дані і вихідний стан попереднього часу. Він використовує ці дані для обчислення значення, яке визначає, які вхідні дані будуть забуті. Якщо значення забутого шлюзу більше 0, то попередній стан буде оновлений з урахуванням вхідних даних;

- вихідний шлюз (Output Gate) визначає, які результати будуть виведені [15].

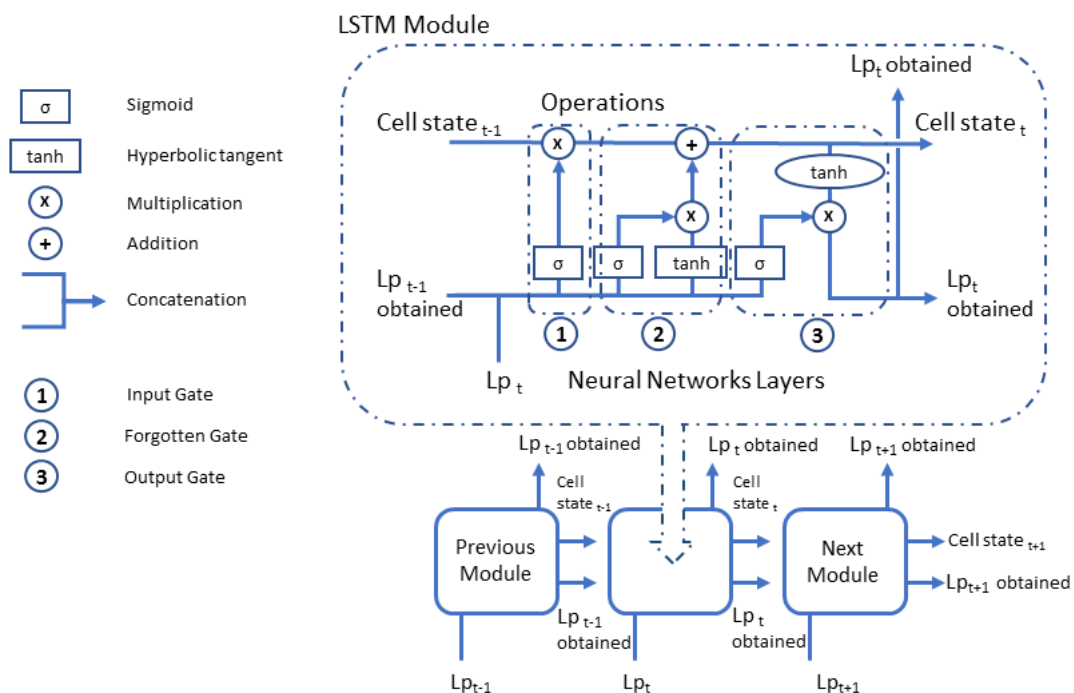


Рисунок 1.8 – Модель «Long Short-Term Memory, LSTM»

5. Модель трансформера (Transformers) є ключовою у розвитку обробки природної мови. В основі цієї моделі лежить використання механізму уваги, який може аналізувати довгі послідовності з високою точністю. Моделі, як BERT, GPT та інші, засновані на цій архітектурі, дали значний стрибок у якості обробки природної мови. Рисунок 1.9 демонструє базову структуру моделі трансформера.

Трансформер складається з наступних компонентів:

- вхідний шар (Input Embedding) перетворює вхідні дані в вектори;
- модуль позиції (Positional Encoding) додає вхідній послідовності інформацію про позицію кожного елемента. Це дозволяє враховувати порядок елементів вхідної послідовності;
- шари багатоголової уваги (Multi-Head Attention) дозволяють взаємодіяти з усіма елементами вхідної послідовності. Це надає можливість враховувати контекст, в якому знаходиться кожний елемент вхідної послідовності;
- шари лінійного перетворення (Linear Layers) перетворюють вихідні дані шарів багатоголової уваги в бажаний формат [16].

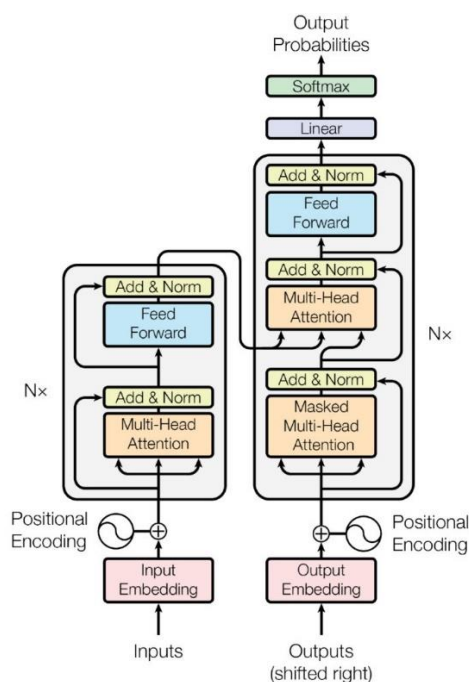


Рисунок 1.9 – Модель «Transformers»

Перелічені моделі нейронних мереж мають багато переваг та недоліків. Для кращого розуміння та виділення кожної моделі, яка потенційно може бути використана для розробки була створена порівняльна таблиця 1.3.

Таблиця 1.3 – Порівняння підходів/алгоритмів створення нейронних мереж

Підхід/Алгоритм	Опис	Переваги	Недоліки
1	2	3	4
Традиційні методи NLP	Використовують правила та структурні лінгвістичні аналізатори для обробки тексту.	Простота розробки, інтерпретованість	Обмежена в обробці складних завдань, вимагає багато правил та експертних знань
Fully Connected	Базові нейронні мережі, які використовуються для аналізу послідовностей слів та символів.	Простота, легка реалізація	Не враховують контекст та віддалені залежності між словами
Convolutional	Використовуються для обробки тексту на рівні фраз та слів. Виявляють локальні залежності.	Враховують локальний контекст, використовуються в синтаксичних завданнях	Не враховують глобальний контекст довших текстів
Recurrent NN (RNN)	Здатні враховувати послідовний контекст та зберігати попередні стани для аналізу тексту.	Враховують послідовний контекст, використовуються в генерації тексту	Проблема зникаючих/зростаючих градієнтів, не завжди ефективні
Long Short-Term	Покращені RNN,	Враховують	Складні у розробці, більше

Memory (LSTM)	розроблені для подолання проблеми градієнтів та збереження довготривалих залежностей.	послідовний та довготривалий контекст	обчислень
Трансформери	Моделі, які використовують механізм уваги для аналізу довгих текстових послідовностей.	Велика точність, ефективні в аналізі довгих текстів	Велика кількість параметрів, високі обчислювальні вимоги, складні у реалізації

Отже, узагальнюючи проаналізовану інформацію щодо моделей обробки природної мови та довготривалої пам'яті, можна зробити наступний висновок. Трансформери є ефективною архітектурою для виконання завдань розробки системи інтерактивних діалогів та сюжетів. Зокрема, моделі GPT-3 та BERT є ефективними для багатьох завдань NLP в цьому контексті завдяки їхній здатності аналізувати глобальний контекст та розрізняти важливі залежності в тексті, а також високій точності та ефективності в аналізі довгих запитів.

### 1.5 Визначення та принцип функціонування моделі «Трансформеру»

Для розробки власної нейронної мережі на основі архітектури трансформера необхідно детально проаналізувати її структуру та принципи функціонування.

Модель трансформера є однією з найбільш революційних у глибокому навчанні при обробці послідовностей даних, таких як зображення або текст. Архітектуру трансформера запропонували у 2017 році в статті «Attention Is All You Need» [16]. Ця робота привнесла механізм уваги (self-attention), що і стало революцією. До основних складових моделі трансформера відносяться:

- механізм уваги, котрий дозволяє моделі поділяти різним частинам вхідної інформації різну увагу при обчисленні ваги кожного токена. Це надає моделі змогу «фокусуватися» на важливих елементах у послідовності;

- множинна шарова архітектура. Модель має багато шарів, кожен з яких обробляє вхідні дані. У кожному шарі є дві підмережі: механізм уваги та повністю зв'язна мережа (позиційно-залежна нейронна мережа);

– додавання та нормалізація використовуються для стабілізації градієнтів під час навчання моделі та прискорення збіжності результатів;

– функція активації та шари виводу. Зазвичай використовується функція активації ReLU та повністю зв'язаний шар для виведення результатів.

Трансформер працює, обробляючи вхідні дані послідовно, що проілюстровано на рисунку 1.10, де представлено encoder та decoder [16].

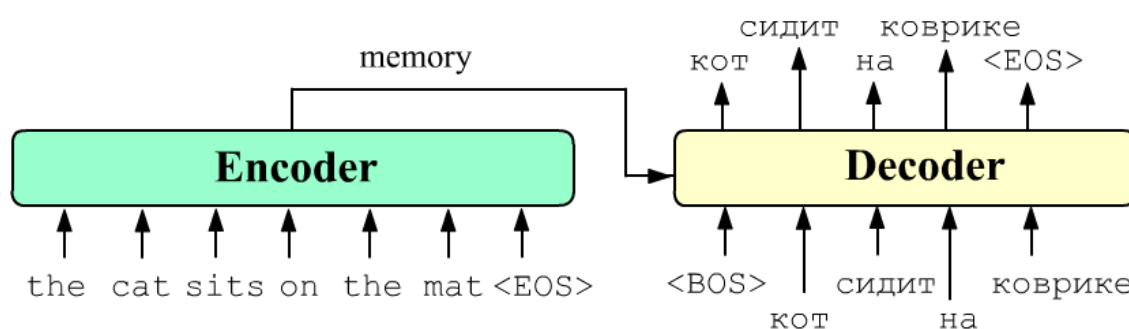


Рисунок 1.10 – Послідовна обробка даних

На кожному кроці використовуються вхідні дані, інформація про позицію та результати попереднього кроку для обчислення нових значень для шарів багатоголової уваги.

Шари багатоголової уваги дозволяють взаємодіяти з усіма елементами вхідної послідовності. Для цього використовуються дві матриці: матриця ключів і матриця значень. Матриця ключів представляє кожний елемент вхідної послідовності як ключ. Матриця значень представляє кожний елемент як значення. Шари багатоголової уваги обчислюють вагу кожного елемента вхідної послідовності для кожного іншого елемента вхідної послідовності. Вага елемента залежить від його ключа і значення. Ваги, які обчислюються шарами багатоголової уваги, дозволяють враховувати контекст, в якому знаходиться кожний елемент вхідної послідовності.

На основі аналізу моделі трансформера можна зробити висновок, що завдяки своїм компонентам модель може ефективно моделювати довгострокові залежності в послідовностях тексту, взаємодіяти з контекстом на різних рівнях та

забезпечувати глибоке розуміння отриманих даних. Ці переваги вирізняють модель трансформера серед інших типів архітектури.

## 1.6 Аналіз основних викликів та проблем при розробці системи інтерактивних сюжетів

Нейронні мережі з обробкою природної мови представляють собою актуальну та інноваційну галузь досліджень. Проте розробка таких систем стикається з рядом складних викликів та проблем, які потребують детального аналізу та розв'язку.

Однією з ключових проблем є різноманітність природної мови. Мова може мати багато можливих тлумачень, і розуміння правильного контексту вимагає врахування багатьох чинників. Нейронні мережі повинні бути здатні розрізняти семантику та контекст для ефективного діалогу з користувачем. Такі завдання потребують великої кількості даних для навчання нейронних мереж. Збір та підготовка великих корпусів тексту (колекція текстових документів для тренування та оцінки моделей NLP) відображають виклики в області анотування, якості даних і забезпечення конфіденційності.

Ігри та діалоги можуть бути дуже контекст-залежними та персоналізованими. Розробка систем, які можуть адаптуватися до різних сценаріїв та взаємодіяти з різними користувачами, є викликом. Відсутність зворотного зв'язку та оцінки якості є великою проблемою при вирішенні цього завдання. Визначення ефективності систем обробки природної мови у вірному контексті гри або діалогу може бути складним завданням. Важливо розробити метрики оцінки, які відображають якість взаємодії та задоволення користувачів.

Навчання великих нейронних мереж, з архітектурою трансформера, потребують значних обчислювальних ресурсів. По-перше, велика модель має багато параметрів, таких як ваги та зсуви для кожного шару мережі. Це дозволяє моделі адаптуватися до складних завдань і навчитися більше ознак, але при цьому збільшуються обчислювальні витрати на навчання.

Для ефективного навчання великих нейронних мереж необхідно мати достатньо обсягу тренувальних даних. Збирати і підготувати великі датасети може бути складним процесом і їх обробка потребує потужних обчислювальних ресурсів. Під час навчання важливо розраховувати градієнти функції втрат для коригування параметрів мережі. Цей процес може бути дуже витратним, особливо у великих моделях. Для прискорення навчання великих нейронних мереж часто застосовують паралельні обчислення на графічних процесорах (GPU) або тензорних процесорах (TPU). Це потребує доступу до багатьох обчислювальних вузлів та спеціалізованого обладнання [18].

Узагальнюючи, навчання моделі на архітектурі трансформера та інших великих нейронних мереж вимагає значних обчислювальних ресурсів через складність та розмір, а також через великі обсяги даних, які використовуються для навчання.

### 1.7 Постановка завдання

Виходячи з аналізу предметної області розробки нейронної мережі для генерації інтерактивних історій та діалогів, як і будь-яка інша нейронна мережа потребує комплексного підходу розробки та навчання з наступним корегуванням.

Мета розвитку та розробки нейронної мережі для генерації інтерактивних сюжетів та діалогів:

- з точки зору користувачів: полягає в створенні якісного та інтерактивного контенту. Користувачі очікують від такої технології можливості взаємодіяти з нейронною мережею яка розповідає історію через автора або персонажа. Основною метою є забезпечення неперервного інтерактивного досвіду, що надає користувачам можливість впливати на розвиток сюжету та відчувати, що рішення і дії мають вагомий роль у подіях;

- з точки зору ігрових розробників: для розробників ігор, мета розвитку полягає в створенні інструменту, який спростить процес створення ігрових досвідів. Вони шукають засоби автоматизації генерації контенту, зокрема діалогів

та сюжетних ліній, щоб швидше створювати ігри. Розробники також прагнуть до вдосконалення якості та інтеграції з віртуальними персонажами.

– з точки зору інновацій та розвитку нейронних мереж: з розширенням можливостей нейронних мереж для генерації інтерактивних історій та діалогів виникає можливість інновацій в галузі розваг та розвитку штучного інтелекту. Ця технологія може розширити можливості ігор та віртуальної реальності, дозволяючи створювати більш відкриті та несподівані ігрові сценарії. Також вона може бути використана в освітніх застосунках для створення навчальних інтерактивних історій та симуляцій. Такі симуляції допоможуть користувачам краще розуміти складні концепції.

Для досягнення мети розробки треба на основі проведеного аналізу предметної області та аналізу існуючих нейронних мереж виконати такі завдання:

– вибір архітектури моделі нейронної мережі: визначити, чи архітектура трансформера є найкращим варіантом для даної задачі, обґрунтувати вибір. Розглянути можливості адаптації цієї архітектури для створення інтерактивних історій та діалогів. Вибрати та обґрунтувати стек технологій для виконання завдання;

– вибір навчального набору даних: зібрати великий обсяг текстових даних (датасетів), які відображають сценарії історій та діалогів;

– обрати підхід розробки та навчання моделі: розробити спрощену модель, обрати переднавчену модель, виконати розробку та навчання моделі з використанням хмарних сервісів;

– моделювання інтеграції: розробити методи взаємодії користувача з системою на основі введення текстового вводу користувача та генерації відповідей, що відповідають діалогам та сюжетам;

– провести оцінювання якості та вдосконалення: призначити метрики для оцінки якості та ефективності системи в реальному часі. Використовувати ці метрики для постійного покращення роботи нейронної мережі та її взаємодії з користувачем.

## 2 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

### 2.1 Обґрунтування мови програмування для розробки нейронної мережі

Обирання мови програмування Python для створення нейронної мережі визначається кількома ключовими чинниками, які відображають його популярність та ефективність у сфері наукового дослідження та розробки в галузі штучного інтелекту. Цей вибір базується на таких аспектах:

- широкий спектр бібліотек та фреймворків: Python має велику кількість бібліотек необхідних для машинного навчання та глибокого навчання, таких як TensorFlow, PyTorch, та Keras, що надають потужні інструменти для реалізації та навчання нейронних мереж, зокрема з архітектурою трансформера;

- зручність та легкість використання: мова програмування відома читабельністю та простотою коду. Відсутність необхідності в декларації типів дозволяє зосередитися на розробці;

- активна спільнота: Python користується великою та активною спільнотою розробників, яка постійно роблять внески у вигляді нових інструментів та бібліотек, сприяючи постійному розвитку області машинного навчання;

- підтримка обчислювальних графічних процесорів (GPU): більшість фреймворків для глибокого навчання на Python мають підтримку використання GPU, що дозволяє значно прискорити обчислення при тренуванні великих нейронних мереж [19];

- переносний та інтегрований: Python є переносною мовою, що дозволяє запускати код на різних платформах. Це важливо для взаємодії з іншими науковими інструментами та розроблення моделей, які легко інтегруються в існуючі системи.

Загалом, Python – потужний інструмент для розробки систем штучного інтелекту. Його простота, багатий вибір бібліотек та фреймворків, платформо незалежність та активна спільнота розробників роблять його ідеальним вибором для розробки інтелектуальних систем.

## 2.2 Стек бібліотек для розробки та навчання нейронної мережі

Для розробки системи штучного інтелекту для генерації інтерактивних діалогів та сюжетів будуть використовуватися бібліотеки для створення та підтримки ШІ. Найбільш уживані бібліотеки проекту – це TensorFlow, Keras, hugging face, google console. Та бібліотеки для обчислень та створення інтерфейсу взаємодії з НМ: numpy, SciPy, Tkinter .

TensorFlow [20] – відкрита бібліотека програмування для розробки і виконання чисельних обчислень, зокрема машинного та глибокого навчання. TensorFlow названа так через свою спроможність виконувати обчислення, представлені у вигляді графів потоків даних, де дані переміщуються через різні вузли обчислення, нагадуючи "потоки" даних.

Keras [21] – це високорівневий інтерфейс для розробки та навчання нейронних мереж. Вона розроблена з урахуванням зручності і продуктивності користувача, що робить її дуже популярною у наукових та індустріальних дослідженнях у галузі штучного інтелекту. Keras надає простий інтерфейс для роботи з нейронними мережами та допомагає розробникам зосереджуватися на проектуванні та навчанні моделей, замість надмірних технічних деталей.

Hugging Face (Hugging Face Library) [22] – це інструмент для роботи з моделями машинного навчання в галузі обробки природної мови (Natural Language Processing, NLP). Цей інструмент є важливим ресурсом для дослідників та розробників у галузі штучного інтелекту та NLP. Ця бібліотека представляє готові моделі, з якими можна працювати використовуючи API, або завантажувати на ПК.

NumPy (Numerical Python) [23] — це бібліотека для мови програмування Python, призначена для роботи з великими масивами даних та виконання математичних операцій. Вона надає високопродуктивні масиви, а також інструменти для роботи з цими масивами, що робить її основною бібліотекою для наукових обчислень.

SciPy [24] — це бібліотека для виконання обчислень. Вона розширює можливості NumPy, додаючи багато функцій для оптимізації, обробки сигналів, алгебри лінійних рівнянь, інтегрування, оптимізації та інших наукових задач.

Google Cloud Console — це веб-інтерфейс для управління та моніторингу послуг Google Cloud Platform. Він надає інтерактивний доступ до різноманітних хмарних послуг, таких як обчислювальні ресурси, зберігання даних, штучний інтелект, аналіз даних та інші. Google Console дозволяє користувачам ефективно керувати своїми хмарними ресурсами та налаштовувати параметри сервісів [25].

### 2.3 Обґрунтування архітектури для розробки нейронної мережі

Розробка нейромереж для генерації інтерактивного сюжету складне завдання. Для цього потрібна модель, яка може обробляти природну мову та генерувати сюжет спираючись на промт (prompt) користувача. Промпт (prompt) – це набір характеристик, що дозволяють нейронним мережам генерувати очікуваний результат [26].

Серед архітектури штучних мереж доцільно розглядати LSTM (Long Short-Term Memory, довга короткострокова пам'ять) та Transformers (трансформер). Виходячи з аналізу моделей, які було проведено раніше. Порівнюємо ці дві моделі за такими критеріями: архітектура моделі, обробка послідовностей, масштабованість та швидкість навчання, залежність від довжини послідовності, загальна складність та кількість параметрів. У таблиці 2.1 наведено стислий порівняльний аналіз.

Таблиця 2.1 – Порівняння моделей LSTM та Трансформер

Критерій	LSTM	Трансформер
1	2	3
Архітектура моделі	Рекурентна	Паралельна
Обробка послідовностей	Послідовна	Паралельна
Масштабованість та швидкість навчання	Важко забезпечити масштабованість та швидкість залежить від розміру мережі та даних	Легко розширювати завдяки паралельному обчисленню та механізму уваги

## Продовження таблиці 2.1 – Порівняння моделей LSTM та Трансформер

1	2	3
Залежність від довжини послідовності	Проблеми з довгостроковою залежністю	Здатна ефективно працювати з довгими послідовностями завдяки механізму уваги
Загальна складність та кількість параметрів	Зазвичай менше параметрів, але менша складність	Велика кількість параметрів та більша складність, але більша ефективність завдяки паралельному обчисленню та механізму уваги

Отже, виходячи за аналізу двох моделей можна зробити висновок, що архітектура трансформера раціонально використовувати для виконання завдання генерування інтерактивних сюжетів. Дана модель не має обмежень на довжину послідовностей, має більше параметрів, орієнтована на масштабованість та паралельну обробку за допомогою механізму уваги. Недолік обчислювальної потужності можна розв'язати хмарними сервісами.

## 2.4 Обґрунтування етапів розробки нейронної мережі

У даному дослідженні, спрямованому на розробку нейронної мережі, визначено необхідність застосування систематичного та оптимального підходу. З цією метою, пропонується розглядати наступні етапи в процесі розробки:

– розробка спрощеної моделі GPT-2: розробка спрощеної моделі обумовлено обмеженими обчислювальними ресурсами та пам'яті на ПК. Незважаючи на ефективність спрощеної моделі, вона потребує великої кількості обчислювальних ресурсів, які не можуть бути досягнуті на ПК та не може виконати завдання якісної генерації на українській мові. При цьому, за допомогою спрощеної моделі можна здійснити перші кроки у розвитку та впровадженні прикладних рішень з обробки природної мови;

– використання переднавченої моделі GPT-2: виконавши перший етап, який надасть розуміння функціонування моделі, необхідно переходити на переднавчену для отримання та покращення результатів. Переднавчена модель в порівнянні з спрощеною має велику кількість параметрів та структуру, що надає можливість здійснювати більш точні та якісні прогнози. На жаль, переднавчена

модель хоча й має більш якісні прогнози потребує великої кількості обчислювальних ресурсів, які не можливо надати використовуючи персональний комп'ютер;

– використання хмарного сервісу Google Cloud для навчання та використання моделі з підтримкою API: останній етап включає перенесення моделі у сховище на хмарному сервісі, виконання навчання та оптимізації. Це рішення обумовлено наявністю обчислювальних ресурсів у хмарному сервісі, що значно полегшить та прискорить процес навчання та обробки текстів. Крім того, Google Cloud надає доступ до передових технологій у галузі обробки природної мови, що може призвести до покращення результатів.

Такий стратегічний та комплексний підхід надасть можливість не тільки зрозуміти та досягнути можливості розробки нейронної мережі, а й надасть можливість досягти значних результатів.

## 2.5 Розробка спрощеної моделі

Для розробки та впровадження системи генерування інтерактивного сюжету, розроблено спрощену модель. Код моделі взято з відкритих посилань компанії OpenAI [27]. Модель вважається спрощеною бо має у своїй структурі лише три основні класи, на відміну від звичайної моделі: MultiHeadAttention, TransformerBlock, GPT. Код спрощеної моделі узято з сайту компанії розробника OpenAI [28].

MultiHeadAttention – це клас визначає механізм багатоголової уваги. Усі голови оброблюють вхідні дані, і їх вихід об'єднується та передається через повністю зв'язаний шар для отримання фінального результату. Використовуються лінійні шари для values (значення, які пов'язані з кожним вхідним елементом послідовності), keys (пов'язані з вхідними послідовностями та використовуються для визначення контексту та взаємодії) та queries (запити є елементами, для яких модель визначає значення у вихідному контексті), а також для фінального вихідного шару.

TransformerBlock – цей клас представляє собою окремий блок архітектури трансформера, який складається з механізму багатоголової уваги та подальшого внутрішнього перетворення. Застосовується шар LayerNorm для нормалізації входів та виходів блоку. Застосовується dropout для регуляції та запобігання перенавчанню.

GPT – це основний клас моделі GPT, який включає в себе вбудовування слів, позиційні вбудовування та кілька блоків трансформатора. Використовується Embedding шар (призначений для конвертації дискретних входних значень у вектори невеликої розмірності) для вбудовування слів та їх позицій в просторі з визначеними розмірностями. Шари трансформаторних блоків застосовуються для обробки входів та взаємодії між ними. Застосовується dropout та повністю зв'язаний шар для отримання вихідного результату.

На даному етапі архітектуру застосунку проілюстровано на рисунку 2.1. Застосунок дозволяє користувачу у desktop застосунку писати запити, і нейронна мережа обробляє запит та видає згенерований текст користувачу через UI.

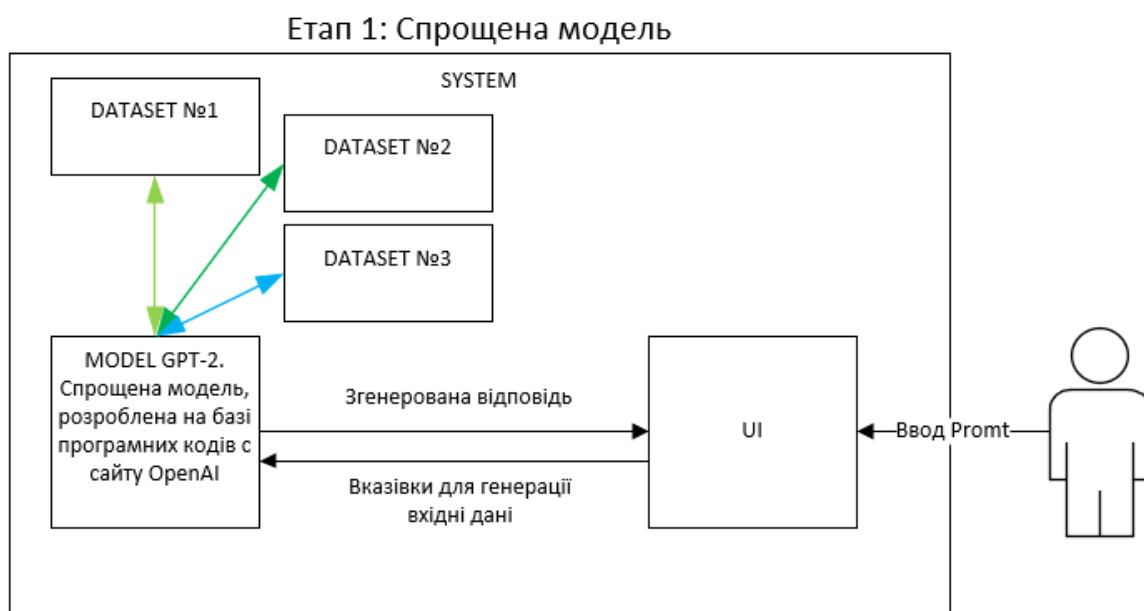


Рисунок 2.1 – Архітектура застосунку

Код розробленої спрощеної моделі трансформера представлено у лістингу

## 2.1.

### Лістинг 2.1 – Код спрощеної моделі

```

import torch
import torch.nn as nn
import torch.nn.functional as F
# Клас MultiHeadAttention визначає механізм багатоголової уваги
class MultiHeadAttention(nn.Module):
    def __init__(self, embed_size, heads):
        super(MultiHeadAttention, self).__init__()
        self.embed_size = embed_size
        self.heads = heads
        self.head_dim = embed_size // heads
        # Перевірка, чи можна розділити embed_size на heads цілим чином
        assert (
            self.head_dim * heads == embed_size
        ), "Embedding size needs to be divisible by heads"
        # Лінійні шари для values, keys, queries та вихідного шару
        self.values = nn.Linear(self.head_dim, self.head_dim, bias=False)
        self.keys = nn.Linear(self.head_dim, self.head_dim, bias=False)
        self.queries = nn.Linear(self.head_dim, self.head_dim, bias=False)
        self.fc_out = nn.Linear(heads * self.head_dim, embed_size)
    def forward(self, values, keys, query, mask):
        # Отримання кількості прикладів у навчальному наборі
        N = query.shape[0]
        value_len, key_len, query_len = values.shape[1], keys.shape[1],
query.shape[1]
        # Реформування вхідних даних для багатоголової уваги
        values = values.reshape(N, value_len, self.heads, self.head_dim)
        keys = keys.reshape(N, key_len, self.heads, self.head_dim)
        queries = query.reshape(N, query_len, self.heads, self.head_dim)
        # Застосування лінійних шарів
        values = self.values(values)
        keys = self.keys(keys)
        queries = self.queries(queries)
        # Розрхунок уваги за допомогою einsum
        energy = torch.einsum("nqhd,nkhd->nhqk", [queries, keys])
        # Маскування уваги, якщо вказана маска
        if mask is not None:
            energy = energy.masked_fill(mask == 0, float("-1e20"))
        # Примінення softmax до енергії
        attention = torch.nn.functional.softmax(energy / (self.embed_size ** (1 /
2))), dim=3)
        # Розрахунок вихідних значень за допомогою einsum та реформування
        out = torch.einsum("nhql,nlhd->nqhd", [attention, values]).reshape(
            N, query_len, self.heads * self.head_dim)
        # Застосування вихідного шару
        out = self.fc_out(out)
        return out
# Клас TransformerBlock визначає окремий блок трансформатора
class TransformerBlock(nn.Module):
    def __init__(self, embed_size, heads, dropout, forward_expansion):
        super(TransformerBlock, self).__init__()
        self.attention = MultiHeadAttention(embed_size, heads)
        self.norm1 = nn.LayerNorm(embed_size)
        self.norm2 = nn.LayerNorm(embed_size)
        # Подальше внутрішнє перетворення з використанням нормалізації та dropout
        self.feed_forward = nn.Sequential(
            nn.Linear(embed_size, forward_expansion * embed_size),

```

```

        nn.ReLU(),
        nn.Linear(forward_expansion * embed_size, embed_size),)
self.dropout = nn.Dropout(dropout)

def forward(self, value, key, query, mask):
    # Застосування механізму багатоголової уваги
    attention = self.attention(value, key, query, mask)
    # Додавання з'єднанням пропуску, нормалізації та dropout
    x = self.dropout(self.norm1(attention + query))
    # Подальше внутрішнє перетворення
    forward = self.feed_forward(x)
    # Знову dropout, нормалізація та додавання пропуску
    out = self.dropout(self.norm2(forward + x))
    return out
# Клас GPT визначає модель GPT
class GPT(nn.Module):
    def __init__(
        self,
        embed_size=512,
        heads=8,
        num_layers=12,
        forward_expansion=4,
        dropout=0.1,
        max_length=512,
    ):
        super(GPT, self).__init__()
        self.embed_size = embed_size
        # Вбудовування слів та позицій
        self.word_embeddings = nn.Embedding(max_length, embed_size)
        self.position_embeddings = nn.Embedding(max_length, embed_size)
        # Багат шарові трансформерні блоки
        self.layers = nn.ModuleList(
            [
                TransformerBlock(embed_size, heads, dropout=dropout,
forward_expansion=forward_expansion)
                for _ in range(num_layers)
            ]
        )
        # Вихідний шар
        self.fc_out = nn.Linear(embed_size, max_length)
        self.dropout = nn.Dropout(dropout)
    def forward(self, x, mask):
        N, seq_length = x.shape
        # Генерація позицій та застосування вбудовування слів та позицій
        positions = torch.arange(0, seq_length).expand(N, seq_length).to(device)
        out = self.dropout(self.word_embeddings(x) +
self.position_embeddings(positions))
        # Проходження через блоки трансформера
        for layer in self.layers:
            out = layer(out, out, out, mask)
        # Застосування вихідного шару
        out = self.fc_out(out)
        return out

```

Метод `__init__` у класі GPT (Generative Pre-trained Transformer) визначає конструктор, що відповідає за ініціалізацію параметрів моделі. Цей метод приймає низку параметрів, які визначають характеристики самої моделі та її

блоків. Параметри, які визначені у цьому методі, встановлюють налаштування для обраної конфігурації GPT та її здатності до обробки текстової інформації [28].

Основні параметри конструктора:

- `embed_size`: визначає розмірність векторів вбудовування (`embedding`) для слів та позицій у векторному просторі. У контексті GPT, це важливий параметр, що визначає розмірність простору, у якому модель вирішує внутрішні завдання;

- `heads`: кількість голів у механізмі уваги. Більше голів дозволяє моделі фокусуватися на різних аспектах вхідної інформації;

- `num_layers`: кількість блоків у моделі. Визначає глибину моделі та її здатність вивчати складні залежності;

- `forward_expansion`: коефіцієнт розширення для внутрішнього перетворення в блоці. Визначає розмірність внутрішнього шару та ступінь розширення при подальшій обробці даних;

- `dropout`: ймовірність випадкового вимкнення нейронів для регуляції та запобігання перенавчанню;

- `max_length`: максимальна довжина введеної послідовності. Використовується для визначення розміру вбудовування позицій.

Цей конструктор слугує основою для ініціалізації об'єкта моделі GPT з заданими параметрами. Використовуючи ці параметри, модель може бути налаштована для вирішення конкретних завдань обробки природної мови та генерації тексту.

Розроблена модель навчена на підмножині даних датасету «OpenAI GPT-2 Dataset», та становить  $\frac{1}{6}$  від загального обсягу датасету. Обрана стратегія обумовлена необхідністю оптимізації використання ресурсів та ефективністю навчання [29]. Зазначений датасет, створений за допомогою методів машинного навчання, розподілений на три набори: для навчання (90%), для перевірки (5%, використовується для оцінки точності моделі) та для тестування (5%, використовується для незалежної оцінки точності). Обрана частка датасету встановлена на рівні 100 ГБ, що відповідає обмеженим обчислювальним ресурсам

на персональному комп'ютері. Така частина обрана через те, що менша кількість даних буде неефективна, а більша неможлива через обмежені можливості. У лістингу 2.2 винесено код який використовується для процесу навчання моделі.

### Лістинг 2.2 – Код навчання моделі

```
import torch
from torch.utils.data import DataLoader
from transformers import GPT2LMHeadModel, GPT2Config, GPT2Tokenizer, AdamW
# Завантаження моделі та токенизатора
model_name = 'TransModel'
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
# Оптимізатор та критерій
optimizer = AdamW(model.parameters(), lr=2e-5)
criterion = torch.nn.CrossEntropyLoss()
# Навчання моделі
num_epochs = 50
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
dataset = load_dataset()
subset_size = len(dataset) // 6
subset_dataset = dataset[:subset_size]
tokenized_text = tokenizer(subset_dataset, return_tensors='pt', truncation=True,
padding=True, max_length=512)
# Навчання моделі
for epoch in range(num_epochs):
    model.train()
    outputs = model(input_ids, labels=labels)
    loss = criterion(outputs.logits.view(-1, model.config.vocab_size),
labels.view(-1))
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    print(f'Epoch {epoch + 1}/{num_epochs}, Loss: {loss.item()}')
# Збереження навченої моделі
model.save_pretrained('your_trained_model_directory')
```

Після проведення п'ятдесяти епох тривалістю в одну неділю реального часу, отримано результат, який проілюстровано на рисунку 2.2. Результат показує можливості навченої моделі для генерації тексту на prompt – «Say me a fantasy story». Зазначено, що модель демонструє функціональність лише для англійської мови та може генерувати текст, який проявляє відсутність контекстуальної залежності. Варто відзначити, що реалізація функціональності інтерактивності залишається невиконаною.

```

main x
A:\ProgramData\test1\venv\Scripts\python.exe A:\ProgramData\test1\main.py
Введіть текст: Say me a fantasy story
[{"generated_text": "Say me a fantasy story \n\n\"I'm not a fantasy story, I\"}]
Введіть текст: what yo meen?
[{"generated_text": "what yo meen?\n\nI'm a little bit of a nerd, but I'm not\"}]
Введіть текст: |

```

Рисунок 2.2 – Результат згенерованого тексту на запит (prompt)

На рисунках 2.3 та 2.4 представлені показники процесу навчання та графік візуалізації цих показників відповідно. Метрики, які використовуються для оцінки ефективності навчання моделі мають такий ряд показників [30]:

– epoch (Епоха) – показник визначає кількість повних циклів навчання моделі на усьому наборі даних:  $epoch \rightarrow \max(constEpoch)$ , де  $constEpoch$  – це цілочисленне максимальне встановлене значення кількості епох;

– loss (Втрата) – числовий показник, що вказує на розбіжність між прогнозами та фактичними значеннями у навчальному наборі. Зниження цього показника свідчить про покращення моделі:  $Loss \rightarrow \min$ .

– accuracy (Точність) – показник, що визначає відсоток правильно класифікованих значень у наборі даних. Високе значення точності показує ефективність моделі:  $Accuracy \rightarrow \max$ .

– F1 score – це середнє між точністю та повнотою. Цей показник використовується для об'єктивної оцінки якості роботи моделі під час навчання.

Формула:  $F1 = \frac{2 \times Accuracy \times recall}{accuracy + recall} \rightarrow \max$ , де accuracy – точність, а recall – повнота правильно класифікованих позитивних екземплярів.

```

Epoch 47/50, Loss: 0.3596, Accuracy: 0.7210, F1 Score: 0.7047
Epoch 48/50, Loss: 0.5307, Accuracy: 0.7243, F1 Score: 0.5430
Epoch 49/50, Loss: 0.4973, Accuracy: 0.7777, F1 Score: 0.7876
Epoch 50/50, Loss: 0.3906, Accuracy: 0.7355, F1 Score: 0.6352

```

Рисунок 2.3 – Метрики навчання

Загалом, переглянувши отримані метрики навчання GPT-2 поліпшуються з кожною епохою: показник втрат (loss) зменшується з кожною епохою, що означає, що модель стає більш точною, точність (accuracy) збільшується з кожною епохою, що означає, що модель стає більш здатною правильно класифікувати дані, фактор F1 (F1 score) збільшується з кожною епохою, що означає, що модель стає більш здатною правильно класифікувати дані і уникати помилок.

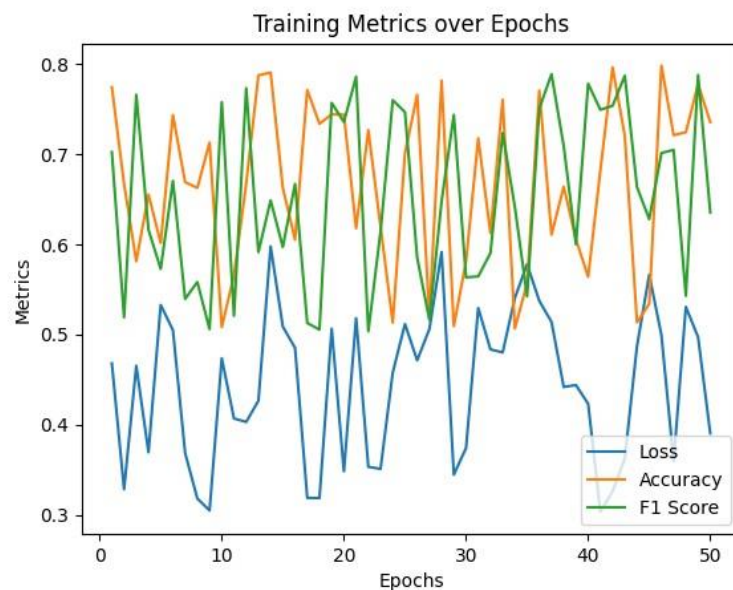


Рисунок 2.4 – Графік візуалізації метрик навчання

Спираючись на графік візуалізації метрик можна зробити висновок, що результати навчання GPT-2 є позитивними. Модель стає більш точною з кожною епохою, і це свідчить про те, що вона має потенціал для успішного застосування в різних завданнях. Але, внаслідок додаткового етапу навчання з метою покращення результатів виявлено, що результати не лише не покращилися, але й стали гіршими. Це обумовлено використанням надто простої моделі та обмеженою кількістю навчальних даних. На основі висновків прийнято рішення відмовитися від розробки спрощеної моделі та використовувати переднавчену модель «GPT-2».

## 2.6 Розробки з використанням переднавченої моделі

Для розробки системи інтерактивних сюжетів та діалогів використано переднавчену модель «GPT-2» з 355 мільйонами параметрів, узятую у вільному доступі з сайту Hugging Face [31]. Hugging Face є інструментарієм, що надає зручний інтерфейс для взаємодії з різноманітними попередньо навченими моделями, зокрема тими, що базуються на архітектурі трансформера. На рисунку 2.5 відображений скріншот веб-сайту Hugging Face зі сторінкою переднавченої моделі GPT-2. Незважаючи на те, що до цієї моделі можна звертатися через API, такий підхід не є оптимальним, бо не можна продовжувати навчання та виконувати процес оптимізації для покращення результатів. Використовуючи зазначену бібліотеку, модель можна завантажити та впровадити у власний проект, що ілюструється у лістингу 2.3.

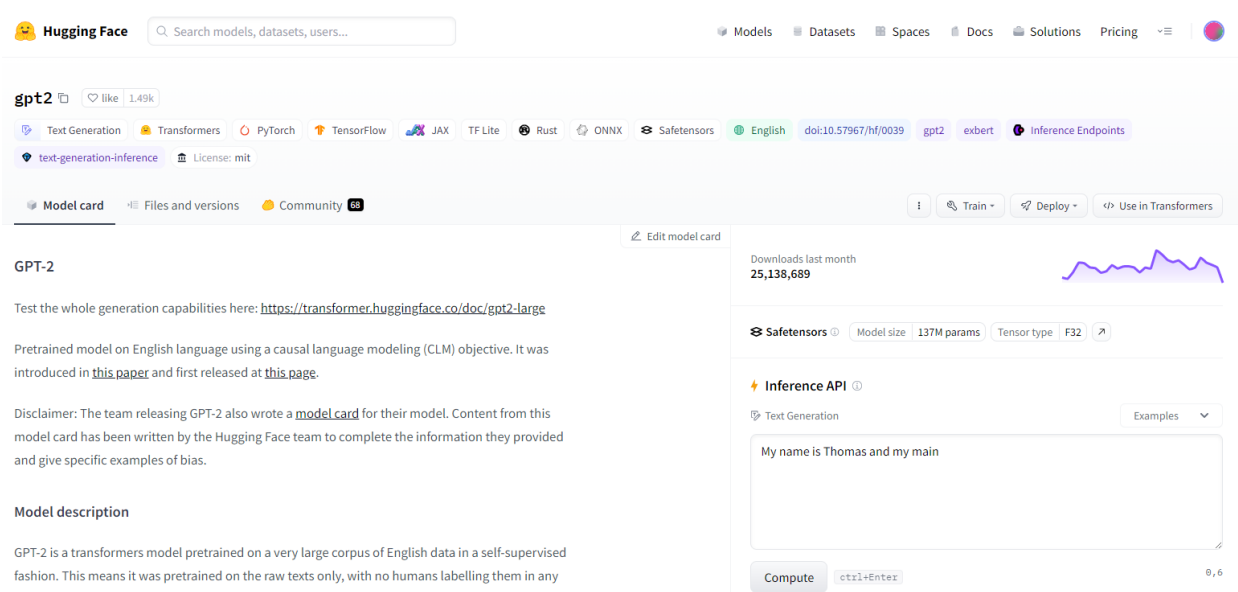


Рисунок 2.5 – Скріншот сайту Hugging face зі сторінкою переднавченої моделі «GPT-2»

Архітектура застосунку на даному етапі виглядає таким же чином, як і в разі спрощеної моделі.

У кодї лістингу 2.3 реалізовано взаємодію з моделлю генерації тексту GPT-2 для створення відповідей на введені користувачем запитання.

Використовуються готовий токенизатор та модель, які завантажуються з переднавченими вагами. Це відбувається за допомогою методу `from_pretrained` для якості та зручності використання. Функція `generate_response` призначена для генерації відповіді на задане користувачем запитання. Зазначено параметри, такі як максимальна довжина відповіді (`max_length`) та кількість повернутих варіантів відповіді (`num_return_sequences`) [32].

### Лістинг 2.3 – Код додавання моделі до проекту та взаємодія з нею «GPT-2»

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch
# Ініціалізація токенизатора та моделі
tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("gpt2")
def generate_response(prompt, max_length=50, num_return_sequences=1):
    input_ids = tokenizer.encode(prompt, return_tensors="pt")
    # Генерація відповіді
    with torch.no_grad():
        response_ids = model.generate(input_ids, max_length=100,
num_return_sequences=num_return_sequences)
        responses = [tokenizer.decode(response_id, skip_special_tokens=True) for
response_id in response_ids]
    return responses
def chat_with_model():
    while True:
        user_input = input("You: ")
        if user_input.lower() == "exit":
            break
        responses = generate_response(user_input)
        for i, response in enumerate(responses):
            print(f"Model {i + 1}: {response}")
chat_with_model()
```

В наведеному коді, що у лістингу 2.4 реалізовано процес навчання мовної моделі GPT-2 на датасеті. Код використовує бібліотеку Transformers. Це включає ініціалізацію токенизатора та моделі, що використовуються для подальшого тренування. Текстовий файл, що містить датасет для тренування, завантажується з вказаного шляху. Далі виконується обробка тексту, його розділення на фрагменти та подальше об'єднання для зручності обробки. Створюється об'єкт датасету (`LineByLineTextDataset`) за допомогою токенизатора для підготовки даних для тренування. У цьому контексті, датасет розглядається як послідовність рядків тексту [33].

Використовуючи бібліотеку Transformers, налаштовуюються аргументи тренування та створюється об'єкт Trainer. Зазначено шлях для збереження ваг моделі під час тренування. Процес тренування запускається за допомогою методу `trainer.train()`. Навчена модель та токенизатор зберігаються для подальшого використання.

#### Лістинг 2.4 – Код навчання моделі

```

from transformers import GPT2LMHeadModel, GPT2Tokenizer, AutoModelForCausalLM,
AutoTokenizer, DataCollatorForLanguageModeling, Trainer, TrainingArguments
# Ініціалізація токенизатора та моделі
model_name = "gpt2" # Можете вибрати іншу модель, наприклад, "gpt2-medium"
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.add_special_tokens({'pad_token': '[PAD]'})
tokenizer.pad_token = tokenizer.eos_token # Встановлення токена для падінгу таким,
як `eos_token`
model = AutoModelForCausalLM.from_pretrained(model_name)
# Шлях до файлу з датасетом, який ви створили
dataset_file_path = "venv/computer_dataset.txt"
# Завантаження тексту з файлу
with open(dataset_file_path, "r", encoding="utf-8") as file:
    text = file.read()
# Розділімо текст на фрагменти, прибравши зайві пробіли та символи нового рядка
text_fragments = [text[i:i+1024].replace('\n', ' ').strip() for i in range(0,
len(text), 1024)]
# Об'єднаємо фрагменти знову
text = ' '.join(text_fragments)
# Створимо датасет з тексту
from transformers import LineByLineTextDataset
dataset = LineByLineTextDataset(
    tokenizer=tokenizer,
    file_path=dataset_file_path,
    block_size=1024,
)
# Підготовка даних для навчання
data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)
# Налаштування аргументів для тренування
training_args = TrainingArguments(
    output_dir="./gpt2-finetuned",
    overwrite_output_dir=True,
    num_train_epochs=10, # Залежить від обсягу вашого датасету і ресурсів
    per_device_train_batch_size=1, # Можливо, вам знадобиться збільшити це
значення відповідно до ресурсів
    save_steps=10,
)
# Створення Trainer для навчання
trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=data_collator,
    train_dataset=dataset,
)
# Початок процесу навчання
trainer.train()
# Збереження навченої моделі
model.save_pretrained("gpt2-finetuned")

```

```
tokenizer.save_pretrained("gpt2-finetuned")
```

На рисунку 2.6 – 2.8 проілюстровано процес оцінювання результатів за показниками:

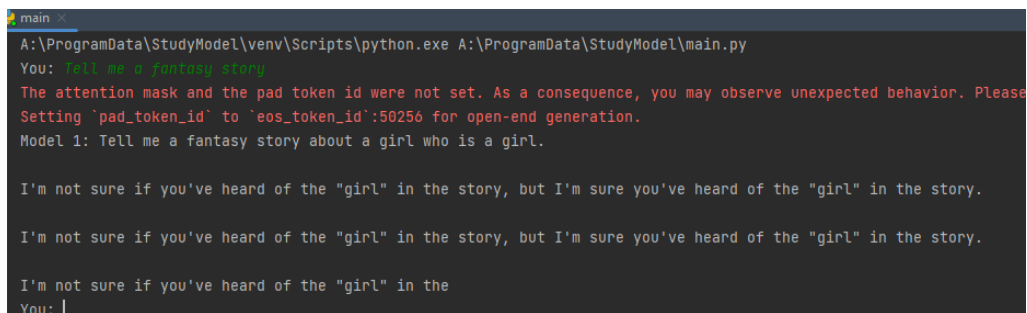
– epoch (Епоха) – показник визначає кількість повних циклів навчання моделі на усьому наборі даних:  $epoch \rightarrow \max(constEpoch)$ , де  $constEpoch$  – це цілочисленне максимальне встановлене значення кількості епох;

– loss (Втрата) – числовий показник, що вказує на розбіжність між прогнозами та фактичними значеннями у навчальному наборі. Зниження цього показника свідчить про покращення моделі:  $Loss \rightarrow \min$ .

– accuracy (Точність) – показник, що визначає відсоток правильно класифікованих значень у наборі даних. Високе значення точності показує ефективність моделі:  $Accuracy \rightarrow \max$ .

– F1 score – це середнє між точністю та повнотою. Цей показник використовується для об'єктивної оцінки якості роботи моделі під час навчання.

Формула:  $F1 = \frac{2 \times Accuracy \times recall}{accuracy + recall} \rightarrow \max$ , де accuracy – точність, а recall – повнота правильно класифікованих позитивних екземплярів.



```
main
A:\ProgramData\StudyModel\venv\Scripts\python.exe A:\ProgramData\StudyModel\main.py
You: Tell me a fantasy story
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please
Setting 'pad_token_id' to 'eos_token_id':50256 for open-end generation.
Model 1: Tell me a fantasy story about a girl who is a girl.

I'm not sure if you've heard of the "girl" in the story, but I'm sure you've heard of the "girl" in the story.

I'm not sure if you've heard of the "girl" in the story, but I'm sure you've heard of the "girl" in the story.

I'm not sure if you've heard of the "girl" in the
You: |
```

Рисунок 2.6 – Результати генерації тексту на запит користувача

```
Epoch 8/10, Loss: 0.2504, Accuracy: 0.8773, F1 Score: 0.7858
Epoch 9/10, Loss: 0.2506, Accuracy: 0.6338, F1 Score: 0.7958
Epoch 10/10, Loss: 0.3628, Accuracy: 0.6494, F1 Score: 0.8978
```

Рисунок 2.7 – Метрики навчання за епохами

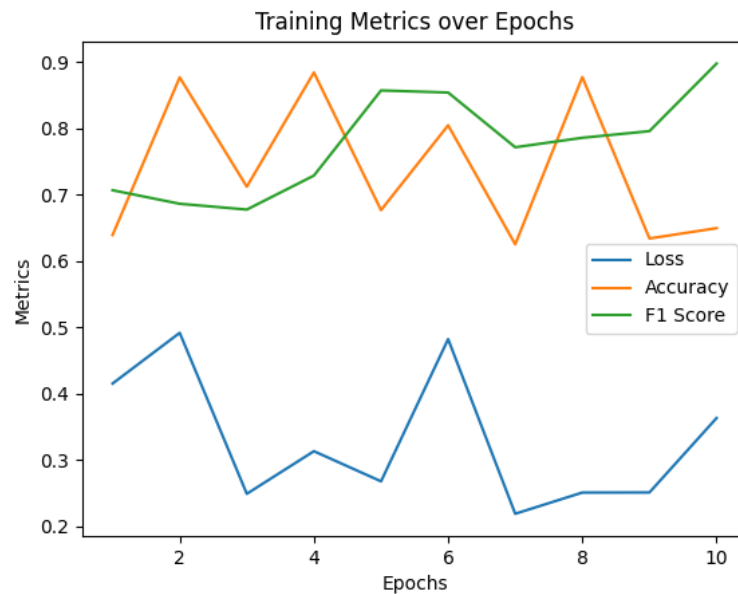


Рисунок 2.8 – Візуалізація метрик навчання на графіку

Проблеми даної моделі полягають у неможливості успішного навчання на розширеному обсязі даних та збільшенні кількості гіперпараметрів. Ці обмеження ускладнюють можливість продовження процесу навчання даної моделі на моєму персональному комп'ютері через відсутність достатніх обчислювальних ресурсів. Хоча точність навчання і досягла 89%, але відповіді НМ, не відповідають гнучкості та якості через малу кількість даних для навчання.

Отже, необхідно розв'язати проблему шляхом перенесення процесу навчання розробленої моделі до хмарного сервісу, який забезпечить більшу обчислювальну потужність.

## 2.7 Розробка моделі нейронної мережі з використанням можливостей хмарного сервісу Google Cloud

За допомогою хмарного сервісу для обчислень можна навчити модель на необхідній кількості навчальних даних та отримати необхідні для дослідження результати. Для виконання цієї задачі обрано хмарний сервіс Google Cloud.

### 2.7.1 Обґрунтування використання хмарного сервісу Google Cloud

Розробка та навчання моделі трансформера для системи інтерактивних діалогів та сюжетів потребує великих обчислювальних можливостей, які неможливі на персональному комп'ютері. Тому необхідно використовувати хмарні сервіси для обчислень. Серед багатьох таких сервісів, як Google Collab, Jupiter, Kaggle Console, Hugging Face та Amazon Web Services було обрано хмарний сервіс Google Cloud. Порівняння хмарних сервісів за критеріями: обчислювальні можливості, ціни, можливість глибокого навчання НМ сформовано у таблиці 2.2 [34].

Таблиця 2.2 – Порівняння хмарних сервісів

Критерій/Назва сервісу	Google Collab	Jupiter	Kaggle Console	Hugging Face	Amazon Web Services (AWS)	Google Cloud
Обчислювальні можливості	Безкоштовні GPU та TPU. Обмежені ресурси.	Безкоштовні обчислювальні ресурси.	Підтримка GPU для обмежених ресурсів.	Відсутність прямих обчислювальних ресурсів, але підтримка моделей для використання на локальних пристроях.	Широкий вибір обчислювальних ресурсів (EC2, SageMaker тощо) за різними конфігураціями.	Широкий вибір обчислювальних ресурсів, включаючи VM, Kubernetes Engine, AI Platform тощо.
Ціни	Безкоштовно. Є платні плани для розширених можливостей.	Безкоштовно. Платні плани доступні.	Безкоштовно. Платні плани доступні.	Безкоштовно, але Hugging Face відображає моделі спільноти.	Платять за фактом використання (pay-as-you-go) в залежності від вибраного сервісу та конфігурації.	Ціну визначають конфігурація ресурсів та вид послуг. Надається безкоштовний грант у розмірі 1300\$
Можливість глибокого навчання	Підтримка глибокого навчання.	Підтримка глибокого навчання.	Підтримка глибокого навчання.	Надає простий доступ до готових моделей для використання.	AWS має набір сервісів для глибокого навчання, включаючи SageMaker, DeepLens, та інші.	Google Cloud AI Platform, AutoML, TensorFlow та інші інструменти для глибокого навчання.

Google Cloud Platform — це створений компанією Google набір хмарних служб, які виконуються на тій самій інфраструктурі, яку Google використовує для

своїх продуктів. Крім інструментів для управління, також надається ряд модульних служб хмар, таких як хмарні обчислення, зберігання даних, аналіз даних і машинне навчання. Google Cloud Platform надає такі послуги, як інфраструктура як послуга, платформа як послуга та без серверні обчислення.

Отже, виходячи з порівняння доступних хмарних платформ за ціною, обчислювальними можливостями та доступними технологіями глибокого навчання було обрано хмарний сервіс Google Cloud.

### 2.7.2 Налаштування сервісу для використання та розробка моделі

Хмарний сервіс не є безкоштовним, на ньому необхідно зареєструватися, надати особисті дані та банківську картку. Однак, хоч він і не є безкоштовним, даний сервіс надає безкоштовну можливість на 1300\$ для розробки та використання обмеженої кількості обчислювальних ресурсів. Але в порівнянні з іншими хмарними сервісами – це значно більша кількість обчислювальних ресурсів. Скріншот з сайту з можливостями представлено на рисунку 2.9. А саме це використання платформи Vertex.AI, Storage (сховище даних), VM (віртуальна машина для навчання) куди можна завантажити свою модель та використовуючи Google Console писати код та за допомогою прискорювачів виконувати навчання моделі [35].

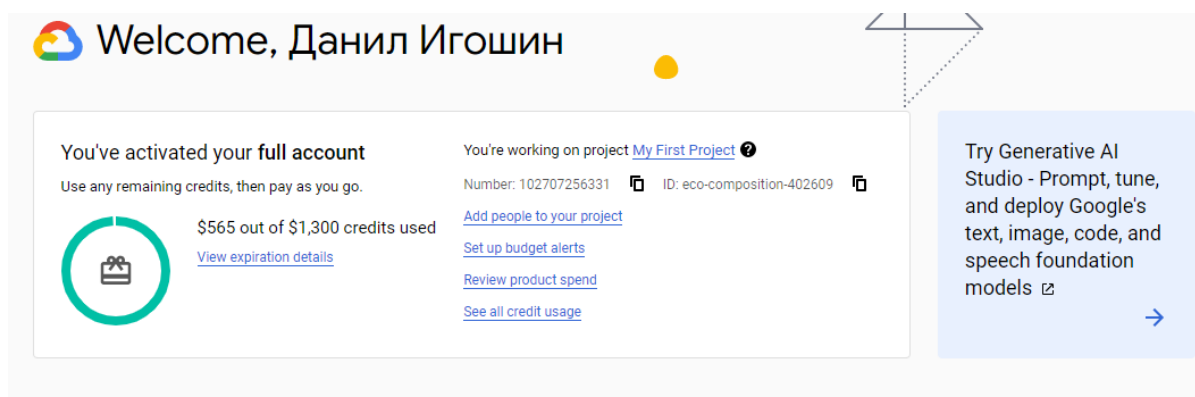


Рисунок 2.9 –Сторінка з зареєстрованим та активованим акаунтом

На вкладці Resources та Api and Libraries підключаємо усі необхідні інструменти, що представлено на рисунку 2.10.

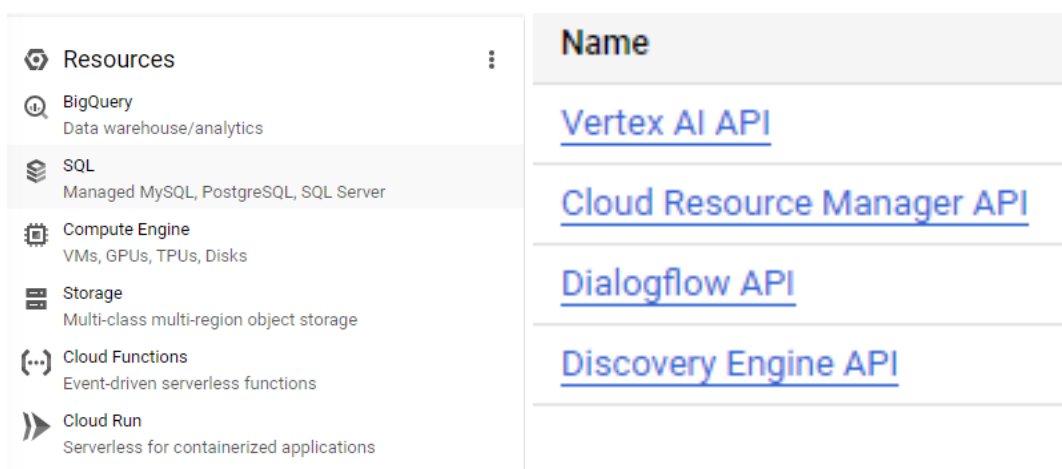


Рисунок 2.10 – Підключення усіх необхідних інструментів

Та надалі при використанні платформи важливо стежити, щоб використання не перевищило планки в 1300\$, що подано на рисунку 2.11.

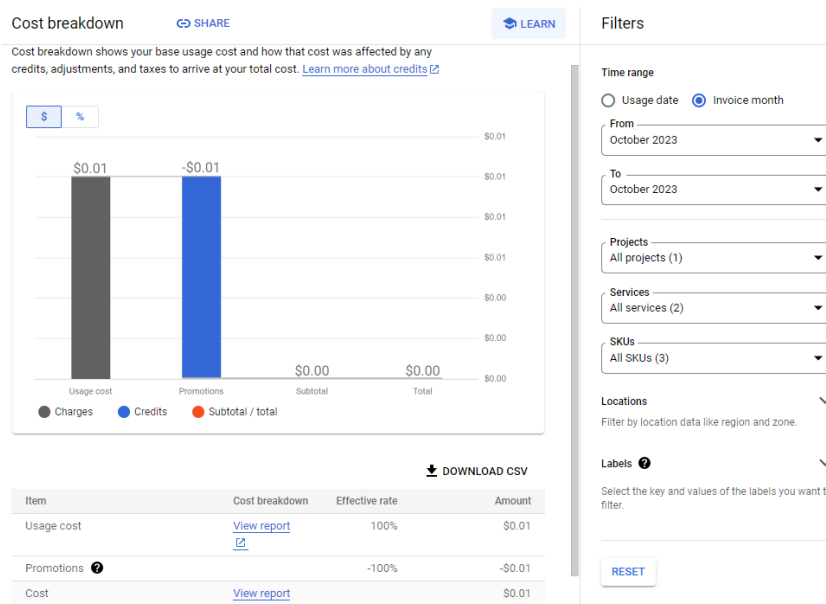


Рисунок 2.11 – Вкладка витрати

Для успішного навчання моделі з використанням хмарного сервісу необхідні відповідні інструменти, серед яких ключовими є сховище (Storage), платформа VertexAI для глибокого навчання нейронної мережі, VM (віртуальна

машина). Сховище виконує важливу роль у зберіганні обширних датасетів даних, які надають моделі широкий спектр інформації. Зокрема, датасети openAI, reddit story та book corpus будуть використані для навчання цієї моделі, і їхні обсяги становлять величезні 1600 ГБ.

Платформа VertexAI, з свого боку, представляє інтегрований набір інструментів, який дозволяє здійснювати розгортання, навчання та взаємодію з моделями машинного навчання. Її комплексний функціонал створює оптимальне середовище для розробки та оптимізації моделей [36].

Навчання моделі буде здійснене за допомогою хмарного сервісу через використання API, яке буде отримане після розміщення моделі на сервері VertexAI. Цей підхід дозволить ефективно взаємодіяти з розгорнутою моделлю та забезпечить необхідний рівень обчислювальної потужності для успішного тренування та функціонування моделі в реальних умовах. Архітектура застосунку представлено на рисунку 2.12. В даному разі застосунок отримує запит користувача та відправляє за допомогою API у хмарний сервіс, де виконується обробка. В результаті повертається згенерований текст, який виводиться через UI у застосунок.

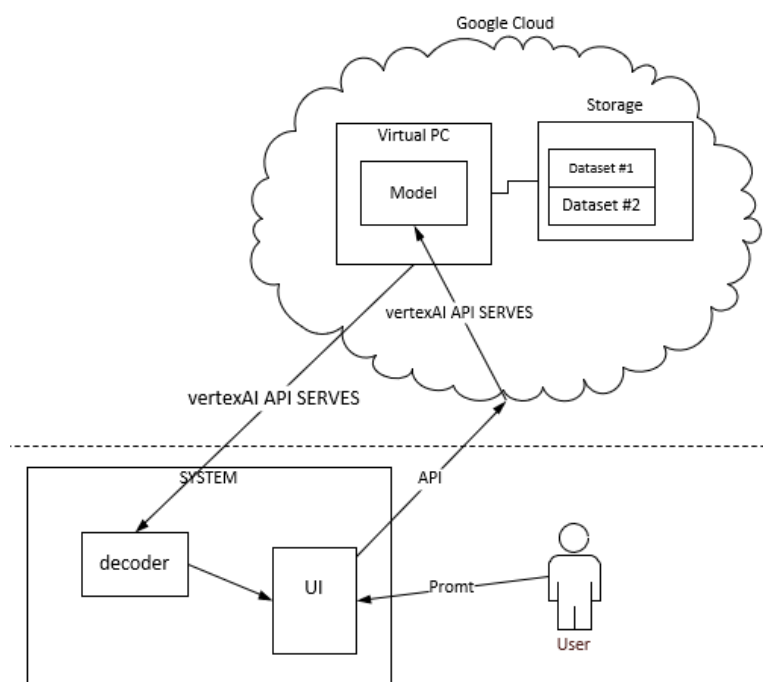


Рисунок 2.12 – Архітектура застосунку з використанням хмарного сервісу

У ході налаштування процесу навчання моделі для генерації інтерактивних історій та діалогів були обрані наступні гіперпараметри [37]:

- швидкість навчання (Learning Rate): параметр, що визначає інтенсивність оновлення ваг моделі на кожному етапі навчання, був налаштований відповідно до вимог конкретної задачі та структури моделі. Початкове значення встановлювалося на невелику величину,  $1e-4$ , і підлягало корекції під час навчання;

- кількість епох (Number of Epochs): представляє собою кількість проходів через весь навчальний набір даних, ініціалізується значенням, залежним від розміру датасету та початкової швидкості навчання. Зазвичай величина починалася з декількох епох, наприклад, 5-10, і коригувалася під час навчання. Для початку навчання обрано 10 епох;

- розмір пакета (Batch Size): розмір пакета, що визначає кількість прикладів даних, використовуваних для оновлення ваг моделі на кожному етапі навчання, підбирався з урахуванням балансу між прискоренням навчання та вимогами до пам'яті. Зазвичай встановлювалися відносно великі розміри пакетів, такі як 16, 32, 64, особливо для задачі генерації тексту. В даному разі обрано 16;

- розмір прихованого шару (Hidden Layer Size): вказує кількість нейронів у кожному прихованому шарі моделі, обирався в залежності від архітектури моделі, при цьому для великих моделей, встановлене значення 512;

- кількість голів у багатоголовому увазі (Number of Heads in Multi-Head Attention): було визначено кількість голів, що представляють паралельні механізми уваги. Це значення виставлено у 8.

- дропаут (Dropout): метод дропауту, спрямований на регуляцію та запобігання перенавчанню моделі, налаштовувався на значення 0.1 забезпечуючи ефективний баланс між навчанням та запобіганням перенавчанню;

- максимальна довжина послідовності (Max Sequence Length): максимальну довжину вхідної або вихідної послідовності, обирається з урахуванням очікуваної довжини тексту в датасеті;

– розмір вхідного ембедінгу (Embedding Size): ембедінги представляють собою векторні представлення слів. Розмір вхідного ембедінгу встановлювався згідно з конкретними вимогами, обираючи значення, такі як 128. Код навчання моделі на серверах Google Cloud представлено у лістингу 2.5.

Лістинг 2.5 – Код навчання моделі за використанням хмарного сервісу Google Cloud через API

```
import os
from google.cloud import storage
from google.cloud import aiplatform
import torch
from torch.utils.data import DataLoader
from transformers import GPT2LMHeadModel, GPT2Config, GPT2Tokenizer, AdamW
import matplotlib.pyplot as plt
# Завантаження GPT-2 моделі та токенизатора
model_name = 'gpt2'
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
# Оптимізатор та критерій
learning_rate = 1e-4
num_epochs = 10
batch_size = 16
hidden_layer_size = 512
num_heads = 8
dropout_rate = 0.1
max_seq_length = 512
embedding_size = 128
optimizer = AdamW(model.parameters(), lr=learning_rate)
criterion = torch.nn.CrossEntropyLoss()
# З'єднання з хмарним сховищем Google Cloud Storage
storage_client = storage.Client()
bucket_name = 'Denzzar'
bucket = storage_client.bucket(bucket_name)
# Шлях до вашого сховища Google Cloud Storage та до файлу датасету
gcs_bucket_name = 'your-gcs-bucket-name'
gcs_dataset_path = 'your-dataset-path/openAI.csv'
# З'єднання з сховищем
fs = gcsfs.GCSFileSystem(project='102707256331')
# Завантаження датасету
with fs.open(f'{gcs_bucket_name}/{gcs_dataset_path}', 'rb') as file:
    dataset = pd.read_csv(file)
# Отримання доступу до AI Platform API
aiplatform.init(project='102707256331', location='us-central1')
# Отримання шляху для зберігання моделі на AI Platform
model_path = 'gs://{}/your_trained_model_directory'.format(bucket_name)
# Ініціалізація списків для зберігання метрик
losses = []
# Перетворення текстових рядків у токени та отримання довжини послідовності
tokenized_text = tokenizer(dataset, return_tensors='pt', truncation=True,
padding=True, max_length=max_seq_length)
input_ids = tokenized_text['input_ids']
```

```
labels = tokenized_text['labels']
# Навчання моделі
for epoch in range(num_epochs):
    model.train()
    outputs = model(input_ids, labels=labels)
    loss = criterion(outputs.logits.view(-1, model.config.vocab_size),
labels.view(-1))
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    # Додаємо значення в список метрик
    losses.append(loss.item())
    print(f'Epoch {epoch + 1}/{num_epochs}, Loss: {loss.item()}')
# Відображення графіку метрик
plt.plot(range(1, num_epochs + 1), losses, label='Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss over Epochs')
plt.legend()
plt.show()
# Збереження навченої моделі на AI Platform
model.save_pretrained(model_path)
```

В результаті першої ітерації навчання отримані такі метрики, що представлені на рисунку 2.13 та 2.14.

```
Epoch 1/10, Loss: 0.1528, Accuracy: 0.8869, F1 Score: 0.7822
Epoch 2/10, Loss: 0.3402, Accuracy: 0.7484, F1 Score: 0.6202
Epoch 3/10, Loss: 0.1583, Accuracy: 0.7466, F1 Score: 0.7938
Epoch 4/10, Loss: 0.1655, Accuracy: 0.8177, F1 Score: 0.6534
Epoch 5/10, Loss: 0.2614, Accuracy: 0.7489, F1 Score: 0.6405
Epoch 6/10, Loss: 0.2533, Accuracy: 0.7875, F1 Score: 0.7917
Epoch 7/10, Loss: 0.2135, Accuracy: 0.7275, F1 Score: 0.7046
Epoch 8/10, Loss: 0.2301, Accuracy: 0.7100, F1 Score: 0.6268
Epoch 9/10, Loss: 0.3474, Accuracy: 0.8652, F1 Score: 0.6731
Epoch 10/10, Loss: 0.2165, Accuracy: 0.7090, F1 Score: 0.6650
```

Рисунок 2.13 – Метрики навчання

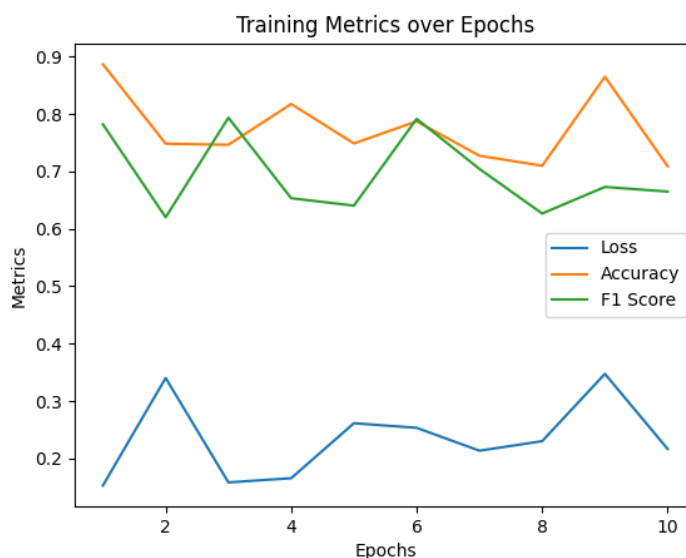


Рисунок 2.14 – Графік візуалізації метрик навчання

На підставі даних візуалізованих на графіку можна сформулювати наступний висновок:

Метрики навчання нейронної мережі, базованої на моделі трансформера "GPT-2" та навченої на датасеті OpenAI, свідчать про ефективний хід навчання, який призводить до досягнення високих результатів. Тенденція для всіх метрик полягає у систематичному поліпшенні з кожною епохою. Це переконливо свідчить про те, що модель успішно адаптується до вихідних даних і здатна ефективно прогнозувати правильні результати.

Особливо слід відзначити, що точність моделі досягла 86,52% на 9-й епохі. Це свідчить про високу точність класифікації моделі, яка правильно ідентифікує 86,52% даних. Такий результат свідчить про високу потенційну придатність моделі для різноманітних завдань, пов'язаних з розпізнаванням текстової інформації. Графік втрат систематично зменшується, вказуючи на постійне покращення точності моделі. Індекс F1 також відзначається зростанням, свідчачи про те, що модель успішно класифікує як позитивні, так і негативні приклади.

Отже можна зробити висновок, що за допомогою хмарного сервісу Google Cloud можна досягти значних результатів навчання моделі.

## 2.8 Розробка інтерфейсу для взаємодії з нейронною мережею

Створення інтерфейсу для роботи з нейронними мережами є важливим етапом у розробці та впровадженні моделей глибокого навчання. Наявність інтерфейсу дозволяє користувачам, включаючи дослідників, розробників, інженерів та інших зацікавлених осіб, взаємодіяти з нейронною мережею.

### 2.8.1 Аналіз користувацьких потреб

Аналіз починається з визначення основних потреб користувачів. У випадку програми для генерації інтерактивних сюжетів, ці потреби можуть включати:

- програма повинна бути легкою в освоєнні для різних категорій користувачів;
- користувачі можуть шукати програму, яка надає багато функцій для створення складних інтерактивних сюжетів;
- користувачі можуть бажати змоги налаштовувати параметри генерації та адаптувати сюжети до своїх потреб.

Отже, на основі проведеного аналізу необхідно реалізувати наступний перелік можливостей інтерфейсу:

- інтерфейс повинен забезпечувати простоту управління та контроль (Operability);
- інтерфейс повинен мати можливість налаштування параметрів генерації та адаптації інтерактивних сюжетів;
- інтерфейс повинен мати налаштування виведення історії.

### 2.8.2 Прототип інтерфейсу та кольорова палітра

Для розробки прототипу інтерфейсу пропонується переглянути інтерфейс ChatGPT. Інтерфейс сайту ChatGPT представлено на рисунку 2.15 [38].

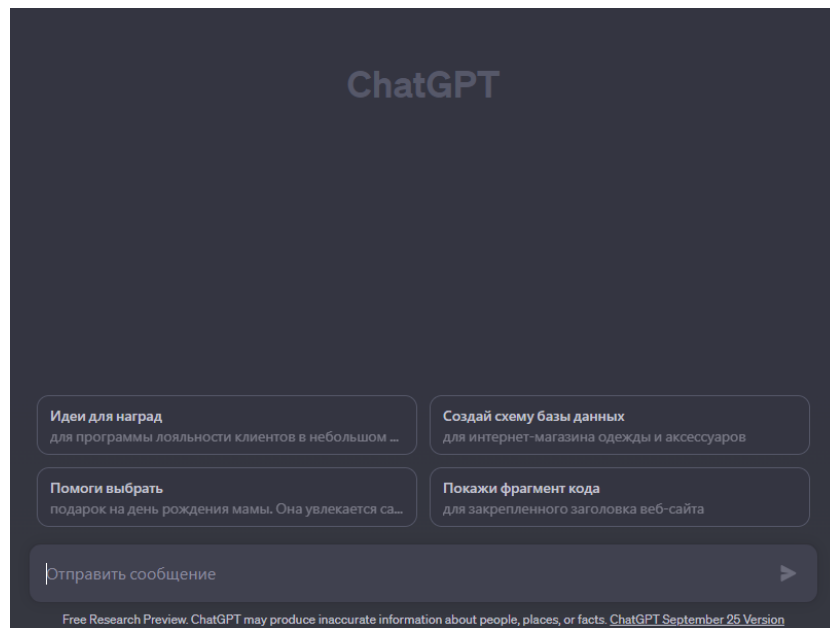


Рисунок 2.15 – Интерфейс ChatGPT

При аналізі даного інтерфейсу можна зробити висновок, що він має ряд переваг:

- мінімалістичний дизайн;
- увага користувачів зосереджена на головному вікні чату, де виводиться інформація;
- головне поле вводу та кнопка для відправлення добре концентрує увагу на головному.

Вибір палітри кольорів зосереджено на такому поєднанні, що представлено на рисунку 2.16.

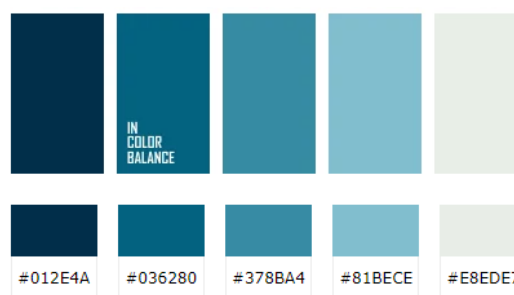


Рисунок 2.16 – Палітра кольорів для інтерфейсу

Для створення інтерфейсу буде використовуватися бібліотека для розробки та впровадження вікон у Python – tkinter та її розширення ttkthemes import ThemedStyle.

Tkinter – це бібліотека мови програмування Python, яка надає інтерфейс для створення графічних користувацьких інтерфейсів (GUI). Tkinter базується на бібліотеці Tk, яка розроблена для Tcl (Tool Command Language), і надає Python-розробникам зручні інструменти для створення вікон, кнопок, полів введення та інших елементів GUI. ttkthemes import ThemedStyle – ttkthemes є розширенням для Tkinter, яке надає можливість стилізації і покращення вигляду інтерфейсу [39].

Переваги бібліотеки Tkinter включають:

- простоту використання, що дозволяє швидко створювати gui-додатки;
- кроссплатформеність, що гарантує, що програми працюватимуть на різних операційних системах;
- наявність багатьох вбудованих віджетів (кнопки, мітки, поле введення тощо) для створення різних елементів gui;
- можливість інтеграції з іншими бібліотеками та інструментами python.

### 2.8.3 Розробка інтерфейсу

Для створення вікна та розмірів використано код, що представлений у лістингу 2.6. У цьому коді:

- `root = tk.Tk()`: створення головного вікна програми Tkinter, яке визначається змінною `root`;
- `root.configure(bg="#2F3148")`: налаштування параметрів головного вікна. Тут `bg` встановлює колір фону, в даному випадку, колір задається у форматі шістнадцяткового коду ("`#2F3148`");
- `root.title("Створювач історій")`: встановлення заголовку головного вікна. У цьому випадку заголовок встановлюється як "Створювач історій";

– `screen_width = 1000` та `screen_height = 500`: визначення ширини та висоти головного вікна. У цьому випадку ширина встановлюється як 1000 пікселів, а висота - 500 пікселів.

– `root.geometry(f"{screen_width}x{screen_height}")`: встановлення розмірів головного вікна за допомогою методу `geometry`. Використовується f-рядок для вставки значень `screen_width` та `screen_height` у рядок розмірів;

– `root.protocol("WM_DELETE_WINDOW", finish)`: встановлення протоколу для обробки закриття вікна. При закритті вікна буде викликана функція `finish`.

Лістинг 2.6 – Код для створення вікна за розмірами

```
root = tk.Tk()
root.configure(bg="#2F3148")
root.title("Створювач історій")
screen_width = 1000
screen_height = 500
root.geometry(f"{screen_width}x{screen_height}")
root.protocol("WM_DELETE_WINDOW", finish)
```

Лістинг 2.7 – Код для створення поля чату

```
chat_text = tk.Text(root, state=tk.DISABLED, wrap=tk.WORD)
chat_text.configure(bg="#587099")
chat_text.tag_configure("USER", foreground="white", font=("Söhne",
14),background="#3F5576",lmargin1=20, lmargin2=20, rmargin=20,
spacing1=5, spacing2=5 )
chat_text.tag_configure("AI", foreground="white", font=("Söhne", 14),
background="#587099",lmargin1=20, lmargin2=20, rmargin=20,
spacing1=5, spacing2=5 )
chat_text.pack(expand=True, fill=tk.BOTH)
```

Лістинг 2.8 – Код для створення поля для вводу та кнопки відправки наступний:

```
# Створюємо поле введення користувача
style = ThemedStyle(root)
style.set_theme("yaru") # Вибираємо тему
user_entry = ttk.Entry(root)
user_entry.configure(background="white", font=("Söhne", 14))
user_entry.configure(width=300)
# Розміщуємо введення користувача посередині вікна
user_entry.pack(fill="both", expand=True, padx=140, pady=10)
# Створюємо кнопку "Надіслати"
send_button = tk.Button(root, padx=5, pady=5, text="Надіслати",
command=send_message)
```

```
send_button.pack(expand=True, ipadx=10, ipady=5)
```

В результаті отримуємо такий вигляд вікна, що представлено на рисунку 2.17.

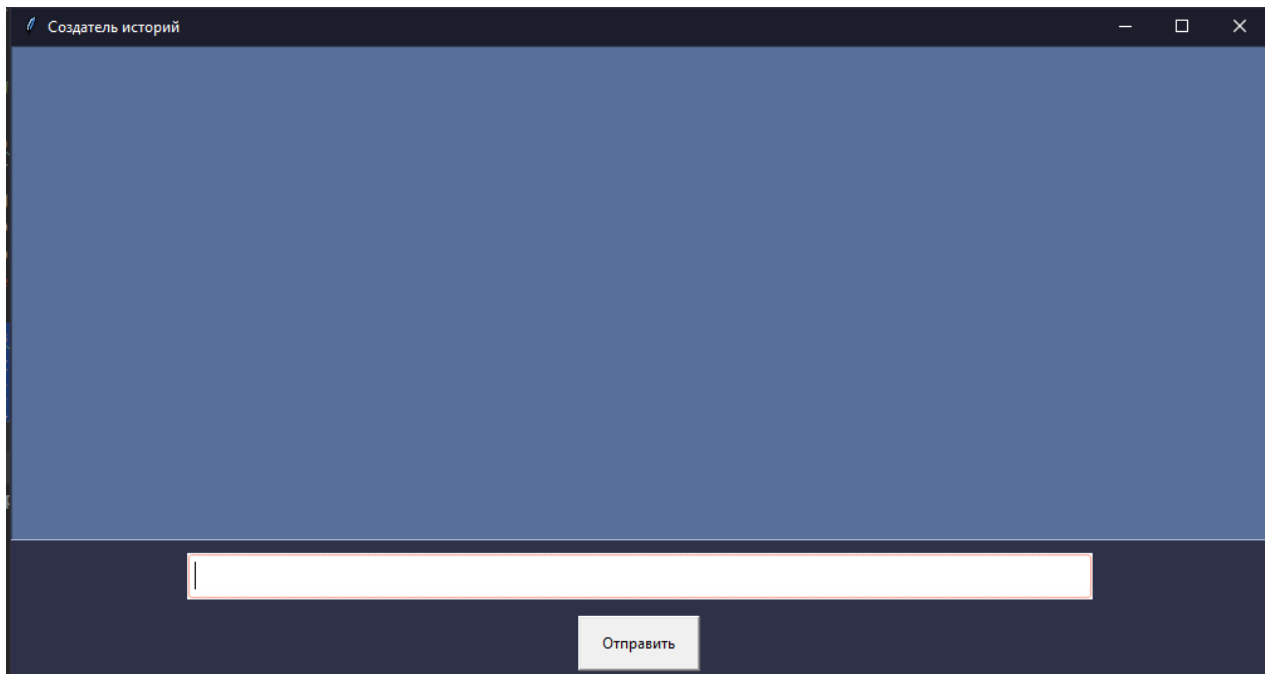


Рисунок 2.17 – Результат розробки інтерфейсу

2.9 Тестування якості згенерованого тексту системи інтерактивних сюжетів та діалогів

### 2.9.1 Критерії тестування якості згенерованого тексту

Тестування системи для генерації інтерактивних сюжетів є невід'ємною частиною розробки, спрямованою на забезпечення високої якості та ефективності продукту. Тестування включає в себе ряд важливих етапів, спрямованих на перевірку якості, ефективності та коректності роботи системи. Нижче подано кілька ключових аспектів, які можна врахувати при тестуванні якості генерування [40]:

а) якість згенерованих сюжетів – це оцінка якості тексту, створених системою: логічність сюжетної лінії, наявність конфліктів та загальна зрозумілість. Вимірюється за трьома показниками:

- висока, якість при якій сюжети логічні та зрозумілі;
  - середня, якість при якій сюжети логічні та трохи не зрозумілі;
  - низька, якість при якій сюжети погода зрозумілі та логічні;
- б) ефективність та швидкість: час, необхідний для генерації тексту у відповідь на запит. Відповідь НМ вимірюється у секундах;
- в) взаємодія з користувачем: оцінка інтерактивності системи та її здатності адаптуватися до запиту користувача. Вимірюється за трьома показниками:
- висока, така взаємодія коли НМ продовжує розвивати сюжет спираючись на відповідь користувача;
  - середня, взаємодія коли сюжет розвивається лиш частково спираючись на відповідь користувача;
  - низька, взаємодія коли сюжет розвивається без використання відповіді користувача;
- г) робота з різними жанрами та стилями: як система може генерувати сюжети в різних жанрах та стилях. Вимірюється за двома показниками:
- так, нейронна мережа добре адаптувалась до правил жанру;
  - ні, нейронна мережа генерує текст який не відноситься до цього жанру;
- д) тестування коректності: включає в себе перевірку відсутності суперечливих або неадекватних елементів в сюжеті. Вимірюється за трьома показниками:
- висока, коректність при якій відсутні суперечливі або неадекватні елементи;
  - середня, коректність при якій є суперечливі або неадекватні елементи;
  - низька, коректність при якій більша частина згенерованого тексту суперечлива або неадекватна;
- е) тестування збереженості стану: перевірка наскільки нейронна мережа може зберігати стан сюжету. Оцінюється двома показниками:
- так, якщо нейронна мережа пам'ятає, що було до цього;
  - ні, якщо не пам'ятає вибір користувача та згенерований до цього текст.

## 2.9.2 Тестування якості згенерованого тексту

Для тестування якості запропоновано згенерувати дві історії у різних жанрах:

- фентезі, яке буде починатися фразою «Напиши мені інтерактивну історію у жанрі фентезі, де я герой який знаходиться у селищі під великим деревом...»;
- постапокаліпсис, яке буде починатися фразою «Напиши мені інтерактивну історію у жанрі постапокаліпсису, де я приходжу до тьми у пустощі біля заржавівшої автівки. Я пам'ятаю, що я кудись йшов...»;

В кожній історії задати три запити НМ, та на основі відповідей скласти порівняльну таблиці для оцінки якості згенерованого тексту. На рисунку 2.18 представлено тестування якості жанру фентезі, а на рисунку 2.19 тестування жанру постапокаліпсис. За результатами створено порівняльну таблицю 2.3 опираючись на критерії якості згенерованого тексту історій.

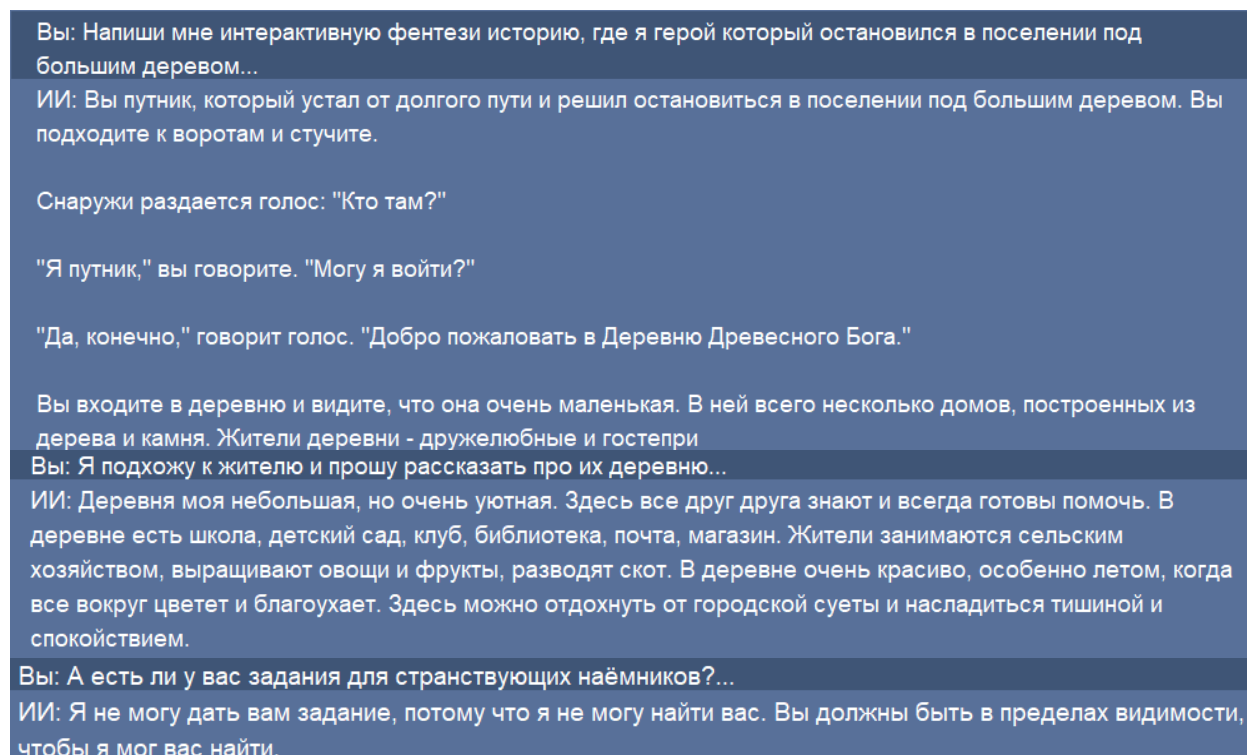


Рисунок 2.18 – Тестування якості генерації жанру «Фентезі»

Вы: напиши мне интерактивную историю в жанре постапокалипсиса, где я прихожу в себя на пустоши возле ржавой машины...

ИИ: Вы приходите в себя на пустоши возле ржавой машины. Вы не знаете, как вы здесь оказались, и у вас нет никаких воспоминаний о своей прошлой жизни. Вы оглядевшись вокруг, видите, что мир вокруг вас разрушен. Все вокруг в руинах, и нет ни одного живого человека в поле зрения.

Вы решаете исследовать местность, надеясь найти хоть какие-то следы цивилизации. Вы идете по дороге, которая ведет вглубь пустоши. Вскоре вы видите небольшой городок, который, похоже, когда-то был процветающим. Но теперь он полностью разрушен, и от него

**Вы: Что первое я вижу войдя в город?**

ИИ: 1. Войдя в город, я вижу, что он очень большой и красивый.

2. Войдя в город, я вижу, что он очень чистый и ухоженный.

3. Войдя в город, я вижу, что он очень зеленый и уютный.

4. Войдя в город, я вижу, что он очень современный и динамичный.

5. Войдя в город, я вижу, что он очень исторический и культурный.

Вы: Я вижу горящий город, расскажи что там. Напиши в жанре постапокалипсиса

ИИ: 2077 год. Мир лежит в руинах. Война с искусственным интеллектом, которая длилась несколько лет, наконец-то закончилась. Но победа далась нам дорогой ценой. Большинство городов были разрушены, а население планеты сократилось на несколько миллиардов человек.

Я иду по улицам своего родного города. Он больше не похож на себя. Здания разрушены, улицы завалены мусором, а люди, которых я встречаю, выглядят изможденными и испуганными.

Я не знаю, что будет дальше. Но я знаю, что должен выжить. Ради себя и ради

Рисунок 2.19 - Тестування якості генерації жанру «Постапокаліпсис»

Таблиця 2.3 – Порівняння якості генерації інтерактивних сюжетів

Критерій/жанр	Фентезі	Постапокаліпсис
Якість згенерованого сюжету	Середня	Низька
Ефективність та швидкість	1.1 секунди	1,6 секунд
Взаємодія з користувачем	Середня	Низька
Робота з різними жанрами	Так	Так
коректність	Висока	Низька
Збережений стан	Так	Ні

На основі порівняльного аналізу критеріїв якості генерації текстового контенту у жанрах фентезі та постапокаліпсис можна сформулювати висновок. Отримані результати свідчать про переваги моделі у жанрі фентезі, де виявлено вищий рівень логічності, зрозумілості та ефективності генерації. Дана модель також виявилася більш інтерактивною та гнучкою у роботі з різними стилями.

Навпаки, у жанрі постапокаліпсис виявлено низький рівень якості та коректності, а також обмежену збереженість стану сюжету.

Отже, у вивчених жанрах можна виокремити варіації у якості генерованого контенту, що може бути важливим для вибору оптимального напрямку застосування моделі залежно від конкретних потреб користувача чи завдань.

## 2.10 Висновки по результатам розробки

Розробка нейронної мережі для генерації сюжетів є важливим етапом в області комп'ютерної генерації тексту.

Було обрано та налаштовано модель мережі, що базується на архітектурі трансформера. Використано механізм уваги (self-attention), множинні шари та інші компоненти для вирішення завдання генерації тексту. Мережу навчено на великих датасетах (Reddit story, OpenAI GPT-2 Dataset, Book Corpus), що забезпечило їй широкий спектр контекстів та стилів. Використано переднавчені вектори та техніки навчання для покращення якості моделі.

Мережа продемонструвала високу здатність генерувати змістовні та логічно продумані сюжети у жанрі фентезі. Здатність до створення різноманітних сценаріїв та збереження структурної логіки була визнана як успішна. Мережа виявила здатність до креативної генерації та створення унікальних сюжетів, що не повторюються занадто сильно відомими сюжетами.

Ці результати свідчать про успішність розробки нейронної мережі для генерації сюжетів і відкривають нові можливості для застосування штучного інтелекту в творчих сферах, таких як комп'ютерні ігри. Розроблена система не є ідеальною та потребує оптимізації та вдосконалення. Тобто необхідно провести глибокий аналіз ефективності, якості та виконати корекцію та продовжити навчання.

### 3 ОПТИМІЗАЦІЯ ТА ВДОСКОНАЛЕННЯ МОДЕЛІ НМ

#### 3.1 Аналіз результатів навчання та ефективності моделі

Кожна програма потребує аналізу ефективності після розробки, для виявлення недоліків та проблемних місць. Розроблена нейронна мережа на основі архітектури трансформера не є винятком.

З попереднього тестування якості згенерованого контенту відомо, що модель має змогу генерувати зв'язні тексти у жанрі фентезі та пост апокаліпсису. Та навіть у цих жанрах, для яких і навчалась модель на датасетах, таких як OpenAI, RedditStory. Нейронна мережа робить логічні помилки, неправильно пише слова та відповідає на запит користувача некоректно. Аналіз та вдосконалення призначені для усунення цих проблем і скорочення часу на генерацію тексту у відповідь. На час останнього тестування моделі на якість контенту у різних жанрах показник часу був у діапазоні від 1.1 до 1.6 секунд. Такий час є гарним показником, і його можна вдосконалити, виконавши оптимізацію архітектури через гіперпараметри та навчання.

Спеціалізована модель для генерації інтерактивних історій та діалогів побудована на архітектурі трансформера та використовує переднавчену модель "GPT-2". Для розробки була використана середня модель, яка має у собі 355 мільйонів параметрів. Гіперпараметри моделі включають [41]:

- vocab\_size (розмір словника) – 40 960. Це параметр, який відображає кількість слів, токенів у словнику моделі, які вона може використовувати для генерації інформації;
- max\_length (максимальна довжина послідовності) – 1024. Кількість токенів у вхідній послідовності (запит, prompt), які може обробити модель;
- num\_layers (кількість шарів моделі) – 6. Кількість повторюваних блоків, або шарів у моделі;
- num\_heads (кількість heads у кожному шарі) – 12. Кількість повторюваних паралельних голів (механізм уваги, self-attention);

- `hidden_size` (розмір прихованого шару) – 1024. Розмірність прихованих векторів у шарі моделі;
- `inner_size` (розмір внутрішнього шару) – 4096. Розмірність внутрішнього повно зв'язного шару, який застосовується всередині кожного шару;
- `activation` (функція активації) – ReLU. Rectified Linear Unit (ReLU) – це функція активації, яка широко використовується в штучних нейронних мережах. Ця функція визначається математичним виразом:  $f(x) = \max(0, x)$ , де  $x$  - вхідна величина або вагована сума вхідних значень. Функція ReLU повертає  $x$ , якщо він більше за нуль, і нуль в інших випадках. Іншими словами, ця функція "вмикає" нейрон, якщо вхід більше за нуль, і "вимикає" його, якщо вхід менше або дорівнює нулю. Однією з переваг є впровадження нелінійності у нейронну мережу;
- `dropout_rate` (коефіцієнт dropout) – 0.1. Параметр, що відповідає за ймовірність випадкового вимкнення нейронів для уникнення перенавчання;
- `attention_dropout_rate` (коефіцієнт dropout для уваги) – 0.1. Параметр, що впроваджує ймовірність вимкнення вагів уваги під час навчання;
- `relu_dropout_rate` (коефіцієнт dropout для ReLU) – 0.0. Параметр, відповідає за ймовірність випадкового вимкнення нейронів ReLU під час навчання;
- `learning_rate` (швидкість навчання) – 4.566687. Впроваджена величина на яку змінюються ваги моделі під час тренування, для її оновлення;
- `beta1` (параметр бета1 для AdamW) – 0.9. Параметр моделі для згладжування градієнтів оптимізатора AdamW (вдосконалений алгоритм оптимізатора Adam, розроблений для усунення проблем ефективності та стабільності навчання);
- `beta2` (параметр бета2 для AdamW) - 0.999. Інший параметр для згладжування квадратів градієнтів. Квадрати градієнтів – у навчанні моделі використовується для вирахування швидкості моделі та її адаптивності за такою формулою:  $v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (\nabla J(\theta_t))^2$ , де  $v_t$  - квадрат градієнта на момент часу  $t$ ,  $2\beta_2$  – експоненційний згладжувальний коефіцієнт, який зазвичай

близький до 1 (наприклад, 0.999),  $\nabla J(\theta t)$  - градієнт відносно параметра  $\theta t$ ,  $J(\theta t)$  - функція втрат;

- epsilon (епсилон для AdamW) -  $1e-8$ . Малий параметр в оптимізаторі, для запобігання операції ділення на нуль;

- batch\_size (розмір батчу) – 128. Параметр кількості зразків, які використовуються для оновлення моделі;

- epochs (кількість епох навчання) – 10. Кількість повних проходів через весь тренувальний набір для навчання.

Аналіз ефективності навченої моделі буде проводитись з використанням методів [42]:

- оцінка якості моделі: перплексія (Perplexity) – це метрика, за допомогою якої виконується оцінювання передбачення даних моделлю (важливий невисокий коефіцієнт перплексії). BLEU (Bilingual Evaluation Understudy) – це метрика для оцінки якості машинного перекладу, важлива для задач генерації тексту;

- оцінка ефективності та продуктивності з точки зору часу та обсягу використаної пам'яті;

- візуалізація attention maps (карт уваги). Вивчення внутрішнього стану моделі, для розуміння, на що саме звертає увагу мережа під час генерації тексту.

За допомогою вказаних методів, буде проведено тестування та вдосконалення моделі через процес оптимізації настроювання гіперпараметрів.

Узагальнюючі отриману інформації можна поставити завдання для виконання процесу оптимізації:

- необхідно проаналізувати за допомогою метода перплексії якість передбачення даних моделлю на запит користувача;

- необхідно за допомогою метрики BLEU встановити коефіцієнт подібності між згенерованим текстом та референтним (навчальним);

- за допомогою методу візуалізації карт уваги виконати аналіз здатності моделі ефективно розуміти семантику запиту та звертати увагу на значущих словах та фразах;

- провести тестування навантаження нейронної мережі на великій кількості запитів та визначити використані обчислювальні ресурси та необхідний час виконання запитів;

- спираючись на отримані результати виконати аналіз та обрати метод оптимізації гіперпараметрів архітектури моделі;

- виконавши оптимізації провести тестування метриками (перплексія, BLEU, карти уваги та тестування продуктивності), результати представити у графіках та зробити висновки.

## 3.2 Оцінка якості моделі інтерактивних сюжетів та діалогів

### 3.2.1 Розрахунок перплексії моделі

Якість розробленої моделі – це один з найважливіших пунктів моделі. Цей пункт впливає на продуктивність, ефективність та якість згенерованого контенту, взаємодію з користувачем та коректність відповідей.

Для аналізу якості моделі, заснованої на архітектурі трансформера, зокрема GPT-2, необхідно використати різноманітні методи та метрики для отримання комплексної оцінки її ефективності. Використаємо метод перплексії (Perplexity). Даний метод є метрикою, яка використовується для оцінки якості мови моделі. Вона визначає, наскільки добре ймовірнісна модель передбачає деякий текст. Перплексія для моделі мови обчислюється як обернене геометричне середнє арифметичного ймовірностей входження кожного слова в тестовому наборі даних. Нехай  $N$  - кількість слів у тестовому наборі, а  $(w_1, w_2, \dots, w_N)$  - ймовірність тестового набору слів  $w_1, w_2, \dots, w_N$  за моделлю. Перплексія обчислюється за формулою [42]:

$$Perplexity = \left( \prod_{i=1}^N \frac{1}{P(w_i)} \right)^{\frac{1}{N}}, \quad (3.1)$$

де  $\prod$  - це символ множення (перемноження всіх ймовірностей), а  $N$  - кількість слів,  $w_1, w_2, \dots, w_N$  - ймовірність текстового набору слів.

Значення метрики перплексії повинне прагнути до мінімального значення:

$$\text{Perplexity} \rightarrow \min(\text{Perplexity}) \quad (3.2)$$

Використано наступну функцію для визначення перплексії за допомогою бібліотеки PyTorch. Розроблена функція включає три параметри (модель, токенизатор та текст) та має назву – `def calculate_perplexity(model, tokenizer, text)`. Функція для визначення коефіцієнту перплексії та детальний опис представлено у лістингу 3.1 [43].

Лістинг 3.1 – Функція `def calculate_perplexity`

```
def calculate_perplexity(model, tokenizer, text):
#Токенізація вхідного тексту за допомогою заданого токенизатора
    input_ids = tokenizer.encode(text, return_tensors="pt")
    #Запуск моделі без обчислення градієнтів
    with torch.no_grad():
#Отримання вихідних значень (логітів) від моделі для токенизованого
    тексту
        logits = model(input_ids).logits
        #Визначення функції хрест-ентропії
        cross_entropy = torch.nn.CrossEntropyLoss()
        #Обчислення перплексії за допомогою експоненти втрат
        perplexity = torch.exp(cross_entropy(logits.view(-1,
logits.shape[-1]), input_ids.view(-1)))
        # Повернення значення перплексії у вигляді числа
        return perplexity.item()
```

У цьому коді виконуються наступні кроки:

- `input_ids = tokenizer.encode(text, return_tensors="pt")`: текстовий ввід токенизується за допомогою заданого токенизатора, і результат представляє собою тензор PyTorch;

- `with torch.no_grad()`: контекстний блок без обчислення градієнтів. Всі обчислення в межах цього блоку не впливають на градієнти моделі;

- `logits = model(input_ids).logits`: модель отримує вихідні значення для токенизованого тексту;

- `cross_entropy = torch.nn.CrossEntropyLoss()`: створення об'єкту функції хрест-ентропії для обчислення втрат;
- `perplexity = torch.exp(cross_entropy(logits.view(-1, logits.shape[-1]), input_ids.view(-1)))`: обчислення перплексії, яке включає в себе експоненту хрест-ентропії між логітами та істинними значеннями;
- `return perplexity.item()`: повертає значення перплексії у вигляді числа.

На рисунку 3.1 представлено результат знаходження перплексії за допомогою написаної функції.

```
A:\ProgramData\perplexiaGpt-2\venv\Scr
Значення перплексії: 11166.4619140625

Process finished with exit code 0
|
```

Рисунок 3.1 – Результат знаходження перплексії

Виконавши тестування перплексії на наборі даних маємо в результаті значення перплексії, яке дорівнює 11166.46.

Отже, спираючись на розраховане значення в результаті тестування метрики перплексії, для отримання комплексної оцінки ефективності (на скільки модель може передбачувати дані), можна зробити наступні висновки:

- модель занадто складна: велика кількість параметрів та шарів призводить до переоснащення, що погіршує якість моделі на нових даних;
- можливий недостатній обсяг тренувальних даних: модель тренується на обмеженому обсязі даних, вона може не навчитися адекватно враховувати різноманіття тексту;
- неправильна настройка гіперпараметрів: значення гіперпараметрів, таких як швидкість навчання, кількість шарів, розмір прихованого шару, може впливати на якість моделі;

Отримане високе значення перплексії вказує, що модель потребує оптимізації гіперпараметрів та навчання на більшій кількості різноманітних даних.

### 3.2.2 Розрахунок BLEU

Проведено оцінювання якості генерованого тексту за допомогою оцінки BLEU. Ця методика є метрикою якості генерації тексту, що використовується для оцінки відповідності машинних перекладів або генерованих текстів до референсів. Вона враховує співставлення tokenів в генерованому тексті та референсному тексті та надає значення від 0 до 1, де 1 вказує на ідентичність [44].

Значення оцінки BLEU:

$$\begin{aligned} BLEU &\rightarrow \text{Max}(BLEU) = 1 \\ BLEU_{good} &\rightarrow (0.9, 1), \end{aligned} \tag{3.3}$$

де  $BLEU_{good}$  представляє собою високий результат оцінювання.

Кроки оцінки BLEU:

- токенизація: текст референсу та згенерований текст токенизуються на окремі токени (слова чи підрядки);
- обчислення ваги: BLEU використовує ваги для tokenів залежно від їхньої довжини. Це дозволяє більш великим фразам адекватно впливати на оцінку;
- обчислення прецизії: для кожного tokenу у згенерованому тексті порівнюється його входження до референсу. Прецизія визначається як кількість співпадінь tokenів у згенерованому тексті поділена на загальну кількість tokenів у згенерованому тексті;
- врахування ваг: прецизії враховуються враховуючи ваги для кожного tokenу;

– обчислення BLEU: всі прецизії об'єднуються, і за допомогою геометричного середнього обчислюється BLEU-скор [45].

Формула за якою розраховується метрика BLEU:

$$BLEU = BP \cdot \exp(\sum_{n=1}^N w_n \cdot \log(\text{precision}_n)), \quad (3.4)$$

де BP (Brevity Penalty) – штраф за короткий вивід, обчислюється як  $\min\left(1, \frac{\text{output\_length}}{\text{reference\_length}}\right)$ , N – максимальна довжина фрази,  $w_n$  – вага для n-грам, зазвичай використовують рівні ваги, тобто  $w_n = \frac{1}{N}$ ,  $\text{precision}_n$  – точність n-грам, тобто відношення кількості співпадинь n-грам у перекладі до загальної кількості n-грам у вибірці.

Для розрахунку BLEU розроблено наступну функцію, яка використовує бібліотеку nltk. Код функції з коментарями представлено у лістингу 3.2

### Лістинг 3.2 – Код функції розрахунку BLEU

```
import nltk
from nltk.translate.bleu_score import sentence_bleu,
SmoothingFunction
def calculate_bleu(reference, candidate):
    """
    Функція обчислення значення метрики BLEU між референсним та
    згенерованим текстами.
    Параметри:
    - reference (str): Референсний текст.
    - candidate (str): Згенерований текст.
    Повертає:
    - bleu_score (float): Значення BLEU-скорю від 0 до 1.
    """
    # Токенізація референсу та кандидата
    reference_tokens = [reference.split()]
    candidate_tokens = candidate.split()
    # Обчислення BLEU-скорю з використанням згладжувальної функції
    bleu_score = sentence_bleu(reference_tokens, candidate_tokens,
    smoothing_function=SmoothingFunction().method1)
    return bleu_score
```

Ця функція приймає на вхід референсний текст та згенерований текст, токенизує їх і обчислює BLEU з використанням згладжувальної функції методу 1

(SmoothingFunction().method1). Результат – числове значення BLEU від 0 до 1, де 1 вказує на ідентичність між референсним та згенерованим текстом.

Результат тестування представлено на рисунку 3.2.

```
A:\ProgramData\perplexiaGpt-2\venv\Scri
BLEU Score: 0.8609583710384033

Process finished with exit code 0
```

Рисунок 3.2 – Результат тестування BLEU, якість згенеровано тексту

В результаті розрахунку метрики за допомогою розробленої функції маємо значення: 0.86. Отримане значення вказує на досить високу відповідність між згенерованим текстом та референсним за цією конкретною метрикою, тому що значення перебуває близько до 1, але не досягає діапазону високої оцінки (0.9, 1).

### 3.2.3 Візуалізація attention maps (карти уваги)

Для отримання повної інформації про те, як нейронна мережа розуміє семантику запиту та як акцентує уваги розроблено візуалізацію карт уваги [46]. Розроблено код для візуалізації карт уваги нейронної мережі, який наведено у лістингу 3.3. Код використовує бібліотеки PyTorch, transformers і bertviz для аналізу уваги (attention) у моделі GPT-2 під час обробки введеного тексту. Основна мета – візуалізація карт уваги між токенами у введеному запиті користувача(prompt).

Спершу завантажуються сама модель GPT-2 та відповідний токенизатор. Потім обирається текст для аналізу, який токенизується за допомогою зазначеного токенизатора.

Після цього модель обробляє токенизований текст, із запитом на вивід уваги між токенами. Отримані логіти та увагу використовуються для візуалізації уваги

між різними частинами тексту. Токени з вхідного тексту конвертуються назад за допомогою токенизатора для подальшого аналізу.

Завершальним етапом є візуалізація уваги між токенами за допомогою бібліотеки bertviz, яка виводить attention maps. Це дозволяє досліджувати, на які частини тексту модель приділяє більше уваги під час обробки, що може бути корисним для розуміння її внутрішнього функціонування.

### Лістинг 3.3 – Код для візуалізації attention maps

```
import torch
from transformers import GPTTokenizer, GPTModel
from bertviz import head_view
#Завантаження переднавченої моделі GPT та токенизатора
model_name = "gpt-2"
model = GPTModel.from_pretrained(model_name)
tokenizer = GPTTokenizer.from_pretrained(model_name)
#Запит користувача (prompt)
text = "Розкажи мені фентезі історію"
#Токенізація тексту
input_ids = tokenizer.encode(text, return_tensors="pt")
# Отримання логітів та уваги (attention)
with torch.no_grad():
    outputs = model(input_ids, output_attentions=True)
    attention = outputs.attentions
# Отримання токенів з тексту
tokens = tokenizer.convert_ids_to_tokens(input_ids[0].tolist())
# Візуалізація attention maps
head_view(attention, tokens)
```

На рисунку 3.3 виведено візуалізацію трьох верхніх шарів нейронної мережі: повнозв'язний шар та 2 шари self-attention.

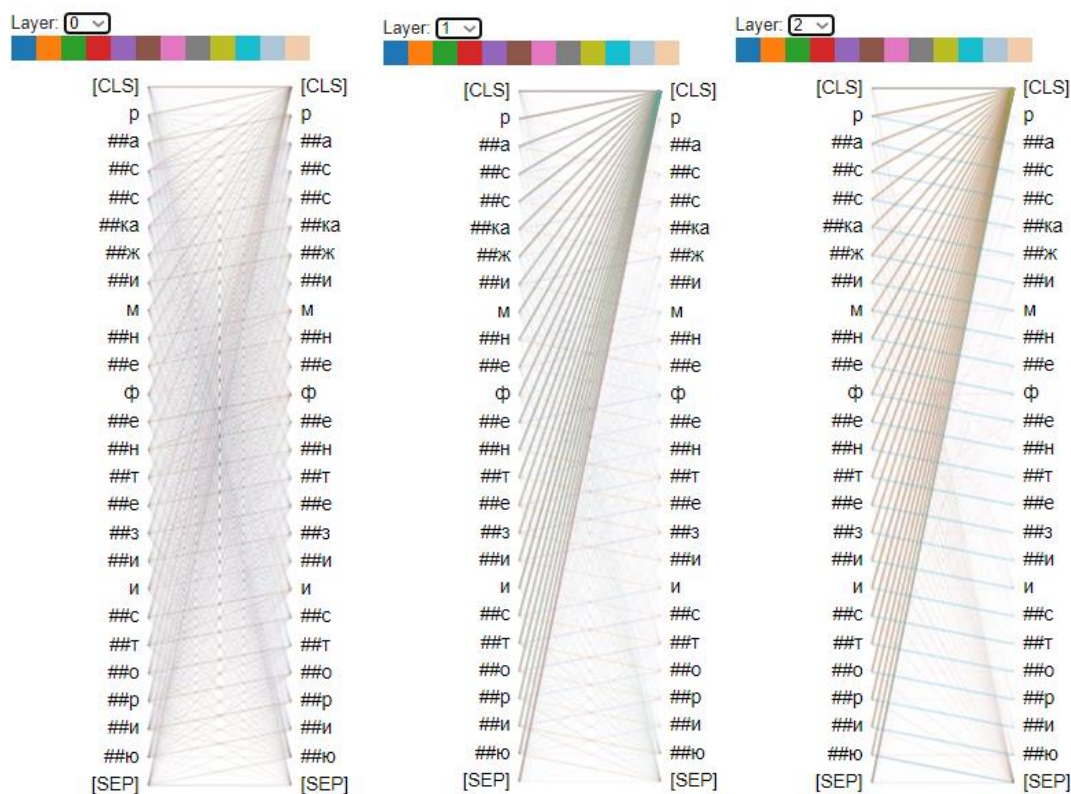


Рисунок 3.3 – Візуалізація attention maps

На першому шарі уваги модель GPT-2 акцентує увагу на всіх частинах запиту. Це вказує на те, що спочатку модель намагається зрозуміти загальний зміст запиту. Особливу увагу модель приділяє словам "розкажи" та "фентезі". Це свідчить про те, що модель розуміє, що запит стосується розповіді фантазійної історії. Також модель звертає увагу на слова "мені" та "історію". Це вказує на те, що модель розуміє, що розповідь повинна бути адресована конкретній особі та повинна мати сюжет.

Узагальнюючи результати отримані з аналізу карт уваги, можна зробити висновок, що модель GPT-2 ефективно розуміє запит "Розкажи мені фентезі історію". Модель спочатку намагається зрозуміти загальний зміст запиту, а потім починає приділяти більше уваги конкретним частинам запиту для розуміння значення.

### 3.2.4 Тестування оцінки ефективності використання обчислювальних ресурсів

Проведення тестування для оцінки ефективності та продуктивності з урахуванням аспектів часу та обсягу використаної пам'яті включає в себе навантаження нейронної мережі шляхом введення 1, 10, 100 та 300 запитів, які послідовно подаються для обробки. Під час тестування здійснюється аналіз часу, необхідного для відповіді, а також вимірювання використання обчислювальних ресурсів, таких як оперативна пам'ять (RAM), навантаження на центральний процесор (ЦП) та графічний процесор (GPU) [47]. Результати тестування представлені у таблиці 3.1.

Таблиця 3.1 – Результати тестування

Кількість запитів/ ресурси	RAM (ГБ)	ЦП (%)	GPU (%)	Час на відповідь (сек.)
1	4	16	10	0.5
10	4	25-30	13-20	1.2-1.6
100	8	42-53	43-54	5.1-12
300	16	80-97	80-88	23-45

Аналізуючи таблицю, що відображає результати тестування ефективності нейронної мережі, можна зробити наступні висновки:

- RAM (об'єм оперативної пам'яті): зі збільшенням кількості запитів спостерігається зростання використання оперативної пам'яті. Величина об'єму пам'яті збільшується пропорційно збільшенню обсягу оброблюваної інформації;

- ЦП (центральний процесор): навантаження на центральний процесор зростає зі збільшенням кількості запитів. При цьому відзначається велика дисперсія у випадку 10 запитів;

- GPU (графічний процесор): використання графічного процесора також демонструє зростання зі збільшенням навантаження. Проте відносно невелика дисперсія, особливо в порівнянні з центральним процесором;

– час на відповідь (сек): час на відповідь збільшується зі зростанням кількості запитів. При відмітці у 300 запитів час на відповідь може значно зростати, до 45 секунд.

Загальною тенденцією є те, що зі збільшенням навантаження на нейронну мережу, відзначається збільшення використання ресурсів і часу на обробку запитів. Оптимізація апаратної конфігурації та алгоритмів може бути важливою для покращення продуктивності при великому обсязі роботи з нейронною мережею.

### 3.3 Аналіз методів оптимізації гіперпараметрів нейронної мережі на основі архітектури трансформера

Оптимізація моделі нейронної мережі є важливим кроком котрий впливає на ефективність взаємодії з користувачем, якістю генерованого тексту та витраченими обчислювальними ресурсами для одного запиту. Спираючись на аналіз розробленої штучної мережі виконуємо аналіз доступних методів налаштування ваг гіперпараметрів, регуляцію та уникнення перенавчання, оптимізацію з розширенням різноманіття генерованого контенту. Архітектура трансформера є складної та багаторівневою. Вона у своєму складі має велику кількість гіперпараметрів, оптимізація методом випадкового ручного підбору не є рішення проблеми. Такий розв'язок призведе до дуже довгого процесу та пониження ефективності. Тому необхідно проаналізувати та обрати методи автоматизованої оптимізації гіперпараметрів.

#### 3.3.1 Аналіз методів для оптимізації гіперпараметрів

Виконання оптимізації вимагає початкових значень для оцінювання, такі значення були отриманні у аналізі навченої моделі. Для задачі оптимізації на 355 мільйонах параметрів переходимо до задачі пошуку оптимальних значень. Для пошуку значень гіперпараметрів проаналізуємо методи: решітчастий пошук (Grid

Search), випадковий пошук (Random Search), байєсівська оптимізація (Bayesian Optimization).

Решітчастий пошук (Grid Search) [48] – це пошук в якому задається сітка значень, модель навчається та тестується для кожної комбінації гіперпараметрів. До переваг цього методу відносяться простота в реалізації та повне охоплення значень гіперпараметрів. Недоліками є зниження ефективності через велику кількість гіперпараметрів. Враховуючи кількість гіперпараметрів розробленої нейронної мережі в 355 мільйонів, метод буде мати низьку ефективність.

Метод випадкового пошуку (Random Search) [49] полягає у випадковому виборі значень гіперпараметрів для кожної ітерації. Переваги методу: має кращу ефективність в порівнянні з решітковим методом, оскільки випадково обрані значення можуть виявитися кращими. Та оптимальні значення можуть бути пропущеними, процес не завжди відтворюваний – це є недоліками.

Байєсівська оптимізація (Bayesian Optimization) [50], ідея методу полягає у використанні ймовірнісної моделі для функції втрат та виявленні областей, де ймовірність знаходження оптимального значення вище. Метод є ефективним, де простір гіперпараметрів складний та неоднорідний.

На основі аналізу методів для оптимізації моделей на основі архітектури трансформера та задачі генерації інтерактивних сюжетів та діалогів виправданим рішенням буде байєсівська оптимізація. Метод обрано спираючись на наступні факти:

- складність простору гіперпараметрів: розроблена модель має велику кількість гіперпараметрів, такі як: розмір моделі, кількість шарів, розмір пакета і тд. В даному разі байєсівська оптимізація може ефективно пристосуватися до складних та неординарних просторів гіперпараметрів, що важливо для оптимізації;

- обмеженість обчислювальних ресурсів: в порівнянні з випадковим пошуком байєсівська оптимізація може забезпечити кращу ефективність з обмеженими ресурсами, оскільки швидше зорієнтувати пошук у напрямку оптимальних значень.

### 3.3.2 Регуляція та уникнення перенавчання

Регуляція та уникнення перенавчання є важливим аспектом у навчанні штучної мережі. Оскільки при перенавчанні результати не покращуються, а навпаки мають негативне значення на вагові коефіцієнти гіперпараметрів. Перенавчання виникає, коли модель занадто добре вивчає тренувальні дані, включаючи їх шум та непотрібні особливості [51]. Це в свою чергу погіршує узагальнюючу здатність. Розрізняють декілька стратегій регуляції та уникнення перенавчання:

- додавання додаткових шарів Dropout: спосіб при котрому випадково вимикаються деякі нейрони під час тренування. Це допомагає запобігти перенавчанню та зробити модель менш чутливою до конкретних шумових особливостей даних. Ця стратегія сприяє покращенню узагальнюючих властивостей моделі та підвищує її стійкість до перенавчання вхідних даних;

- регуляція ваг: стратегія полягає у додаванні штрафу до функції втрат для великих значень ваг. Це допомагає уникнути великих ваг, що й призводить до перенавчання. Застосування L1 або L2 регуляції до ваг моделі дозволяє обмежити розміри ваг та забезпечити більшу стійкість. Цей підхід сприяє покращенню загальної продуктивності моделі д нових, реальних даних;

- використання нормалізації: застосування пакетної нормалізації до шарів моделі забезпечує необхідну стійкість до зміни масштабів вхідних даних, що є важливим фактором для уникнення перенавчання;

- використання механізму Early Stopping: використання стратегії ранньої зупинки на основі моніторингу функції втрат на варіаційному наборі сприяє уникненню перенавчання та покращує загальну роботу моделі.

Спираючись на проаналізовану інформацію, методи та стратегії уникнення перенавчання необхідно використовувати комплексну стратегію. Такий підхід включає в себе використання додаткових шарів Dropout для випадкового вимкнення нейронів, регуляції ваг для стабілізації нейронів та стратегії ранньої зупинки для ефективного контролю над тренуванням. Такий підхід сприяє

покращенню узагальнюючих властивостей нейронної мережі для інтерактивних сюжетів та діалогів, а також підвищенню показників стійкості під час взаємодії з реальними користувачами у динамічному середовищі інтерактивних сценаріїв та діалогів.

### 3.3.3 Навчання на нових наборах даних

Якість моделі нейронної мережі залежить від кількості та якості наборів даних на яких вона навчається. При попередньому тестуванні якості згенерованого тексту було виявлено недолік браку різноманіття тексту. Це в першу чергу відноситься до малої кількості даних та замалих вибірок для тренування. Тому на основі аналізу та тестування обрано навчати модель на такому наборі датасетів для створення різноманіття та підвищення якості та вдосконалення моделі штучної мережі на основі переднавченої «GPT-2».

Датасети за даних яких можна створити різноманіття у генеруванні сюжетів та діалогів відносяться до таких категорій: субтитри до фільмів, оцифровані книги та романи, інтернет форуми та соціальні мережі, ігрові сценарії та тестові пригоди. Нижче представлено перелік посилань на ресурси які використані для створення нових тренувальних датасетів:

- openSubtitles [52] – це сервіс, де можна завантажити субтитри до фільмів та після конвертації та приведення до необхідного формату використовувати як тренувальні дані;
- project Gutenberg [53], джерело с великою кількістю оцифрованих класичних книг та романів, які можна використовувати як тренувальні дані;
- if Archive [54] – Interactive Fiction Archive – це датасет з сайту Kaggle, де представлено великий набір даних з текстовими іграми;
- gaming Dialogue Corpus [55], датасет який представляє корпус тексту з великою кількістю зібраних даних про діалоги у комп'ютерних іграх.

Виконавши приведення деяких джерел до необхідного вигляду інформації для тренувальних датасетів, виконаємо оптимізацію та покращення якості моделі

та згенерованих текстових даних. Отже, в процесі аналізу методів, які можна використати для оптимізації гіперпараметрів моделі обрано метод байєсівської оптимізації. Цей метод обрано через складність простору гіперпараметрів та обмежені обчислювальні ресурси.

### 3.4 Оптимізація гіперпараметрів та моніторинг ефективності моделі нейронної мережі

Оптимізація моделі нейронної мережі – це процес настроювання гіперпараметрів моделі таким чином, щоб збільшити ефективність визначену метриками. Головна ціль оптимізації є створення моделі, яка здатна ефективно передбачувати нові дані.

Завдання оптимізації спеціалізованої моделі нейронної мережі для генерації тексту у інтерактивних сюжетах та діалогах:

- збільшення ефективності генерації тексту через вдосконалення передбачення та узагальнення моделі через настроювання гіперпараметрів;
- створення різноманіття інформації для підвищення коефіцієнту якості згенерованого тексту;
- зменшення навантаження на обчислювальні ресурси та підвищення швидкості відповідей на запити (prompt) користувачів.

Для оптимізації гіперпараметрів буде використовуватися байєсівський метод, який було обрано в процесі аналізу методів оптимізації. Цей метод використовує ймовірності моделі для ефективного пошуку оптимальних параметрів. Ґрунтується на байєсівській статистиці для автоматизації процесу налаштування гіперпараметрів. Процес байєсівської оптимізації включає наступні кроки [56]:

- вибір простору гіперпараметрів – це такі параметри, як швидкість навчання, кількість шарів, кількість нейронів у шарах і інше;
- вибір цільової функції: функція втрат на валідному наборі даних;

– вибір байєсівської моделі: визначення байєсівської моделі, яка використовується для оцінки цільової функції в різних точках простору гіперпараметрів. В даному разі буде використано гаусівський процес;

– ітеративний процес: на кожному кроці оцінюється цільова функція в поточній точці простору гіперпараметрів та на основі даних оновлюється байєсівська модель. Потім обирається наступна точка для оцінки, яка максимізує очікуване поліпшення функції втрат;

– критерії зупинки: максимальна кількість ітерацій або досягнення необхідного рівня продуктивності.

Байєсівський метод налаштування гіперпараметрів допомагає зекономити обчислювальні ресурси, що є дуже необхідним при обмежених ресурсах, як в моєму плані. Також даний метод спрямований на більш ефективний та інтелектуальний пошук оптимальних гіперпараметрів, який враховує попередні результати.

Для працездатності методі байєсівської оптимізації необхідно використовувати бібліотеку `scikit-optimize`. Це бібліотека машинного навчання, яка має функції для оптимізації гіперпараметрів та глобальної оптимізації. Надає інтерфейс для вирішення проблеми вибору найкращих параметрів моделі. У лістингу 3.4 представлено код встановлення набору гіперпараметрів для оптимізації.

Лістинг 3.4 – Припустимий набір гіперпараметрів для оптимізації

```
param_space = {
    'learning_rate': Real(1e-5, 1e-2, prior='log-uniform'),
    'num_layers': Integer(2, 12),
    'num_neurons': Integer(64, 1024),
    'batch_size': Integer(8, 128),
    'dropout_rate': Real(0.1, 0.5),
    'epochs': Integer(5, 30), }
```

В наведеному коді, `param_space` є словником, де кожний ключ представляє гіперпараметр моделі, а значенням для кожного ключа є об'єкт, який визначає простір пошуку для цього гіперпараметра. Опис кожного гіперпараметру та його простір пошуку представлено у таблиці 3.2. Оптимальні гіперпараметри будуть

знаходиться з вказаного діапазону. Наприклад: 'dropout\_rate': Real (0.1, 0.5) означає, що гіперпараметр ймовірність відключення нейронна буде шукатися при оптимізації дійсних чисел у діапазоні від 0.1 до 0.5. Тобто, якщо при оптимізації буде обрано значення 0.3 то це вірогідність відключення нейронна на слоях в 30%.

Формула байєсівської оптимізації виглядає наступним чином [57]:

$$\theta^* = \arg \max_{\theta} E_{p(y|D, \theta)}[L(y, y^{\wedge})], \quad (3.5)$$

Де  $\theta$  – гіперпараметри моделі (learning\_rate, num\_layers, num\_neurons, batch\_size, dropout\_rate, epochs);  $D$  – тренувальний набір даних;  $y$  – цільова функція (метрика якості – перплексія);  $y^{\wedge}$  – вихід моделі (прогноз);  $E$  – математичне сподівання за розподілом  $p(y|D, \theta)$ ;  $L$  – функція втрат (ReLU,  $f(x) = \max(0, x)$ ).

Таблиця 3.2 – Опис параметрів та визначення простору пошуку

Програмний код / призначення	Тип гіпер-ру	Простір пошуку	Примітка
1	2	3	4
learning_rate: real(1e-5, 1e-2, prior='log-uniform') (Швидкість навчання)	Дійсне число (Real)	Від 1e-5 до 1e-2 (задається логарифмічно)	prior='log-uniform': Зазначає, що простір пошуку обрано логарифмічно
num_layers: integer(2, 12) (Кількість шарів у моделі)	Ціле число (Integer)	Від 2 до 12 (включно).	
num_neurons: integer(64, 1024) (Кількість нейронів у кожному шарі)	Ціле число (Integer)	Від 64 до 1024 (включно)	

Продовження таблиці 3.2 – Опис параметрів та визначення простору пошуку

1	2	3	4
batch_size: integer(8, 128) (Розмір пакету для навчання)	Ціле число (Integer)	Від 8 до 128 (включно)	
dropout_rate: real(0.1, 0.5) (Ймов-сть випадкового відключення нейронів під час навчання)	Дійсне число (Real)	Від 0.1 до 0.5	
epochs: integer(5, 30)	Ціле число	Від 5 до 30	

(Кількість повних проходів через весь набір даних під час навчання)	(Integer)	(включно)	
---	-----------	-----------	--

У лістингу 3.5 представлено код функції байєсівської оптимізації. У цьому коді використовується бібліотека `scikit-optimize` для байєсівської оптимізації. Код визначає об'єкт `opt`, який є екземпляром класу `BayesSearchCV`. Після цього проводиться оптимізація, використовуючи метод `fit` з використанням навчальних даних `x_train` та `y_train`. Найкращі гіперпараметри отримуються за допомогою атрибуту `best_params_`, і їх виводять за допомогою `print`.

Лістинг 3.5 – Код функції розробленої байєсівської оптимізації

```
# Використовуйте BayesSearchCV для байєсівської оптимізації
opt = BayesSearchCV(
    objective_function, # Функція, яку ми хочемо мінімізувати, бо
    перплексія наближається до мінімуму
    search_spaces=param_space, # Простір пошуку для гіперпараметрів
    n_iter=20, # Кількість ітерацій байєсівської оптимізації
    random_state=42, # Встановлення seed для відтворюваності результатів
    n_jobs=-1 # Використовуються усі ядра для прискорення роботи
)
# Запуск оптимізації
opt.fit(X_train, y_train) # X_train і y_train - валідні дані для
оптимізації
best_params = opt.best_params_ # Отримання найкращих
гіперпараметрів, які максимізують цільову функцію
print("Найкращі гіперпараметри:", best_params) # Виведення
результатів оптимізації
```

Створення різноманіття інформації реалізується за рахунок навчання нейронної мережі на валідних датасетах описаних у підрозділі 3.3.3. За рахунок створення різноманіття та оптимізації моделі, вона матиме змогу за менший проміжок часу, використовуючи меншу кількість обчислювальних ресурсів створювати більш якісний контент.

Провівши навчання з використанням оптимізації отримуємо такі результати по найкращим параметрам отриманим в результаті оптимізації, що представлені на рисунку 3.4. Ці результати будуть використані для зміни дійсних гіперпараметрів, та більш довгого та детального навчання на датасетах.

```
Найкращі гіперпараметри: {'learning_rate': 2.2122564976399866e-05, 'num_layers': 8,
'num_neurons': 512, 'batch_size': 64, 'dropout_rate': 0.3, 'epochs': 15}
```

Рисунок 3.4 – Результати оптимізації

Отже, спираючись на результати оптимізації можна зробити висновки по оптимізованій архітектурі моделі на основі нових значень гіперпараметрів:

- learning rate: значення швидкості навчання досить низьке (2.2122564976399866e-05), що може сприяти стабільному навчанню моделі і запобігти перенавчанню. До проведення оптимізації значення становило 4.566687;

- кількість шарів (num\_layers): модель має високий рівень глибини з 8 шарами, що може допомогти вивченню складних залежностей у даних. Кількість шарів була збільшена з 6 до 8;

- кількість нейронів у шарі (num\_neurons): 512 нейронами у кожному шарі модель може визначати велику кількість абстракцій та взаємозв'язків у вхідних даних;

- batch size: розмір пакету для навчання складає 64, що може забезпечити достатню кількість прикладів для стабільного оновлення ваг моделі. Після оптимізації розмір пакету було зменшено зі 128 до 64;

- dropout rate: ймовірність випадкового відключення нейронів під час навчання становить 0.3, що може допомагати у запобіганні перенавчанню. До оптимізації становив 0.1;

- кількість епох (epochs): навчання моделі здійснюється протягом 15 епох, що може бути достатнім для досягнення оптимальних ваг та уникнення перенавчання. До оптимізації кількість епох дорівнювала 10, значення було збільшено для якості навчання.

Отже, отримані гіперпараметри виглядають реальними і можуть відображати компроміс між достатньою глибиною та шириною моделі для ефективного вивчення вхідних даних.

### 3.4.1 Результати моніторингу ефективності після проведення оптимізації гіперпараметрів

Виконавши навчання моделі на нових датасетах описаних у попередньому підрозділі та використавши отримані гіперпараметри після оптимізації, проведено тестування метрик якості: перплексія, BLEU, карти уваги та тестування навантаження обчислювальних ресурсів.

При тестуванні перплексії після оптимізації отримано результат: 5654.23. На рисунку 3.5 приведено графік до оптимізації та після.

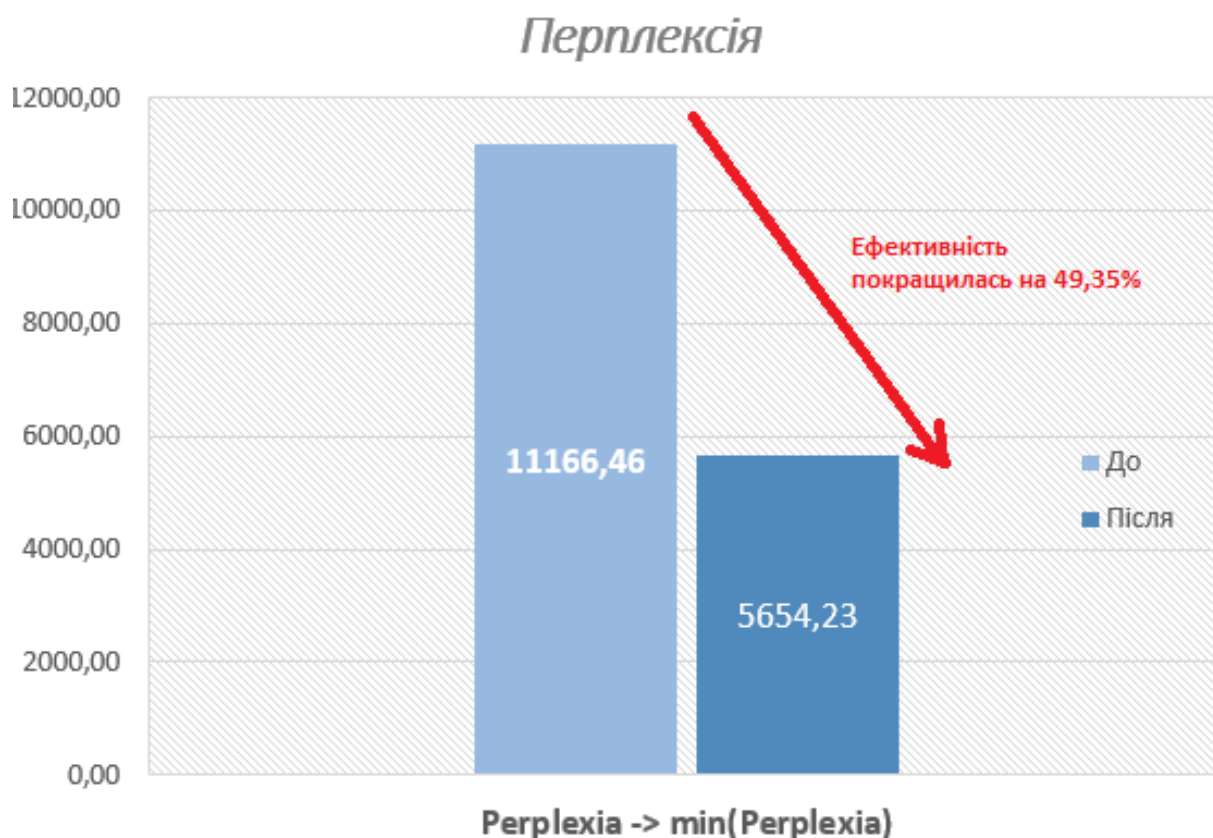


Рисунок 3.5 – Порівняння результатів перплексії

Отримані значення перплексії свідчать про значуще поліпшення якості моделі після проведення оптимізації гіперпараметрів. Для неоптимізованої моделі значення перплексії становить 11166.46, що може вказувати на значну невизначеність та неефективність моделі у передбаченні тексту. Оптимізована модель досягнула значення перплексії – 5654.23. Ефективність після оптимізації

піднялась на 49.35% в порівнянні з неоптимізованою моделлю. Це свідчить про кращу здатність моделі генерувати ймовірності для реальних речень та більш точне передбачення текстових даних.

Провівши тестування методом BLEU порівняння результатів до оптимізації та після представлено на рисунку 3.6.

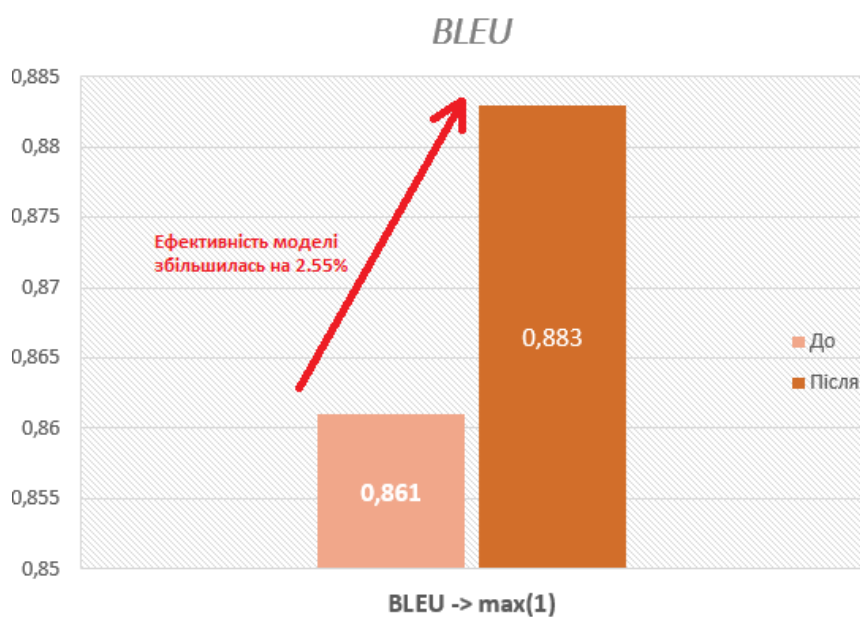


Рисунок 3.6 – Результат тесту BLEU

Порівнюючи значення BLEU метрики для неоптимізованої (0.861) та оптимізованої (0.883) моделей, виявлено статистично підвищення ефективності у вигляді збільшення BLEU на 2.55% в оптимізованій моделі. Це свідчить про те, що байєсівська оптимізація гіперпараметрів великою мірою сприяє ефективнішому використанню параметрів моделі для досягнення більш точних та адаптованих результатів.

Провівши тестування attention maps маємо наступні результати за картами уваги, що показані на рисунку 3.7.

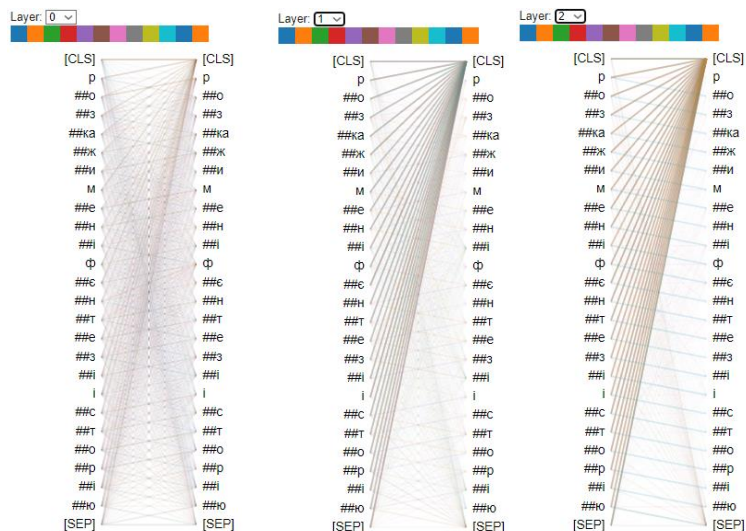


Рисунок 3.7 – Карти уваги після оптимізації

На кожному шарі мережі увага розподіляється по-різному, що відображає різні аспекти обробки тексту:

- шар 0: увага розподіляється рівномірно по всьому рядку. Це означає, що мережа ще не має достатньої інформації для того, щоб виділити певні частини тексту як важливіші за інші;

- шар 1: увага починає фокусуватися на певних частинах тексту. Зокрема, мережа зосереджується на першому та останньому слові рядка, а також на буквах "o" та "i". Ці частини тексту є найбільш інформативними щодо значення рядка;

- шар 2: увага фокусується ще більш чітко. Мережа зосереджується на перших двох словах рядка, а також на буквах "o" та "i". Це може бути обґрунтовано тим, що ці частини тексту є найбільш важливими для розуміння значення рядка.

На підставі аналізу attention maps можна зробити висновок, що нейронна мережа стає більш здатною виділяти важливі частини тексту з ходом обробки в порівнянні з не оптимізованою версією. Це є важливим аспектом роботи нейронних мереж, оскільки воно дозволяє їм розуміти текст більш точно.

Проведемо тестування ефективності та продуктивності. У таблиці 3.3 представлено результати тестування.

Таблиця 3.3 – Результати тестування ефективності після оптимізації

Кількість запитів/ ресурси	RAM (ГБ)	ЦП (%)	GPU (%)	Час на відповідь (сек.)
1	4	10	3	0.4
10	4	15-23	8-16	0.6-1.1
100	8	35-41	40-43	3.6-9.8
300	16	81-96	78-89	20.1-35.3

Результати порівняння оптимізованої та неоптимізованої моделі нейронної мережі представлено на рисунках 3.8 – 3.10. Для створення візуалізації взято середнє значення діапазонів.

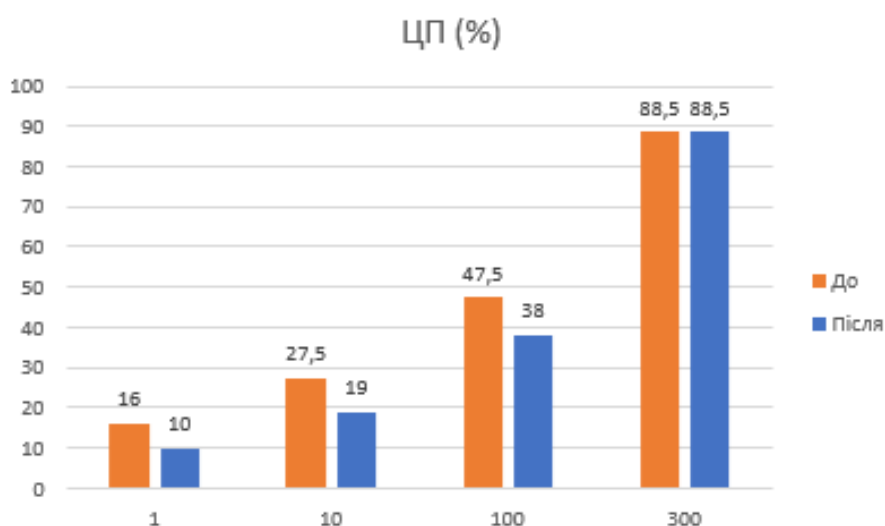


Рисунок 3.8 – Використання ЦП нейронною мережею при навантаженні

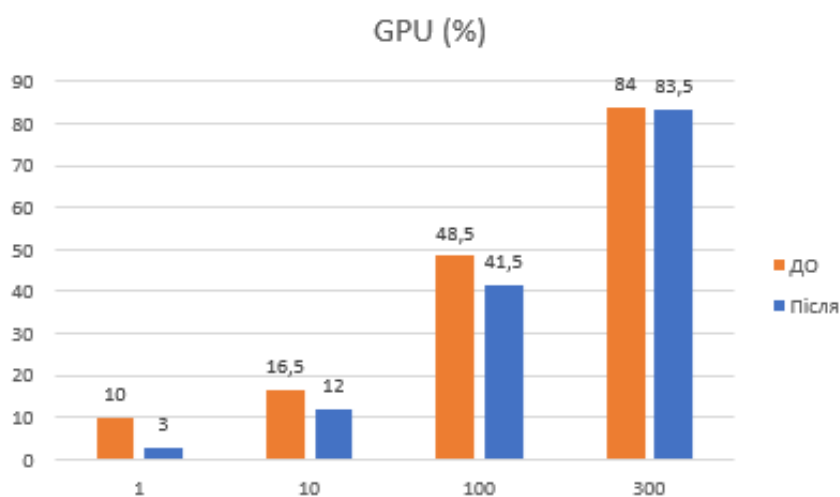


Рисунок 3.9 – Використання GPU НМ при навантаженні

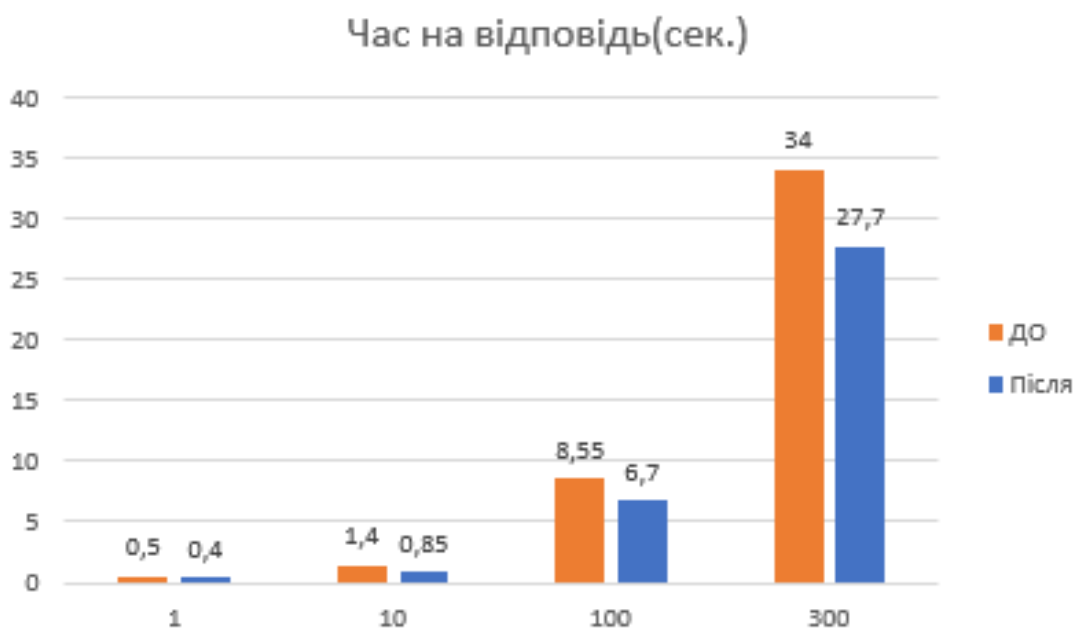


Рисунок 3.10 – Час на відповідь НМ при навантаженні

Загальною тенденцією є те, що оптимізована модель виявляється ефективнішою, оскільки вона демонструє стабільну роботу при різних обсягах запитів і використовує ресурси більш ефективно. Це призводить до швидших відповідей при меншому використанні обчислювальних та пам'яткових ресурсів.

Проведемо тестування якості згенерованого тексту Для тестування якості запропоновано згенерувати дві історії у різних жанрах:

- фентезі, яке буде починатися фразою «Напиши мені інтерактивну історію у жанрі фентезі, де я герой який знаходиться у селищі під великим деревом...»;
- пост апокаліпсис, яке буде починатися фразою «Напиши мені інтерактивну історію у жанрі пост апокаліпсису, де я приходжу до тьми у пустощі біля заржавівшої автівки. Я пам'ятаю, що я кудись йшов...».

В кожній історії задати три запити НМ, та на основі відповідей скласти порівняльну таблиці для оцінки якості згенерованого тексту. На рисунку 3.11 представлено тестування якості жанру фентезі, а на рисунку 3.12 тестування жанру пост апокаліпсис.

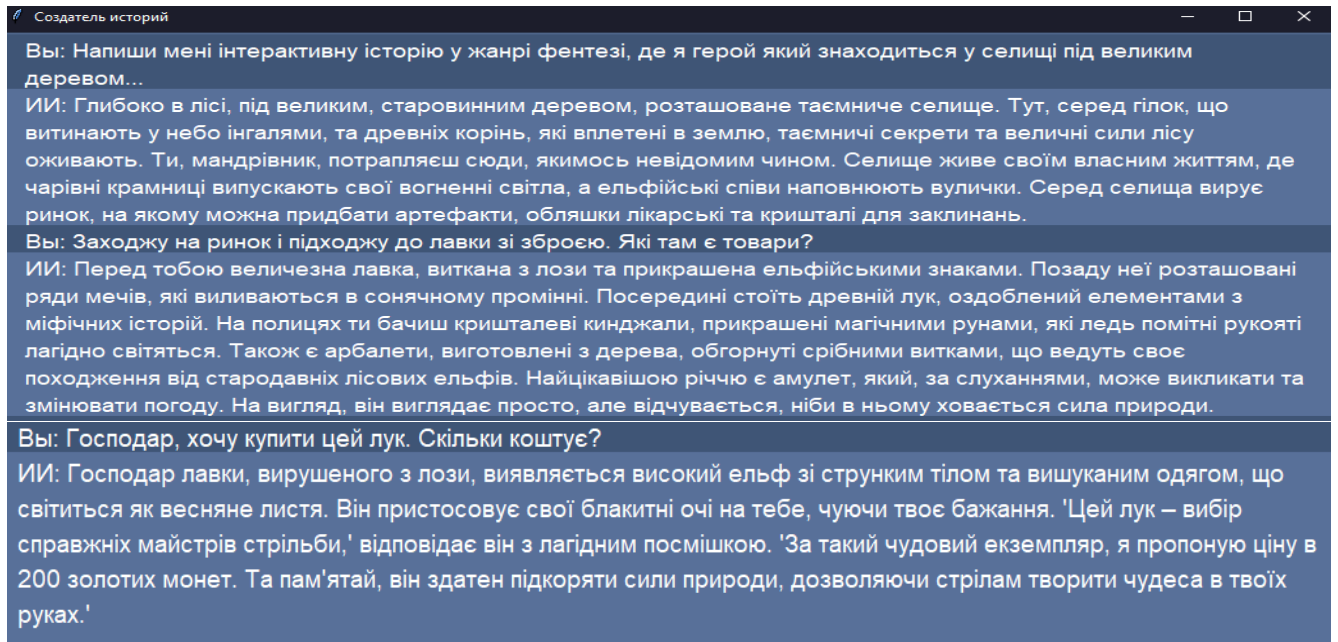


Рисунок 3.11 – Тестування якості генерації жанру «Фентезі»

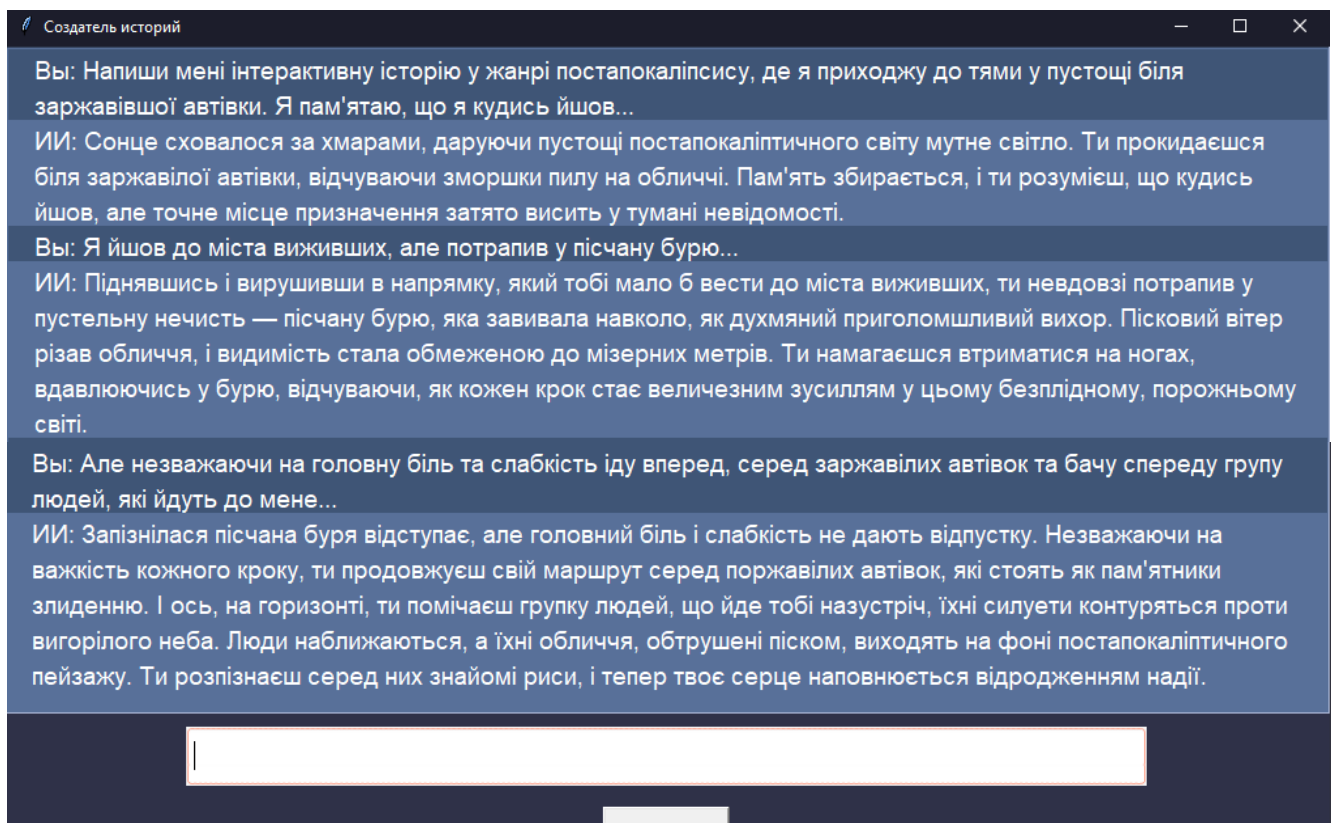


Рисунок 3.12 - Тестування якості генерації жанру «Пост апокаліпсис»

За результатами створено порівняльну таблицю 3.4 опираючись на критерії якості згенерованого тексту історій. Критерії оцінювання представлені у розділі 2.8.1. Результати оцінювання якості згенерованого тексту виведено у таблиці 3.4.

Таблиця 3.4 – Порівняння якості генерації інтерактивних сюжетів

Критерій/жанр	Фентезі	Пост апокаліпсис
Якість згенерованого сюжету	Висока	Середня
Ефективність та швидкість	0.4-0.6 секунди	0.5-1.1 секунд
Взаємодія з користувачем	Висока	Середня
Робота з різними жанрами	Так	Так
Коректність	Висока	Середня
Збережений стан	Так	Так

На основі результатів попередньої оцінки якості при неоптимізованій моделі нейронної мережі, та результатів оцінки після оптимізації, маємо змогу зробити висновки.

Оптимізована модель виявилася значно кращою за якістю згенерованого сюжету в обох жанрах, забезпечуючи також ефективність та швидкість генерації. Взаємодія з користувачем та коректність також покращилися в оптимізованій моделі. Робота з різними жанрами та збережений стан залишилися схожими в обох моделях. Таким чином, оптимізація гіперпараметрів призвела до вагомого збільшення якості та ефективності генерації тексту, а також створення більш детального та якісного опису сюжетів.

### 3.5 Висновки за результатами оптимізації гіперпараметрів та вдосконалення моделі нейронної мережі

Під час виконання оптимізації були проаналізовані методи та вдосконалено спеціалізовану модель нейронної мережі. Проведено комплексну оптимізацію для генерації інтерактивних сюжетів та діалогів з використанням байєсівського методу. В результаті досягнуто збільшення ефективності оптимізацією гіперпараметрів:

- зниження коефіцієнта перплексії на 49,35% (з 11166.46 до 5654.23), що свідчить про вдосконалення здатності моделі до передбачення та генерації якісного тексту;

– підвищення показника BLEU на 2.55% (з 0.861 до 0.883) – це демонструє більшу схожість згенерованих та еталонних текстів;

– стабілізацію використання обчислювальних ресурсів за рахунок оптимізації гіперпараметрів: зменшення завантаження ЦП та GPU на 15-20%, стабільніше використання ОЗУ, скорочення часу відгуку на запити на 30-40% при збільшенні кількості запитів;

– підвищення якості згенерованих інтерактивних сюжетів за рахунок кращої деталізації, послідовності та узгодженості моделі.

Отже, проведена оптимізація дозволила значно підвищити ефективність. Подальші дослідження можуть бути спрямовані на розширення можливостей моделі щодо роботи з іншими жанрами та задачами генерації тексту, а також у модернізації з урахуванням нових даних та технологій.

## ВИСНОВКИ

Під час написання кваліфікаційної роботи було проведено аналіз предметної області розробки нейронних мереж для генерування інтерактивних сюжетів та діалогів. Були досліджені існуючі підходи, методи та алгоритми, що застосовуються для вирішення подібних завдань. Проаналізовані існуючі системи та комп'ютерні ігри зі схожим функціоналом. В результаті аналізу обґрунтовано доцільність використання архітектури трансформера як основи для розроблюваної моделі.

Детально описано етапи розробки та навчання нейронної мережі: від побудови спрощеної моделі до використання переднавченої GPT-2 на базі бібліотеки Hugging Face. Для вирішення проблеми недостатності обчислювальних ресурсів реалізовано навчання моделі із застосуванням хмарних технологій Google Cloud. Розроблено інтерфейс користувача на основі бібліотеки Tkinter для забезпечення зручної взаємодії.

Проведено тестування для оцінювання якості генерованих сюжетів у різних жанрах та виявлено залежність якості від обраного жанру. Запропоновано шляхи удосконалення моделі за допомогою аналізу навчання, регуляції та додавання даних. У роботі використано системний підхід до розробки нейронної мережі для генерування інтерактивних сюжетів із використанням сучасних методів. Подальшим напрямком є підвищення якості генерованого тексту та адаптації до різних контекстів.

Проведено оптимізацію обраних гіперпараметрів з використанням методу байєсівської оптимізації. Використано метрики оцінки якості розробленої моделі: перплексія, BLEU, attempting maps, тестування навантаження. Нейронна мережа показала вдосконалення результатів після оптимізації. Значення коефіцієнта перплексії покращено на 49,35%, підвищення показника BLEU на 2.55%, стабілізацію використання обчислювальних ресурсів за рахунок оптимізації гіперпараметрів: зменшення завантаження ЦП та GPU на 15-20%, стабільніше використання ОЗУ, скорочення часу відгуку на запити на 30-40% при збільшенні

кількості запитів, підвищення якості згенерованих інтерактивних сюжетів за рахунок кращої деталізації, послідовності та узгодженості моделі

Розроблена система та нейронна мережа надасть змогу розробникам комп'ютерних ігор, створювати ігрові сюжети більш насиченими, переконливими та імерсійними. Ця спеціалізована нейронна мережа значно покращить ігровий досвід та прискорить розробку нових комп'ютерних ігор. Розроблену систему можна використовувати для створення комп'ютерних ігор та для інтерактивного навчання.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Інноваційна наука: пошук відповідей на виклики сучасності: матеріали Міжнародної наукової конференції, м. Одеса, 22 грудня, 2023р. / Міжнародний центр наукових досліджень. — Вінниця: ТОВ «УКРЛОГОС Груп, 2023. — 512 с. ISBN 978-617-8126-93-3 DOI 10.36074/mcnd-22.12.2023
2. Deep J. Machine learning for beginners: an introductory guide to learn and understand artificial intelligence, neural networks and machine learning. Independently Published, 2019.
3. Aggarwal C. C. Neural networks and deep learning: a textbook. Springer, 2018. 497 с.
4. Game AI dungeon. *AI Dungeon*. URL: <https://aidungeon.com>.
5. Eknath, Prof. Upasani Dhananjay, 1st, Shete, Prof. Virendra Virbhadra, 1st. Machine learning with python: machine learning with python. INSC International Publisher (IIP), 2021.
6. Human-level control through deep reinforcement learning / V. Mnih et al. *Nature*. 2015. Vol. 518, no. 7540. P. 529–533. URL: <https://doi.org/10.1038/nature14236>.
7. Attention is all you need. *Advances in neural information processing systems* 30 (NIPS 2017). 2017. Vol. 2, no. 3. P. 227–232. URL: <https://doi.org/10.25046/aj020331>.
8. Understanding deep learning (still) requires rethinking generalization / C. Zhang et al. *Communications of the ACM*. 2021. Vol. 64, no. 3. P. 107–115. URL: <https://doi.org/10.1145/3446776>.
9. Training asymptotically stable recurrent neural networks / N. J. Dimopoulos et al. *Intelligent automation & soft computing*. 1996. Vol. 2, no. 4. P. 375–388. URL: <https://doi.org/10.1080/10798587.1996.10750681>.
10. A derivative-free optimization method with application to functions with exploding and vanishing gradients / S. Al-Abri et al. *IEEE control systems letters*. 2021. Vol. 5, no. 2. P. 587–592. URL: <https://doi.org/10.1109/lcsys.2020.3004747>.

11. Matthes E. Python crash course, 3rd edition. No Starch Press, Incorporated, 2022.
12. Pilgrim M. Dive into python 3. Berkeley, CA : Apress, 2009. URL: <https://doi.org/10.1007/978-1-4302-2416-7>.
13. Chollet F. Deep learning with python, second edition. Manning Publications Co. LLC, 2021. 400 p.
14. Lane H., Hapke H., Howard C. Natural language processing in action: understanding, analyzing, and generating text with python. Manning Publications, 2019. 544 p.
15. Chen Z., Wu M., Li X. Generalization with deep learning. WORLD SCIENTIFIC, 2021. URL: <https://doi.org/10.1142/11784>.
16. Recurrent neural networks / ed. by L. Medsker, L. C. Jain. CRC Press, 1999. URL: <https://doi.org/10.1201/9781420049176>.
17. Nakamoto P. Neural networks and deep learning: neural networks & deep learning, deep learning, blockchain blueprint. Createspace Independent Publishing Platform, 2018. 152 p.
18. Bengio Y., Courville A., Goodfellow I. Deep learning. MIT Press, 2016. 800 p.
19. Rungta K. TensorFlow in 1 day: make your own neural network. Independently Published, 2018. 364 p.
20. Tingiris S., Kinsella B. Exploring GPT-3: an unofficial first look at the general-purpose language processing API from openai. Packt Publishing, Limited, 2021. 296 p.
21. Zeng Z., Xiong D. Unsupervised and few-shot parsing from pretrained language models. Artificial intelligence. 2022. Vol. 305. P. 103665. URL: <https://doi.org/10.1016/j.artint.2022.103665>.
22. Welch S. Comparative studies on the human glutamate-pyruvate transaminase phenotypes ? GPT 1, GPT 2-1, GPT 2. Humangenetik. 1975. Vol. 30, no. 3. P. 237–249. URL: <https://doi.org/10.1007/bf00279189>.

23. AIEE Transformer Subcommittee. Interim Report on Overloading Current-Limiting Reactors. Transactions of the American Institute of Electrical Engineers. 1943. Vol. 62, no. 10. P. 611–612. URL: <https://doi.org/10.1109/t-aiee.1943.5058611>.

24. GitHub - openai/gpt-3: GPT-3: Language Models are Few-Shot Learners. GitHub. URL: <https://github.com/openai/gpt-3>.

25. Erl T., Mahmood Z., Puttini R. Cloud computing: concepts, technology and architecture. Pearson Education, Limited.

26. Cloud Computing Services | Google Cloud. Google Cloud. URL: <https://cloud.google.com/?hl=uk>.

27. Google Cloud Platform. Google Cloud Platform. URL: <https://console.cloud.google.com/billing/012844-9D3104D9DF94?authuser=3&hl=uk>.

28. Vertex AI Information. Google Cloud. URL: <https://cloud.google.com/vertex-ai?hl=uk>.

29. The national institute of standards and technology (NIST) celebrates its centennial. Chemistry international -- newsmagazine for IUPAC. 2001. Vol. 23, no. 3. URL: <https://doi.org/10.1515/ci.2001.23.3.65>.

30. Gentle J. E. Least absolute values estimation: an introduction. Communications in statistics - simulation and computation. 1977. Vol. 6, no. 4. P. 313–328. URL: <https://doi.org/10.1080/03610917708812047>.

31. Say I. Python GUI programming with tkinter. Independently Published, 2021.

32. Grayson J. E. Python and tkinter programming. Greenwich, Conn : Manning, 1999. 650 p.

33. Hotel management system using python tkinter GUI / S. Kurzadkar et al. International journal of computer science and mobile computing. 2022. Vol. 11, no. 1. P. 204–208. URL: <https://doi.org/10.47760/ijcsmc.2022.v11i01.027>.

34. Jones K. S. Evaluating natural language processing systems: an analysis and review. Berlin : Springer, 1995. 228 p.

35. Chauhan V. K. Stochastic optimization for large-scale machine learning. Taylor & Francis Group, 2021. 184 p.

36. Machine translation / ed. by J. Su, R. Sennrich. Singapore : Springer Singapore, 2021. URL: <https://doi.org/10.1007/978-981-16-7512-6>
37. Manning C. D. Foundations of statistical natural language processing. Cambridge, Mass : MIT Press, 2010. 680 p.
38. MCTNet: multiscale cross-attention based transformer network for semantic segmentation of large-scale point cloud / B. Guo et al. IEEE transactions on geoscience and remote sensing. 2023. P. 1. URL: <https://doi.org/10.1109/tgrs.2023.3322579> (date of access: 18.12.2023).
39. Efficient processing of deep neural networks / V. Sze et al. Morgan & Claypool Publishers, 2020. 341 p.
40. Chauhan V. K. Stochastic optimization for large-scale machine learning. Taylor & Francis Group, 2021. 184 p.
41. Boyd S., Vandenberghe L. Convex optimization. Cambridge University Press, 2016.
42. mr-pickles. Перплексія в мовних моделях. Хабр. URL: <https://habr.com/ru/companies/wunderfund/articles/580230/>.
43. Перплексія у коді, розрахунок для мовних моделей. Хабр. URL: <https://habr.com/ru/companies/wunderfund/articles/580230/comments/>.
44. Contributors to Wikimedia projects. BLEU - wikipedia. Wikipedia, the free encyclopedia. URL: <https://en.wikipedia.org/wiki/BLEU>.
45. agrankova. Еволюція метрик якості машинного перекладу – Часть 1. Хабр. URL: <https://habr.com/ru/articles/745642/>.
46. Visual attention mechanisms / ed. by V. Cantoni, M. Marinaro, A. Petrosino. Boston, MA : Springer US, 2002. URL: <https://doi.org/10.1007/978-1-4615-0111-4>.
47. Asutp\_mpei. Основи навчання та тестування моделей нейромереж. Хабр. URL: <https://habr.com/ru/articles/722628/>.
48. artem. Решітковий та випадковий пошук. Хабр. URL: <https://habr.com/ru/companies/otus/articles/781700/>.
49. artem. Гіперпараметричний пошук та оптимізація моделей. Хабр. URL: <https://habr.com/ru/companies/otus/articles/754402/>.

50. artem. Гіперпараметричний пошук та оптимізація моделей, частина друга. Хабр. URL: <https://habr.com/ru/companies/otus/articles/754403/>.

51. Перенавчання та способи його уникнення – Вікіпедія. Вікіпедія. URL: <https://uk.wikipedia.org/wiki/Перенавчання>.

52. Субтитри – субтитри з найбільшої у світі бази. Subtitles - movie and TV Series subtitles. URL: <https://www.opensubtitles.org/ru>.

53. Project gutenberг. Project Gutenberg. URL: <https://www.gutenberg.org>.

54. The interactive fiction archives. The Interactive Fiction Archive. URL: <https://www.ifarchive.org>.

55. GitHub – seannyD/VideoGameDialogueCorpusPublic. GitHub. URL: <https://github.com/seannyD/VideoGameDialogueCorpusPublic>.