

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
ГЕНЕРАЦІЇ ТЕКСТОВОЇ ІНФОРМАЦІЇ
RESEARCH AND SOFTWARE IMPLEMENTATION OF
TEXT GENERATION METHODS

Виконав: студент 2 курсу, групи КНТ-112м

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ШАПОВАЛ А.С.

(ПРІЗВИЩЕ та ініціали)

Керівник ГОФМАН Є.О.

(ПРІЗВИЩЕ та ініціали)

Рецензент КОЦУР М.І.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ШАПОВАЛА Арсена Сергійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів генерації текстової інформації. Research and Software Implementation of Text Generation Methods

керівник проєкту (роботи) к.т.н., ГОФМАН Євгеній Олександрович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 14 ” листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ГОФМАН Є.О., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання “ 16 ” жовтня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки		
4	Розробка структури програми.	3 тиждень	Розділ 3
5	Розробка програми.	4-7 тижні	Розділи 3,4
6	Тестування та експериментальне дослідження	8 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	9-10 тижні	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	11 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Арсен ШАПОВАЛ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Євгеній ГОФМАН
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
97 с., 28 рис., 5 табл., 3 дод., 19 джерел.

PYTHON, PYCHARM, ГЕНЕРАЦІЯ ТЕКСТІВ, НЕЙРОННІ МЕРЕЖІ,
ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ТЕКСТОВА ІНФОРМАЦІЯ.

Об'єкт дослідження – процес розробки програмного забезпечення для генерування текстів.

Предмет дослідження – методи та програмні засоби для розв'язання завдань створення текстової інформації.

Метою роботи є дослідження та розробка програмного забезпечення для розв'язання завдань створення текстової інформації.

Матеріали, методи та технічні засоби: мова програмування Python, середовище розробки PyCharm.

Результати. Розроблено програмне забезпечення для розв'язання завдань створення текстової інформації за допомогою мови програмування Python та середовища розробки PyCharm.

Практична цінність роботи полягає у розробці програмного забезпечення, що реалізує методи генерації тексту та дозволяє розв'язувати створювати тексти у вигляді поетичних форм.

Висновки. Виконано проєктування програмного забезпечення для розв'язання завдань створення текстової інформації. Розроблено програмне забезпечення для розв'язання завдань створення текстової інформації. Здійснено тестування розробленого програмного забезпечення для розв'язання завдань створення текстової інформації.

Галузь використання – програмні засоби для генерації текстів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 97 pages, 28 figures, 5 tables, 3 appendixes, 19 sources.

PYTHON, PYCHARM, PYCHARM, TEXT GENERATION, NEURAL NETWORKS, SOFTWARE, TEXT INFORMATION.

The object of research is the process of developing software for text generation.

The subject of the research is software for solving the tasks of creating textual information.

The purpose of this work is to develop the software for solving the tasks of creating textual information.

Materials, methods and technical tools: Python programming language, PyCharm development environment.

Results. The software tools for solving problems of creating textual information using Python programming language and PyCharm development environment have been developed.

The practical value of the work lies in the development of software that implements text generation methods and allows you to create texts in the form of poetic forms.

Conclusions. The software for solving problems of creating text information has been designed. The software for solving problems of creating text information has been developed. The developed software for solving problems of creating text information has been tested.

Scope of use – software tools for text generation.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	8
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Поняття про програмне оброблення мови та генерацію тексту	11
1.2 Аналіз програмного забезпечення для генерації текстової інформації ..	25
1.3 Висновки за розділом 1	34
2 Матеріали і методи	36
2.1 Вибір мови програмування	36
2.2 Вибір середовища розробки для генерації текстової інформації	38
2.3 Методи та моделі машинного навчання для генерації текстової інформації	41
2.4 Розробка моделі взаємодії користувача з програмним забезпеченням генерації текстової інформації	47
2.5 Висновки за розділом 2	49
3 Опис програми	50
3.1 Структура програмного забезпечення для генерації текстової інформації	50
3.2 Функціонування програмного забезпечення для генерації текстової інформації	51
3.3 Основні рішення щодо розробки програмного забезпечення для генерації текстової інформації	54
3.4 Проєктування інтерфейсу програмного забезпечення для генерації текстової інформації	60
3.5 Висновки за розділом 3	62
4 Експлуатація, тестування та експериментальне дослідження програми	63
4.1 Призначення й умови застосування програми	63
4.2 Характеристики програми для розв’язання задачі генерації текстової інформації	64

4.3 Інструкція по експлуатації програми.....	64
4.3.1 Звернення до програми.....	64
4.3.2 Вхідні й вихідні дані	64
4.3.3 Повідомлення.....	65
4.4 Виконання програмного забезпечення для розв’язання задачі генерації текстової інформації.....	65
4.5 Тестування програмного забезпечення для генерації текстової інформації.....	67
4.6 Висновки за розділом 4	67
Висновки	68
Перелік джерел посилання	71
Додаток А Технічне завдання	73
Додаток Б Фрагмент тексту програми	77
Додаток В Слайди презентації	90

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface; LMV;

GAN – Generative Adversarial Network;

GPT – Generative Pre-trained Transformer;

LSTM – Long Short Term Memory;

МОВ – машини опорних векторів;

НМ – нейронні мережі;

ПЗ – програмне забезпечення;

ШІ – штучний інтелект.

ВСТУП

У наш час широко використовуються програмні засоби для генерації текстової інформації, що допомагають створювати різні тексти, взаємодіяти з користувачами, відповідати на різні запитання та допомагати у вирішенні різних проблем [1]-[6].

Програмне забезпечення для генерації текстової інформації застосовуються в багатьох сферах, від створення чат-ботів та віртуальних асистентів до машинного перекладу, і має велике значення для вирішення проблем різного характеру [1]-[6]. Зокрема, використання програмного забезпечення для генерації текстової інформації дозволяє забезпечити взаємодію з користувачами, відповідати на їх запитання та надавати необхідну інформацію. Генерація текстового контенту за допомогою відповідних алгоритмів та програмних засобів дозволяє створювати описи новин, оглядів і інші види текстів, що може значно полегшити роботу журналістів та працівників інших сфер [1]-[6].

Отже, у наш час програмні засоби для генерації тексту є важливим інструментом для автоматизації та поліпшення багатьох аспектів роботи та взаємодії користувачів з інформацією. Програмні засоби для генерації тексту допомагають в створенні автоматизованих систем відповіді на користувацькі запитання, розробці контенту для вебсайтів та соціальних мереж, а також в інших областях, де генерація тексту може забезпечити ефективну та швидку обробку інформації. Застосування цих програмних засобів включає створення текстових блоків для маркетингових матеріалів, написання оглядів, переклад тексту між мовами, розробку віртуальних асистентів та чат-ботів, а також удосконалення інтерфейсів для зручної взаємодії з користувачами [1]-[6]. Однак відомі програми для генерації тексту можуть створювати стереотипні або загальні висловлювання, що робить їх менш корисними для унікальних або специфічних завдань, деякі програми можуть іноді генерувати неправильну або недостовірну інформацію, що може

призвести до поширення некоректної інформації. Інші подібні програми для досягнення оптимальних результатів та уникнення неправильних висновків програми часто потребують постійного нагляду та втручання від фахівців для покращення генеративної моделі [1]-[6].

Тому актуальною є розробка програмного забезпечення для генерації текстової інформації.

У дипломній кваліфікаційній роботі магістра розв'язується актуальне завдання розробки програмного забезпечення для генерації текстової інформації.

Для досягнення поставленої мети у кваліфікаційній роботі магістра необхідно розв'язати такі задачі:

- виконати аналіз предметної області та програмних засобів для генерації текстової інформації;
- розробити модель взаємодії користувача з програмним забезпеченням для генерації текстової інформації;
- здійснити проектування програмного забезпечення для генерації текстової інформації;
- створити програмне забезпечення для генерації текстової інформації;
- виконати тестування розробленого програмного забезпечення для генерації текстової інформації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття про програмне оброблення мови та генерацію тексту

Генерація тексту – це метод, яким штучний інтелект створює текст, що наслідує природне спілкування людини. Генерація тексту – це процес, під час якого системи штучного інтелекту відтворюють шаблони та стилі людської мови для створення тексту. Цей процес породжує зв'язні, значущі тексти, що схожі на природне людське спілкування. Значення генерації тексту зростає в різних областях, таких як обробка природної мови, створення контенту, обслуговування клієнтів і розробка програмного забезпечення [1]-[3].

Процес генерації тексту використовує алгоритми та мовні моделі для обробки вхідних даних і створення тексту на виході. Це включає в себе навчання моделей штучного інтелекту розпізнавати шаблони, граматику і контекстуальну інформацію, використовуючи великі набори текстових даних. Після цього ці моделі використовують набуті знання для створення нових текстів на основі наданих вказівок чи умов [1]-[3].

В базисі створення тексту лежать мовні моделі, такі як GPT (Generative Pre-trained Transformer) і Google PaLM, які пройшли попередню підготовку і навчені аналізувати об'ємні тексти з Інтернету. Ці моделі використовують методи глибокого навчання, зокрема нейронні мережі (НМ), для розуміння структури речень та створення зв'язного контекстуального тексту [1]-[3].

Під час генерації тексту модель штучного інтелекту отримує вхідні дані, такі як фраза чи ключове слово, і використовує цю інформацію для прогнозування найбільш ймовірного наступного слова чи фрази. Процес продовжується, і шаблон генерує контекст та послідовний текст, досягаючи бажаної довжини або зупинки [1]-[3].

Генерація тексту знаходить практичне використання в численних застосуваннях у реальному житті, включаючи [1]-[3]:

- створення вмісту: програмі системи на основі штучного інтелекту можуть автоматично генерувати статті, блог-пости та описи продуктів. Ці

програмі системи навчені обробляти об'ємні дані і створювати релевантний контент значно швидше, ніж людина [1]-[6];

– чат-боти та віртуальні помічники: використання програмних систем генерації тексту в чат-ботах та віртуальних помічниках дозволяє взаємодіяти з користувачами у формі розмови. Вони можуть розуміти запити користувачів, надавати відповіді, а також надавати персоналізовану підтримку та інформацію [1]-[6];

– мовний переклад: моделі генерації тексту можна використовувати для поліпшення послуг мовного перекладу. Аналізуючи великі обсяги перекладеного тексту, такі моделі та відповідне програмне забезпечення можуть надавати точні переклади в режимі реального часу, поліпшуючи багатомовне спілкування [1]-[6];

– реконструкція: текстові підсумки надають стислу версію інформації, виділяючи ключові моменти, які з використанням спеціалізованого програмного забезпечення можна використовувати для створення наукових статей, блог-постів, статей, розділів та резюме книг [1]-[6].

Проаналізуємо переваги програмного забезпечення створення текстів. Перевагами такого програмного забезпечення є такі [1]-[6]:

– підвищена продуктивність: використання штучного інтелекту для генерації тексту у програмному забезпеченні дозволяє автоматизувати процес створення вмісту, зменшуючи час і зусилля, які зазвичай потрібні для ручного написання. Це може суттєво підвищити продуктивність і дозволити користувачам генерувати великі обсяги вмісту [1]-[6];

– індивідуальна настройка: механізми генерації тексту у програмному забезпеченні розроблені для створення персоналізованого вмісту, враховуючи вподобання користувача та його історію. Це дозволяє відповідному програмному забезпеченню надавати індивідуальні рекомендації, персоналізовані маркетингові повідомлення та адаптовані відповіді під час взаємодії з користувачами [1]-[6];

– використання мови: генерація тексту розширює можливості у програмного забезпечення автоматичного перекладу та синтезу мовлення, надаючи доступ до інформації тим, хто може зазнавати труднощів з читанням або розумінням тексту. Таке програмне забезпечення сприяє новим формам спілкування та покращує доступність інформації [1]-[6].

Проаналізуємо обмеження програмного забезпечення створення текстів. Генерація тексту супроводжується кількома обмеженнями [1]-[6]:

– відсутність розуміння контексту: програмне забезпечення та відповідні моделі генерації тексту часто мають складнощі з осмисленням ширшого мовного контексту і нюансів. Вони формують текст, користуючись шаблонами з навчальних даних, не завжди розуміючи справжнє значення та призначення слів. Це може вести до неточностей, відхилень або непрозорих результатів [1]-[6];

– сильна залежність від навчальних даних: програмне забезпечення та відповідні моделі генерації тексту значно покладаються на якість та характер отриманих навчальних даних. Якщо навчальні дані обмежені, або вони мають упередження або не відображають всіх мовних варіацій, згенерований за допомогою відповідного програмного забезпечення текст може бути упередженим, недостатньо різноманітним або містити інші недоліки [1]-[6];

– проблеми, що виникають у процесі обробки незвичайних або непередбачуваних сценаріїв для моделей генерації тексту. Моделям генерації тексту може бути складно впоратися з ситуаціями, які не були враховані в навчальних даних, і це може призвести до неправильних або нелогічних відповідей, особливо у випадках, коли сценарії є непередбачуваними або винятковими [1]-[6];

– етичні аспекти в процесі створення тексту. Ця проблема включає питання стосовно можливості виникнення дезінформації, пропаганди чи шкідливого вмісту. Без належного контролю та відстеження шаблони генерації тексту та відповідне програмне забезпечення можуть

використовуватися для розповсюдження дезінформації, посилення упередженості або завдання шкоди окремим особам чи подіям [1]-[6].

Проаналізуємо процес розробки програмного забезпечення для створення текстів [1]-[6]:

На першому етапі процесу розробки програмного забезпечення для створення текстів виконується збір та попередня обробка даних. Тут текстові дані збираються, проходять через процес очищення та розбиття на менші одиниці для створення вхідних даних для моделі. Цей етап становить фундамент для подальших кроків у створенні текстів за допомогою штучного інтелекту [1]-[6]. Під час збору даних у процесі розробки програмного забезпечення для створення текстів, текстова інформація збирається з різних джерел. Це може бути інтернет-контент, літературні твори, наукові публікації та інше. Зібрані за допомогою спеціального програмного забезпечення дані потім піддаються ретельній попередній обробці для виключення непотрібної інформації, а також для підготовки текстів до подальшого використання у моделі [1]-[6].

Процес попередньої обробки текстових даних при розробці програмного забезпечення для створення текстів включає в себе не лише видалення надлишкових символів та форматування тексту, але й детальний аналіз структури речень та слів. Такий аналіз може включати в себе визначення частоти використання слів, виявлення синонімів та інших важливих рис тексту [1]-[6]. Після виконання збору та попередньої обробки текстової інформації, синтезується велика маса даних, готових для використання у глибокому навчанні для розробки моделей та програмного забезпечення для створення текстів [1]-[6]. Ці дані стають основою для наступних етапів розробки програмного забезпечення для створення текстів, таких як навчання моделі та генерація текстів, дозволяючи програмній системі штучного інтелекту зрозуміти імовірність та логіку використання слів у різних контекстах [1]-[6].

На наступному етапі розробки програмного забезпечення для створення текстів виконується навчання моделі. Обрана модель для генерації текстів, наприклад, GPT (Generative Pre-trained Transformer), навчається на послідовностях перетворених текстових даних, підганяючи свої параметри для точного прогнозування наступного слова в послідовності на основі попередніх [1]-[6]. Під час навчання модель для генерації текстів адаптує свої параметри на основі великої кількості текстових послідовностей. Процес навчання моделі для генерації текстів спрямований на те, щоб модель розпізнавала закономірності в мові, вивчала граматичні конструкції, синтаксичні особливості і здатність розуміти контекст в тексті. Параметри моделі для генерації текстів постійно коригуються для того, щоб максимізувати точність її прогнозів [1]-[6].

По завершенні етапу навчання модель для генерації текстів стає здатною генерувати текст з високим рівнем природності, враховуючи контекст і попередньо засвоєні мовні структури. Важливою характеристикою моделі для генерації текстів у відповідному програмному забезпеченні є те, що модель може розуміти зв'язки між словами та їх семантику, що поліпшує рівень якості генерованого за допомогою моделі тексту [1]-[6]. Такий підхід до навчання моделі для генерації текстів у відповідному програмному забезпеченні дозволяє створювати текст не лише на основі конкретного шаблону, але і враховуючи логіку та структуру мовлення, забезпечуючи різноманітність та природність виходу [1]-[6].

Після успішного завершення етапу навчання синтезована модель набуває здатності до генерації текстів. Цей процес у програмному забезпеченні розпочинається з використання заданої початкової послідовності, яка слугує початковим стимулом для генерації. Генеративна модель та програмне забезпечення використовує отримані знання про мовні структури та залежності для прогнозування наступного слова в послідовності [1]-[6].

Процес генерації текстів виконується ітеративно: слово за словом. Починаючи з першого слова в послідовності, генеративна модель враховує

контекст та ймовірності для визначення, яке слово буде найбільш відповідати логіці тексту [1]-[6]. Коли воно вибране, модель та програмне забезпечення додає відповідне слово до початкової послідовності, і процес повторюється для генерації наступного слова [1]-[6].

Цей ітеративний підхід дозволяє генеративній моделі у програмному забезпеченні створювати тексти, які взаємодіють з контекстом та зберігають природність мовлення. Генеративна модель може автоматично регулювати свої прогнози, враховуючи змінність контексту та введення. Такий підхід забезпечує створення текстів, які мають логічну структуру та стилістичну відповідність [1]-[6].

Наступним етапом у роботі програмного забезпечення генерації текстів є використання стратегій декодування, які в процесі генерації текстів відіграють важливу роль у визначенні наступного слова в послідовності. Різноманітні підходи дозволяють досягти різної якості та креативності в контенті, що генерується [1]-[6]. Виконаємо дослідження деяких з основних стратегій декодування. Жадібне декодування тексту є простим, але ефективним методом, при якому модель вибирає наступне слово з найвищою ймовірністю. Цей підхід часто використовується для забезпечення легкості розуміння та лінгвістичної точності. Пошук з використанням променя розширює можливості вибору, дозволяючи генеративній моделі розглядати кілька альтернативних слів одночасно та обирати слово з променя або певної множини ймовірностей, що дозволяє більш гнучко адаптуватися до різноманітних лінгвістичних контекстів. Вибірка зі списку дозволяє враховувати різні рівні ймовірності для кількох можливих слів у процесі генерації контенту, зокрема обирати слова з верхньої групи за ймовірністю, або обмежувати вибір слів до тих, які складають вказаний кумулятивний ймовірнісний поріг [1]-[6]. Ці стратегії декодування дозволяють моделям генерації тексту та відповідному програмному забезпеченню бути більш гнучкими та адаптованими до різних вимог і контекстів текстової інформації,

що призводить до створення більш якісного та різноманітного контенту за допомогою інтелектуальної системи [1]-[6].

Тонке налаштування синтезованої генеративної моделі є важливим кроком у процесі розвитку програмного забезпечення генерації тексту за допомогою штучного інтелекту. Попередньо підготовлені генеративні моделі, які пройшли етапи збору даних, попередньої обробки, навчання та генерації, часто потребують додаткового налаштування параметрів для досягнення оптимальної продуктивності моделі та адаптації її до конкретних завдань чи доменів генерації тексту [1]-[6].

Під час тонкого налаштування параметрів генеративної моделі може вноситися ряд модифікацій, які дозволяють досягти кращої здатності до вирішення конкретних завдань. Тонке налаштування параметрів генеративної моделі включає в себе оптимізацію ваг та параметрів синтезованих нейронних мереж для генерації текстів, а також врахування особливостей конкретних вхідних даних завдання створення текстової інформації [1]-[6].

Тонке налаштування параметрів генеративної моделі є важливим етапом при розробці програмного забезпечення створення текстової інформації, оскільки воно дозволяє забезпечити, що генеративна модель належним чином вирішує конкретні завдання створення текстової інформації та забезпечує високу якість результатів генерованих текстів [1]-[6].

Для створення моделей генерації текстової інформації використовуються моделі генеративного штучного інтелекту [1]-[6]. Генеративний інтелект (ГІ) представляє собою важливу галузь ІІ та машинного навчання, яка використовується для створення програмного забезпечення, здатного генерувати штучний контент, наприклад, текстові матеріали, зображення, аудіо та відео. Його основною метою є розвиток інтелектуальних програмних систем, які здатні аналізувати та відтворювати властивості, характерні для людського творчого вираження [1]-[6].

У процесі використання програмного забезпечення на основі генеративного інтелекту на першому етапі важливою частиною є накопичення

великих обсягів навчальних даних, які будуть використані для навчання генеративних моделей, що будуть вбудовані у відповідному ПЗ. Ці дані можуть охоплювати різні галузі, включаючи тексти, зображення та інші форми медіа. Після збору даних для створення генеративних моделей відбувається їхнє попереднє оброблення [1]-[6].

Далі виконується навчання генеративних моделей. Обрані алгоритми машинного навчання та відповідні моделі, такі як глибокі моделі нейронних мереж, піддаються тренуванню (виконується їх синтез, тобто підбір певних параметрів) на навчальних даних з метою розпізнавання закономірностей та структур в цих даних, зокрема, текстових даних. Це дозволяє майбутнім генеративним моделям набувати здатність генерувати новий контент, який відповідає зразкам і структурам, виявленим у навчальних даних [1]-[6].

Генеративний інтелект використовує ШІ і алгоритми машинного навчання, щоб дозволити програмним засобам використовувати їхні навчальні дані для створення штучного вмісту, такого як текст, зображення, аудіо та відео. Велика кількість технологічних компаній або створюють власні генеративні рішення ШІ, або інвестують у компанії, які розробляють великі мовні моделі для генерації текстової інформації та оброблення природної мови [1]-[6].

Генеративний ШІ стикається з різними викликами, пов'язаними із конфіденційністю, шахрайством, злочинним використанням та обфускацією. Однією з ключових проблем використання генеративних моделей та програмного забезпечення є питання конфіденційності. Генеративні моделі та відповідне програмне забезпечення можуть мати доступ до великих обсягів конфіденційної текстової та іншої інформації, що може створити загрозу несанкціонованого використання відповідних даних. За недбалістю в обробці цих даних за допомогою генеративного програмного забезпечення може виникнути ризик їхнього витоку або некоректного використання [1]-[6].

Іншою проблемою використання генеративних моделей та програмного забезпечення є можливість використання генеративних моделей для

цифрового шахрайства. Шахраї можуть використовувати технології генеративного інтелекту для автоматичного створення обманливого контенту, що може впливати на довіру споживачів через використання спеціалізованого програмного забезпечення або соціальні мережі [1]-[6].

Злочинне використання генеративних моделей та програмного забезпечення становить суттєвий серйозний ризик, оскільки такі технології та програмні засоби можуть використовуватися для створення фальшивих документів, образів або іншого обманливого контенту, що може бути використано у злочинних діях[1]-[6].

Обфускація або утруднення розуміння та виявлення того, що створено за допомогою використання генеративних моделей та програмного забезпечення також є важливою проблемою [1]-[6].

Генеративний ШІ є технікою, що використовує наявний текст, аудіо файли або зображення для створення нового контенту. Шляхом використання генеративного штучного інтелекту комп'ютери та відповідне програмне забезпечення аналізують фонові шаблони, пов'язані з початковим введенням, і створюють аналогічний контент. Генеративний ШІ відрізняється від більшості інших методів штучного інтелекту, де моделі штучного інтелекту спрямовані на вирішення завдань, що мають конкретні відповіді (наприклад, задачі класифікації або прогнозування) [1]-[6].

Проаналізуємо техніки генеративного штучного інтелекту для синтезу генеративних мовних моделей. Моделі-трансформери, зокрема, GPT-3, LaMDA, Wu-Dao і ChatGPT, використовують технологію трансформації для моделювання когнітивної уваги та окремо оцінюють важливість вхідних компонентів. Ці моделі навчаються розуміти мову або зображення, вивчають класифікаційні завдання та генерують текст або зображення на основі великих наборів матеріалів [1]-[6].

Однією з ключових характеристик моделей-трансформерів є їх здатність окремо оцінювати важливість різних елементів вхідної текстової інформації. Це дозволяє таким моделям фокусуватися на суттєвих аспектах та

взаємодіяти з ними в більш контекстуальний спосіб, що покращує якість та різноманітність генерованого вмісту текстових даних [1]-[6].

Моделі-трансформери, як правило, навчаються на великих наборах текстових даних, що включають в себе різноманітність текстових та візуальних матеріалів. Це сприяє їхній здатності розуміти різні стилі мови, концепції та візуальні зразки, роблячи їх універсальними інструментами для створення текстуального та візуального контенту в різних галузях науки, виробництва та технологій [1]-[6].

Мережа генеративно-змагальної генерації (Generative Adversarial Network, GAN) складається з двох нейронних мереж – генератора та дискримінатора, які вступають в конкурентні відносини для досягнення балансу між ними [1]-[6]. Генераторна мережа відповідає за створення нової текстової інформації або іншого контенту, такого як вихідні дані [1]-[6]. Мережа дискримінатора розпізнає відмінності між вихідними та створеними даними та намагається визначити, які з них є найбільш схожими на вихідні дані [1]-[6].

Генеративно-змагальна мережа використовує концепцію двох нейронних мереж для досягнення результатів у генерації нового текстового контенту [1]-[6]. Генераторна мережа відповідає за створення нової інформації або вмісту. Вона працює на основі вхідних даних, заздалегідь навчених на широкому масиві текстової інформації, і генерує вихідні дані, які, як правило, мають виглядати так само, як і дані, що використовувались під час навчання генеративної моделі [1]-[6].

Мережа дискримінатора виконує завдання розпізнавання та розмежовування між вихідними (реальними) даними і тими, які створені нейронною мережею-генератором. Мережа дискримінатора тренується (синтезується) на великому наборі даних і вивчає різницю між оригінальними та створеними даними [1]-[6].

Процес синтезу генеративно-змагальної мережі для синтезу генеративної моделі текстової інформації ґрунтується на взаємодії зазначених

вище двох мереж. Нейронна мережа, що виконує функції генератору, намагається піднімати якість своїх створених текстових даних, тоді як нейронна мережа-дискримінатор удосконалює свою здатність розрізняти відмінності між реальними та синтезованими текстовими даними. Цей процес триває до того моменту, коли мережа генератор не досягне такого рівня, коли вихід цієї нейронної мережі важко розрізнити від реальних текстових даних [1]-[6].

Генеративно-змагальна мережа набула широкого спектру застосувань, зокрема, оброблення текстової інформації, генерація зображень, вирішення завдань обробки зображень та створення текстового контенту на основі текстових даних. Генеративно-змагальна мережа виявляється особливо ефективним у випадках, коли необхідно створювати новий контент, який має виглядати природним та схожим на той, що генерується звичайною людиною [1]-[6]. Налаштування генеративно-змагальної мережі включає в себе процес, під час якого модель перетворює вхідні дані в компактний код, а декодер відтворює вихідну інформацію з цього коду. При правильному виборі та навчанні це стисле представлення фіксує розподіл вхідних даних як представлення низької розмірності [1]-[6].

Проаналізуємо сфери застосування генеративного штучного інтелекту [1]-[6].

Основною сферою застосування генеративного штучного інтелекту є генерація тексту. Програми застосунки генеративного штучного інтелекту можуть формувати текст та взаємодіяти з користувачами [1]-[6]. Прикладом такого інструменту є ChatGPT, який генерує розмови. Будь ласка, перегляньте нашу докладну статтю про генерацію тексту за допомогою штучного інтелекту [1]-[6].

Створення зображень облич, предметів та сцен зображень рослин: генеративний ШІ може створювати графічні зображення, що імітують реальні сцени та об'єкти [1]-[6].

Перетворення тексту в зображення: генеративний ШІ дозволяє створювати реалістичні візуальні представлення на основі текстових описів простих об'єктів, таких як птахи і квіти [1]-[6].

Моделі генеративного штучного інтелекту активно використовуються для додавання вмісту, що розширює сферу його можливостей. Із застосуванням 3D-друку та інших технологій, генеративний ШІ може виготовляти складові елементи з самого початку [1]-[6].

Генеративний ШІ має застосування у написанні програмного коду, використовуючи складні алгоритми, які дозволяють створювати код. Ці алгоритми можуть аналізувати вхідні дані, навчатися на їх основі та генерувати нові частини програмного забезпечення. Це допомагає автоматизувати процес розробки програм та покращує їхню ефективність і точність, що дозволяє швидше створювати та вдосконалювати програми [1]-[6]. Технологія генеративного штучного інтелекту може виявитися корисною не лише для генерації конкретного програмного коду, але й для покращення роботи веброзробників та програмістів. Технологія генеративного штучного інтелекту може допомогти у вирішенні певних завдань, створенні складних алгоритмів та навіть вплинути на оптимізацію процесів розробки програмних продуктів у багатьох галузях [1]-[6].

Генеративний штучний інтелект, застосовуючи мовні генеративні моделі та навчені на великій кількості текстів, має потенціал для створення поетичних творів. Технологія генеративного штучного інтелекту може враховувати не лише граматику та структуру речення, а й емоційний відтінок, виразність мови та стиль письма. Технологія генеративного штучного інтелекту використовується для втілення естетичних та художніх аспектів у формі літературних творів, що мають виразити відчуття, почуття, філософію чи сприйняття світу через слово [1]-[6].

Технологія генеративного штучного інтелекту може бути важливим інструментом для письменників, поетів та літературних творців, сприяючи їм у створенні нових творчих ідей, відкриваючи можливості для експериментів з

мовою та формами вираження. Технологія генеративного штучного інтелекту дозволяє генерувати тексти, які мають естетичну цінність та можуть вразити читачів своєю глибиною та красою [1]-[6].

Генеративний штучний інтелект знаходить широке застосування в обробці відео та реставрації візуального контенту, зокрема фільмів, відео та фотографій. Використовуючи техніки глибокого навчання, він здатний масштабувати та покращувати якість відео, перетворюючи його у вищу роздільну здатність, таку як 4K чи навіть вище [1]-[6].

Технологія генеративного штучного інтелекту може відновлювати старі відеозаписи, які мають низьку роздільну здатність чи пошкодження, перетворюючи їх у якісне, високо роздільне відео. Технологія генеративного штучного інтелекту може підвищувати чіткість, деталізацію, відновлювати кольори та зменшувати шуми на зображенні, надаючи нове життя старим матеріалам [1]-[6].

Технологія генеративного штучного інтелекту також застосовується для покращення швидкості кадрів у відео, що може бути корисним у реставрації старих фільмів, де швидкість кадрів може бути низькою. Такий процес дозволяє відтворити відео з більш плавною і природною рухомою картинкою [1]-[6].

Генеративний штучний інтелект відіграє важливу роль у локалізації контенту, особливо в медіа та розважальній індустрії. Технології синтезу тексту та клонування голосу дозволяють адаптувати контент до різних мовних аудиторій та культур, забезпечуючи оригінальний контент у мові та формі, яка буде більш зрозумілою та прийнятною для різних глядачів [1]-[6].

Генеративні моделі в медицині відіграють важливу роль у виявленні передумов для розвитку захворювань та раннього діагностування хвороб. Вони використовуються для аналізу медичних даних, щоб виявляти відмінності та шаблони, які можуть вказувати на ризики конкретних захворювань [1]-[6].

Клонування голосу за допомогою генеративного штучного інтелекту також використовується для локалізації аудіо, де оригінальний голос артиста чи спікера переноситься на інші мови. Ця технологія дозволяє зберігати стиль та інтонацію голосу, що є важливим для збереження оригінальності та звукової цілісності контенту в новому мовному середовищі [1]-[6].

Генеративне моделювання відіграє значну роль у галузі машинного навчання, оскільки дозволяє створювати менш упереджені моделі, зокрема, для генерації текстової інформації, що виявляють вищу гнучкість в розумінні та взаємодії з абстрактними поняттями в різних сценаріях та різних галузях знань [1]-[6]. Генеративні моделі також застосовуються у дослідженнях для створення більш точних прогнозів та вивчення взаємозв'язків між різними параметрами. Вони сприяють розвитку нових методів аналізу даних, які дають глибше розуміння складних процесів у природі та суспільстві [1]-[6].

Визначено, що у наш час широко використовуються програмні засоби для генерації текстової інформації, що допомагають створювати різні тексти, взаємодіяти з користувачами, відповідати на різні запитання та допомагати у вирішенні різних проблем. Показано, що генерація тексту – це метод, яким штучний інтелект створює текст, що наслідує природне спілкування людини. Відзначено, що значення генерації тексту зростає в різних областях, таких як обробка природної мови, створення контенту, обслуговування клієнтів і розробка програмного забезпечення. Показано, що в базисі створення тексту лежать мовні моделі. Проаналізовано практичні застосування генерації тексту за допомогою засобів генеративного штучного інтелекту. Визначено переваги та обмеження програмного забезпечення створення текстів. Проаналізовано процес розробки програмного забезпечення для створення текстів на основі генеративних моделей. Відзначено, що генеративний інтелект штучний інтелект представляє собою важливу галузь штучного інтелекту та машинного навчання, яка використовується для створення програмного забезпечення, здатного генерувати штучний контент, наприклад, текстові матеріали,

зображення, аудіо та відео. Проаналізовано техніки генеративного штучного інтелекту для синтезу генеративних мовних моделей.

1.2 Аналіз програмного забезпечення для генерації текстової інформації

Проаналізуємо програмне забезпечення підтримки для генерації текстової інформації [7]-[13].

Програмне забезпечення Jasper (рис. 1.1) є текстовим генератором на основі генеративних моделей штучного інтелекту. Програмне забезпечення Jasper використовує мовну модель OpenAI GPT-3 для створення унікального та цікавого тексту [9], [11]. З програмним забезпеченням Jasper користувач може використовувати генеративний штучний інтелект для швидкого та легкого створення контенту для блогів, статей, текстів у соціальних мережах, рекламних текстів, описів продуктів, цільових сторінок [9], [11].

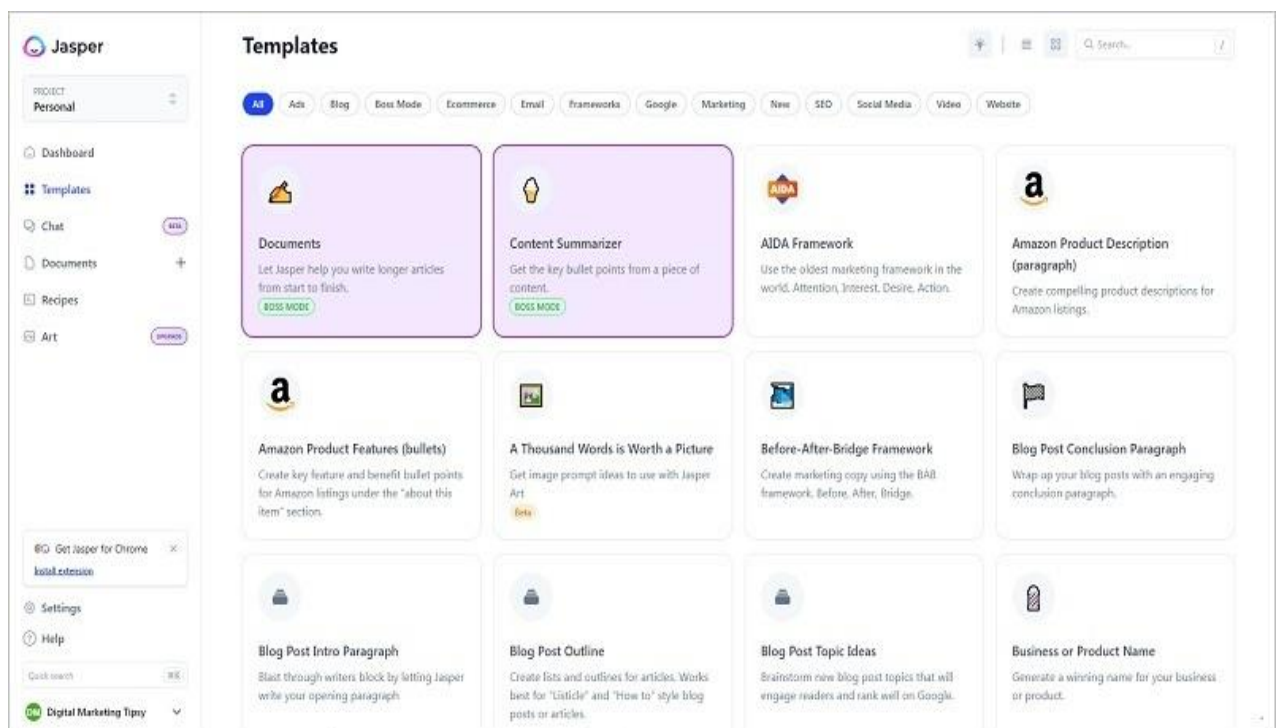


Рисунок 1.1 – Програмне забезпечення Jasper [9], [11]

Головна функція програмного забезпечення Jasper дозволяє створювати довгі тексти за допомогою наявних функцій. Окрема функція програмного забезпечення Jasper Chat використовує ChatGPT для природної інтеграції програми з генеративним штучним інтелектом для створення різноманітного текстового вмісту [9], [11].

Програмне забезпечення Jasper є дуже простим у використанні, тому навіть для користувачів без досвіду роботи зі штучним інтелектом використання цієї програми є зрозумілим та зручним. Програмне забезпечення Jasper має детальну документацію про те, як працює це програмне забезпечення [9], [11]. Програмне забезпечення Jasper може бути корисним для блогерів, маркетологів чи інших професій, де виникає необхідність створення текстового контенту [9], [11].

Серед особливостей програмного забезпечення Jasper для створення текстового контенту можна виділити такі [9], [11]:

- програмне забезпечення Jasper використовує генеративний штучний інтелект GPT-3 для створення текстового вмісту [9], [11];
- наявність великої кількості програмних інструментів штучного інтелекту та сценаріїв використання [9], [11];
- функція чату, яка дозволяє користувачам створювати контент, шляхом взаємодії зі штучним інтелектом [9], [11];
- підтримка великої кількості мов світу [9], [11];
- засоби інтеграції [9], [11];
- можливість залучати членів команди та керувати кількома проектами одночасно [9], [11];
- модулі генератору тексту є досить зручними та подані у форматі сітки, що полегшує вибір відповідних фільтрів [9], [11];
- інтеграція з граматичними засобами, що полегшує роботи над граматичними помилками (якщо такі є) та перевірки створеного тексту на предмет плагіату [9], [11];

- можливість отримання доступу до файлів, створених із тексту та історії спілкування з генеративною моделлю [9], [11];

- безкоштовна пробна версія, яка за декілька днів дозволяє безкоштовно створити тисячі слів контенту. Після закінчення безкоштовного пробного періоду необхідно підписатися на платний план, вартість якого стартує від 29 доларів США на місяць [9], [11];

- можливість створювати великі тексти, які також може бути подані у розмовному жанрі [9], [11];

- можливість працювати як з засобом розширення Chrome [9], [11].

Серед недоліків програмного забезпечення Jasper можна відзначити те, що ця програма для можливості роботи з усіма функціями вимагає оплати, що ускладнює її використання для організацій з невеликим бюджетом. Деякі типи користувачів з невеликим досвідом роботи можуть мати певні проблеми при генерації тексту за допомогою цього програмного забезпечення [9], [11].

Програмне забезпечення Writesonic [9], [12] є корисним інструментом для швидкого та легкого створення різноманітного контенту. Як контент, що створюється за допомогою цієї програми, можуть бути публікації в блогах, статті, заголовки для соціальних мереж, електронні листи або вебсайти, програмне забезпечення Writesonic здатне генерувати ці тексти за досить швидкий час [9], [12].

Програмне забезпечення Writesonic розроблене з метою максимально простого використання, дозволяючи різним користувачам створювати текстовий контент з будь-якої теми. Використовуючи технологію штучного інтелекту генеративну модель OpenAI GPT-3, Writesonic створює точний, унікальний текстовий контент [9], [12].

Програмне забезпечення Writesonic може бути ефективно використано для створення текстового вмісту на різних мовах, що робить цю програму ефективним інструментом для багатомовних проектів у сфері бізнесу та маркетингу та розробки програмного забезпечення. Гнучкість та можливості програмного забезпечення Writesonic (рис. 1.2) дозволяють користувачам

створювати якісний контент для широкого кола завдань і цілей у різних середовищах [9], [12].

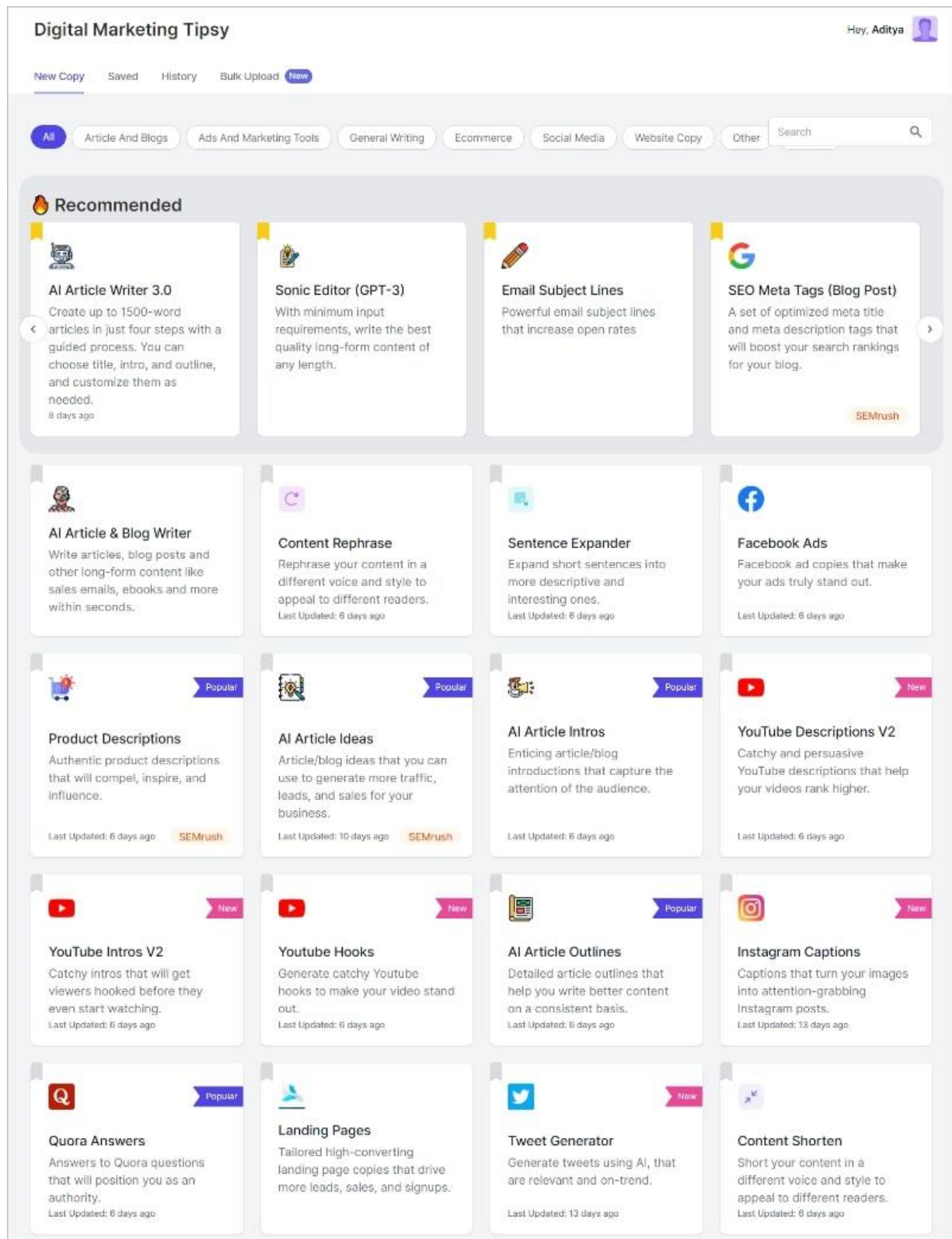


Рисунок 1.2 – Програмне забезпечення Writesonic [9], [12]

Крім того, програмне забезпечення Writesonic можна налаштувати відповідно до тону, стилю та голосу вашого бренду. Отже, якщо програмне забезпечення Writesonic є генератором контенту на основі штучного інтелекту, який дозволяє досить швидко розв'язувати відповідні завдання [9], [12].

Серед особливостей програмного забезпечення Writesonic для створення текстового контенту можна виділити такі [9], [12]:

- програмне забезпечення використовує технологію штучного інтелекту на основі генеративної моделі GPT-3 [9], [12];
- Writesonic пропонує велику кількість інструментів та шаблонів для різних потреб у генерацію текстового вмісту [9], [12];
- розмовний чат-бот, який підтримує спілкування з користувачами у зручному для них форматі [9], [12];
- можливість створення різних текстів з текстовими та голосовими повідомленнями [9], [12];
- можливість інтеграції з різним програмним забезпеченням, зокрема, з SlvPress, Semrush, SurferSEO та Zapier [9], [12];
- засоби роботи з різними членами команди для масштабування процесу створення тексту [9], [12];
- можливість створення текстового контенту на великій кількості глобальних мов [9], [12];
- можливість створювати контент швидко, тобто за лічені хвилини користувач може отримати статті, публікації в блозі, заголовки для соціальних мереж та багато іншого [9], [12];
- різноманітність контенту: інструмент може генерувати різні типи контенту, такі як електронні листи, статті для сайту, заголовки, що зацікавляють у соціальних мережах, та інше [9], [12];
- простота використання: Writesonic розроблений так, щоб бути простим у використанні. Користувач може вести текст на будь-яку тему та швидко отримати готовий контент [9], [12];

– використання засобів штучного інтелекту: застосування технології генеративного штучного інтелекту допомагає створювати точний та унікальний контент [9], [12];

– Writesonic пропонує користувачам безкоштовну пробну версію, яка дозволяє безкоштовно створювати текст на кілька тисяч слів. Після вичерпання безкоштовних кредитів, користувачу необхідно підписатися на один із преміальних пакетів, вартістю від 19 доларів США на місяць, що дозволяють генерувати до 60 000 слів і включають різні додаткові програмні функції [9], [12].

Серед недоліків програмного забезпечення Writesonic можна відзначити те, що ця програма для можливості роботи з усіма функціями вимагає оплати, що ускладнює її використання для організацій з невеликим бюджетом. Деякі типи користувачів з невеликим досвідом роботи можуть мати певні проблеми при генерації тексту за допомогою цього програмного забезпечення [9], [12].

Програмне забезпечення Anyslv [9], [13] є текстовим генератором на базі штучного інтелекту, який дозволяє генерувати текстову інформацію. Використовуючи вдосконалену модель штучного інтелекту GPT-3, програмне забезпечення Anyslv може швидко й точно генерувати високоякісний текст [9], [13].

Крім того, програмне забезпечення Anyslv є одним із перших інструментів, який дає прогностичні оцінки продуктивності для кожного типу тексту, який створюється користувачем за допомогою цієї програми [9], [13]. Програмне забезпечення Anyslv дозволяє створити текст за невелику кількість часу, дозволяючи зосередитися на контент-маркетингу чи інших важливих завданнях [9], [13].

Програмне забезпечення Anyslv (рис. 1.3) дозволяє користувачам пристосовувати результат до своїх конкретних потреб за допомогою відповідних засобів настройки програми. Це забезпечується завдяки спеціальному режиму роботи програмного забезпечення Anyslv, який дозволяє

навчити генеративні моделі штучного інтелекту генерувати текст, який відповідає особливостям та потребам конкретного користувача [9], [13].

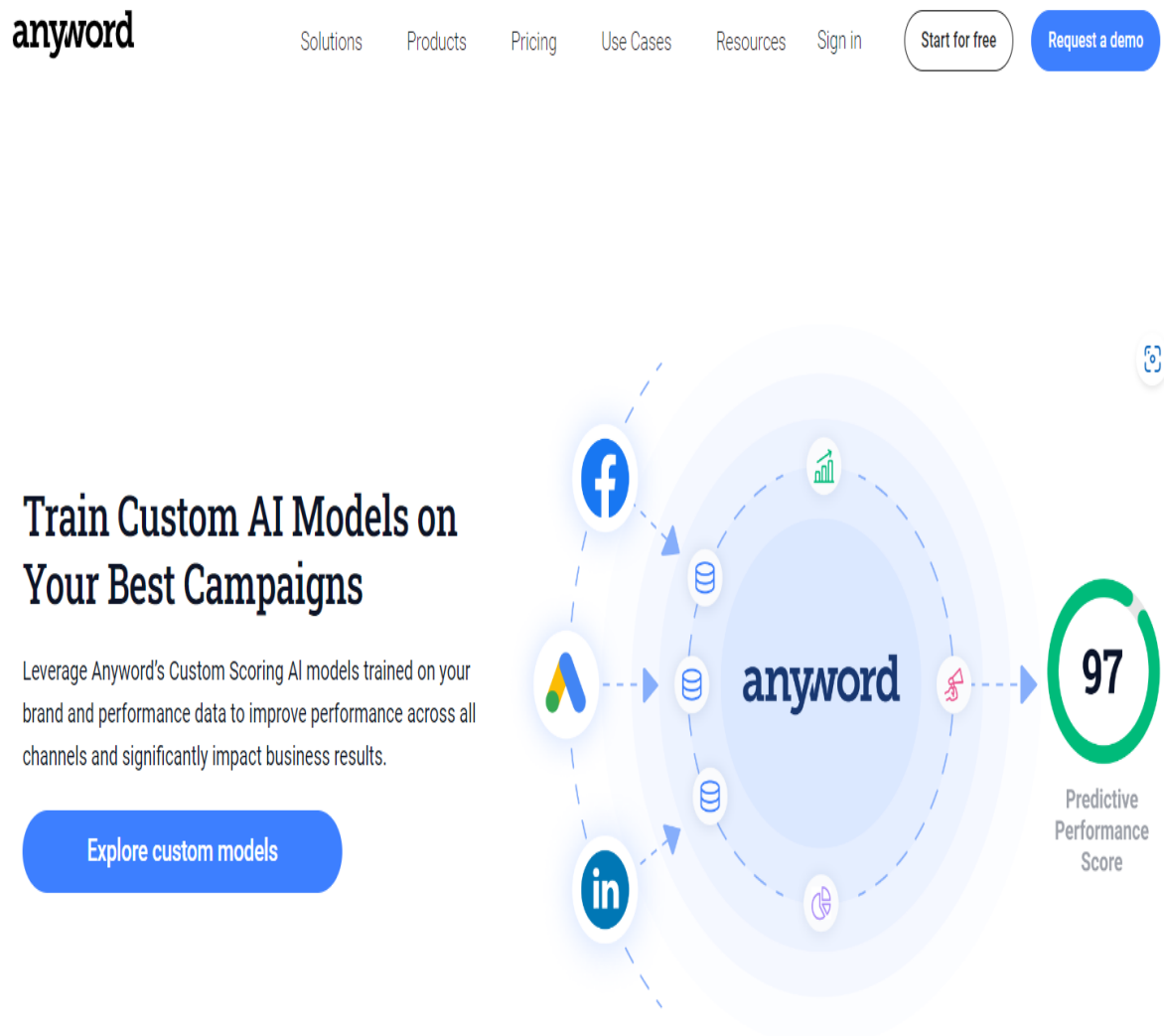


Рисунок 1.3 – Програмне забезпечення Anyslv [9], [13]

Незалежно від того, чи потрібен текст для вебсайту, блогу чи електронної пошти, програмне забезпечення Anyslv може створити його досить швидко без необхідності додаткових налаштувань. Цей текстовий генератор на основі штучного інтелекту підходить для різних компаній або маркетологів, яким потрібно швидко й ефективно створювати великі обсяги вмісту [9], [13].

Серед особливостей програмного забезпечення Anyslv для створення текстового контенту можна виділити такі [9], [13]:

- використання генеративної моделі GPT-3 та мовної моделі T5 для створення тексту [9], [13];
- засоби оцінювання продуктивності генерації тексту [9], [13];
- наявність великої кількості (більше ста) програмних інструментів штучного інтелекту та сценаріїв використання [9], [13];
- функція текстового спілкування, яка дозволяє користувачам створювати контент, шляхом взаємодії зі штучним інтелектом [9], [13];
- підтримка великої кількості мов світу [9], [13];
- засоби інтеграції [9], [13];
- можливість залучати членів команди та керувати кількома проектами одночасно [9], [13];
- модулі генератору тексту є досить зручними та подані у форматі сітки, що полегшує вибір відповідних фільтрів [9], [13];
- інтеграція з граматичними засобами, що полегшує роботи над граматичними помилками (якщо такі є) та перевірки створеного тексту на предмет плагіату [9], [13];
- можливість отримання доступу до файлів, створених із тексту та історії спілкування з генеративною моделлю [9], [13];
- безкоштовна пробна версія, яка за декілька днів дозволяє безкоштовно створити тисячі слів контенту. Після закінчення безкоштовного пробного періоду необхідно підписатися на платний план, вартість якого стартує від 29 доларів США на місяць [9], [13];
- можливість створювати великі тексти, які також може бути подані у розмовному жанрі [9], [13];
- можливість працювати як з засобом розширення з різними веббраузерами [9], [13].

Серед недоліків програмного забезпечення Anyslv можна відзначити те, що ця програма для можливості роботи з усіма функціями вимагає оплати, що

ускладнює її використання для організацій з невеликим бюджетом. Деякі типи користувачів з невеликим досвідом роботи можуть мати певні проблеми при генерації тексту за допомогою цього програмного забезпечення [9], [13].

Порівняльну характеристику програмного забезпечення для генерації текстової інформації наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння програмного забезпечення для генерації текстової інформації

Критерій порівняння	Jasper [11]	Writesonic [12]	Anyslv [13]
Зручність роботи з програмою	+—	+	+
Засоби оцінювання продуктивності генерації тексту	—	—	+
Генеративні моделі для створення тексту	GPT-3	GPT-3, GPT-3.5, GPT-4	GPT-3, T5, CTRL
Кількість інтелектуальних засобів для підтримки роботи користувача	50	90	100
Інтеграція з граматичними засобами	+	—	+

За результатами проведеного аналізу можна зробити висновок, що у наш час існує досить багато програмних засобів для розв’язання завдань генерації текстової інформації. Проте деякі програми для генерації тексту можуть створювати стереотипні або загальні висловлювання, що робить їх менш корисними для унікальних або специфічних завдань, деякі програми можуть іноді генерувати неправильну або недостовірну інформацію, що може призвести до поширення некоректної інформації. Інші подібні програми для досягнення оптимальних результатів та уникнення неправильних висновків програми часто потребують постійного нагляду та втручання від фахівців для

покращення генеративної моделі. Тому актуальною є розробка програмного забезпечення для генерації текстової інформації, зокрема поетичних форм.

При розробці програмного забезпечення для генерації текстової інформації необхідно забезпечити такі функціональні вимоги:

- можливість генерації текстової інформації, зокрема, у вигляді поетичних форм;
- використання інтелектуальних засобів як базис для створення текстового контенту;
- можливість роботи з програмою як в консольному, так і графічному режимі;
- можливість створення словнику з базового тексту;
- можливість задавання параметрів генерованого тексту (наприклад, кількість рядків генерованої поезії).

1.3 Висновки за розділом 1

Визначено, що у наш час широко використовуються програмні засоби для генерації текстової інформації, що допомагають створювати різні тексти, взаємодіяти з користувачами, відповідати на різні запитання та допомагати у вирішенні різних проблем. Показано, що генерація тексту – це метод, яким штучний інтелект створює текст, що наслідує природне спілкування людини. Відзначено, що значення генерації тексту зростає в різних областях, таких як обробка природної мови, створення контенту, обслуговування клієнтів і розробка програмного забезпечення. Показано, що в базисі створення тексту лежать мовні моделі. Проаналізовано практичні застосування генерації тексту за допомогою засобів генеративного штучного інтелекту. Визначено переваги та обмеження програмного забезпечення створення текстів. Проаналізовано процес розробки програмного забезпечення для створення текстів на основі генеративних моделей. Відзначено, що генеративний інтелект штучний інтелект представляє собою важливу галузь штучного інтелекту та машинного

навчання, яка використовується для створення програмного забезпечення, здатного генерувати штучний контент, наприклад, текстові матеріали, зображення, аудіо та відео. Проаналізовано техніки генеративного штучного інтелекту для синтезу генеративних мовних моделей.

За результатами проведеного аналізу зроблено висновок, що у наш час існує досить багато програмних засобів для генерації текстової інформації. Проте деякі програми для генерації тексту можуть створювати стереотипні або загальні висловлювання, що робить їх менш корисними для унікальних або специфічних завдань, деякі програми можуть іноді генерувати неправильну або недостовірну інформацію, що може призвести до поширення некоректної інформації. Інші подібні програми для досягнення оптимальних результатів та уникнення неправильних висновків програми часто потребують постійного нагляду та втручання від фахівців для покращення генеративної моделі. Тому актуальною є розробка програмного забезпечення для генерації текстової інформації.

Сформульовано функціональні вимоги до програмного забезпечення для генерації текстової інформації.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування

При розробці програмного забезпечення для генерації текстової інформації було використано мову програмування Python [14], [15].

Python є мовою програмування загального призначення, спрямована на покращення продуктивності розробника та зрозумілість коду [14], [15].

Унікальність синтаксису Python не обмежується. Стандартна бібліотека містить безліч корисних функцій. Python підтримує кілька підходів до програмування, включаючи структурне, об'єктно-орієнтоване, функціональне та імперативне [14], [15].

Важливі особливості включають динамічний тип даних, автоматичне управління пам'яттю, повну перевірку на виключення, засоби обробки винятків, підтримку багатопотокових операцій і зручну структуру даних високого рівня [14], [15].

В Python програмний код організовується у вигляді функцій та класів, які можна об'єднувати в програмні модулі, що, в свою чергу, можна об'єднати в програмні пакети [14], [15].

Python існує у різних реалізаціях, одна з яких – це інтерпретатор CPython, розроблений для підтримки різних платформ. Він поширюється під ліцензією Free Python Software Foundation, що дає можливість використовувати його в будь-яких програмах. Крім CPython, існують інші реалізації для JVM, MSIL, LLVM і таке інше. Проект Piri дозволяє використання Python над Python, знижуючи витрати на перемикання мови та тестування нових можливостей [14], [15].

Python – це мова програмування, що постійно розвивається, і нова версія (із змінами/модифікаціями) виходить приблизно кожні два з половиною роки. Відсутність офіційного стандарту ANSI, ISO або іншого, поряд із постійним розвитком, зробила CPython лідером серед інших реалізацій. Python

підтримує динамічний тип, що означає, що типи змінних визначаються тільки під час виконання програми [14], [15].

Вбудовані типи Python включають логічні значення, рядки, рядки Unicode, цілі довільної точності, плаваючі числа, комплексні числа та інші. В ньому також присутні вбудовані колекції, такі як списки, кортежі, словники та інші структури даних. У Python усе є об'єктом, включаючи функції, методи, модулі та класи. Крім того, ви можете створити новий тип, написавши клас або визначивши його в модулі розширення (наприклад, на мові C) [14], [15].

Система класів Python підтримує успадкування (одиначне та множинне) та мета програмування. Більшість вбудованих типів та типів розширень можна успадковувати [14], [15].

Усе у Python можна розділити на атоми і архіви. Атоми - це `int`, `long`, `complex` та інші елементи. При визначенні атомарних об'єктів їх значення копіюються, але для посилальних об'єктів копіюється лише вказівник на об'єкт. Таким чином, після визначення обидві змінні використовують одне і те ж значення. Можливо, посилальний об'єкт буде змінюватися, але це може не стосуватися всіх випадків. Наприклад, рядки та кортежі є незмінними, але списки, словники та інші об'єкти можуть змінюватися [14], [15].

Кортежі у Python в основному є незмінними списками. У більшості випадків кортежі працюють швидше, ніж списки, тому їх доцільніше використовувати, якщо не плануєте змінювати порядок елементів [14], [15].

Типи Python можуть бути різними, зокрема, є можливість використання логічних значень, рядків, цілих довільної точності, плаваючі числа, комплексні числа та інші. В ньому також присутні вбудовані колекції, такі як списки, кортежі, словники та інші структури даних. У Python усе є об'єктом, включаючи функції, методи, модулі та класи. Крім того, ви можете створити новий тип, написавши клас або визначивши його в модулі розширення (наприклад, на мові C) [14], [15].

Обґрунтування вибору мови програмування для розробки програмного забезпечення для генерації текстової інформації наведено у таблиці 2.1.

Таблиця 2.1 – Обґрунтування вибору мови програмування для розробки програмного забезпечення для генерації текстової інформації

Критерій порівняння мов програмування	Мова програмування		
	Python	Matlab	C++
Зручність розробки програмного забезпечення для генерації текстової інформації	+	—	—
Робота з текстовими даними	+	+—	+
Засоби спадкування	+	+—	+
Засоби генеративного штучного інтелекту	+	+—	+—
Наявність бібліотек для машинного навчання	+	+	+—

Отже, для реалізації програмного забезпечення для генерації текстової інформації обрано мову програмування Python, яка є мовою програмування загального призначення, спрямована на покращення продуктивності розробника і зрозумілість коду та дозволяє ефективно працювати з текстовими даними.

2.2 Вибір середовища розробки для генерації текстової інформації

В якості середовища розробки для створення програмного забезпечення для генерації текстової інформації було обрано PyCharm [16], [17].

PyCharm є інтегрованим середовищем розробки, призначеним для мови програмування Python, яке має в собі багато корисних функцій. Воно надає засоби для аналізу коду, візуального налагодження, тестування модулів, а також підтримує веброзробку в Django та Flask. Це програмне забезпечення доступне для операційних систем Windows, Mac OS X та Linux [16], [17].

PyCharm, розроблене компанією JetBrains, є ідеальним інструментом для програмістів, які працюють з мовою програмування Python. Воно не лише надає широкий спектр корисних функцій, але й пропонує засоби для аналізу та оптимізації коду, що дозволяє поліпшити продуктивність розробки програмного забезпечення [16], [17].

Однією з головних переваг середовища розробки PyCharm є його вбудовані можливості візуального налагодження, які спрощують виявлення та виправлення помилок у коді. Платформа також має інструменти для ефективного тестування модулів, що сприяє підтримці високої якості програмного продукту [16], [17].

Потужність середовища розробки PyCharm проявляється у його здатності підтримувати веброзробку в популярних фреймворках, таких як Django та Flask. Це робить це середовище незамінним для тих, хто працює над проектами веброзробки [16], [17].

Це інтегроване середовище розробки доступне для операційних систем Windows, Mac OS X та Linux, що робить його універсальним і зручним для розробників, що використовують різні платформи. PyCharm - це не просто інструмент, він - ваш вірний помічник у створенні та оптимізації програмного забезпечення, забезпечуючи широкий спектр інструментів та функцій для роботи з Python [16], [17].

Основні можливості PyCharm такі [16], [17]:

- статичний аналіз коду, підсвічування синтаксичних помилок;
- проста навігація у проекті та вихідному коді для швидкого переходу між різними структурами, файлами, класами та методами;

- функції рефакторингу, такі як перейменування, видалення, додавання змінних, методів та констант;
- інструменти для веброзробки на фреймворку Django;
- вбудований відладчик для Python;
- інтегровані засоби модульного тестування;
- підтримка Google App Engine;
- робота з системами контролю версій, такими як Mercurial, Git, Subversion, Perforce і CVS через загальний інтерфейс [16], [17].

Обґрунтування вибору середовища розробки для створення програмного забезпечення для генерації текстової інформації наведено у таблиці 2.2.

Таблиця 2.2 – Обґрунтування вибору середовища розробки для створення програмного забезпечення для генерації текстової інформації

Критерій порівняння середовищ розробки	Середовище розробки		
	PyCharm	Visual Studio	Sublime
Зручність розробки на мові Python	+	+-	+
Засоби для аналізу коду	+	+	+-
Засоби візуального налагодження	+	+	-
Зручність для створення ПЗ для генерації текстової інформації	+	-	-
Проста навігація у проєкті та вихідному коді	+	-	-

Таким чином, для створення програмного забезпечення для генерації текстової інформації обрано середовище розробки PyCharm, що являє собою

ефективний інструмент для створення програм, написаних на мові програмування Python та включає в себе основні класи для роботи з генеративним штучним інтелектом, потрібні для розробки програмних застосунків генерації текстової інформації.

2.3 Методи та моделі машинного навчання для генерації текстової інформації

Машинне навчання представляє собою групу інтелектуальних методів, які спроможні вчитися на основі даних для виявлення різноманітних зв'язків та закономірностей. Для створення таких методів використовуються різноманітні математичні методи, включаючи теорію ймовірностей, чисельні алгоритми, статистичні та оптимізаційні методи, а також теорію графів [18, 19].

У практичних завданнях вхідні дані можуть бути неповними, неточними, нечисловими та неоднорідними, що породжує потребу в застосуванні різних методів машинного навчання [18, 19].

Після освоєння концепцій алгоритми машинного навчання можуть бути використані для вирішення завдань створення поетичних форм. Серед низки алгоритмів, які частково розв'язують цю задачу, можна виділити: кластеризація k -середніх; нейронні мережі; опорні вектори; генетичні алгоритми; класифікатор Байєса [18, 19].

Метод кластеризації K -середніх відображає підхід у групуванні набору об'єктів у відносно однорідні класи. Основна мета цього методу полягає в розділенні n спостережень на k груп так, щоб кожне спостереження належало до того кластера, чий центр найбільше збігається з ним. Сутність методу полягає у тому, щоб мінімізувати суму квадратів відстаней між кожним спостереженням та центром його кластера [18, 19].

Метод кластеризації K -середніх розпочинається з визначення k початкових точок даних як центри кластерів. Далі повторюються наступні

кроки до завершення процесу оновлення центрів кластерів. Призначаються точки даних до кластеру, який має найближчий центр. Кластер повинен містити хоча б одну точку. Якщо кластер порожній, йому призначається випадкова точка, що знаходиться далеко від центру кластеру. Далі оновлюється центр кожного кластера, встановивши його як середнє значення точок у кластері. Процес повторюється, поки відбуваються зміни в центрах кластерів, що є інструкцією про продовження виконання циклу або завершення алгоритму, але в даному описі він може бути не пов'язаним з алгоритмом [18, 19].

У цьому методі при обробці природної мови розглядаються слова, які необхідно навчити, як точки даних. Кожен кластер містить слова з однаковим коренем. Таким чином, ми намагаємось розбити словник на групи слів з однаковими коренями та визначити невикористані корені. Основні переваги методу К-середніх полягають у його простоті та швидкості виконання. Цей метод добре підходить для групування великих наборів даних (у нашому випадку – слів). Незважаючи на його переваги, метод має декілька значних недоліків. Результат класифікації залежить від вибору початкового кореня слова. Кількість кластерів (коренів) має бути відома заздалегідь, що обмежує деталізацію класифікації [18, 19].

Отже, метод k-середніх добре працює для великих обсягів даних, якщо дані узгоджені, тобто, коли кластери вже існують. Але якщо правильний кластер не визначений, це може призвести до серйозних помилок, що суттєво вплине на продуктивність програмної системи [18, 19].

Одним з методів у машинному навчанні вчителів є застосування нейронних мереж [19], які представляють собою складні системи з простими комп'ютерними вузлами, що взаємодіють між собою. Один з типів цих мереж – рекурентні нейронні мережі, які мають зворотний зв'язок та здатні зберігати інформацію [18, 19].

Однією з переваг рекурентних нейронних мереж є їхній здатний працювати з довгими контекстами. Кожен раз, коли у клітинку надходить на

вхід слово, вона використовує вектор, що представляє всю попередню історію – всі слова, які були оброблені до цього моменту [18, 19].

При формуванні текстового контенту важливо підібрати найвідповідніший варіант з певного набору слів. Розглянемо тип рекурентної нейронної мережі, що має назву довгострокова пам'ять (Long Short Term Memory, LSTM) і вимагає здатності зберігати довгий контекст. Кожен раз, коли нове слово подається на вхід у рекурсивний вузол, ми маємо інформацію про всі попередні слова [18, 19].

Для вирішення задачі потрібна мовна модель, яка визначає найбільш підходяще наступне слово зі створеної послідовності. Це означає, що ми уникнемо використання непідготованої інформації під час передбачення. Отже, ми контролюємо кількість слів у словнику, щоб налаштувати розмір моделі [18, 19].

Завжди існують проблеми з архітектурою мережі. Можна використовувати модель багато до багатьох (спробувати передбачити наступне слово для кожного слова) або багато до одного (передбачати одне слово, виходячи з послідовності перших слів) [18, 19]. Отримана модель безсумнівно здатна генерувати текстові форми, але вони можуть бути лишені граматичної логіки та значення [18, 19].

Метод опорних векторів у машинному навчанні – це спосіб аналізу даних для класифікації та регресійного аналізу, що базується на контрольованих моделях і використанні пов'язаних алгоритмів навчання, відомих як машини опорних векторів (МОВ). Зокрема, ця мережа підтримує векторні мережі, спрямовані на англійську мову [18, 19].

При наявності набору навчальних прикладів, які розмічені як належність до двох класів, метод навчання МОВ будує модель, яка приписує нові приклади до певного класу. Це зазвичай створює два роздільні класи за допомогою двійкового лінійного класифікатора. МОВ – це представлення зразків у вигляді точок у просторі моделі, таких, що зразки різних класів максимально відділені один від одного. Потім нові приклади порівнюються з

цим розділом, щоб передбачити їх клас, відповідно до того, до якого боку розділювальної лінії вони належать [18, 19].

Крім лінійної класифікації, МОВ може успішно виконувати нелінійну класифікацію за допомогою так званого ядерного трюку, який неявно перетворює вхідні дані до простору функцій великої розмірності [18, 19].

У випадку відсутності міток на даних, навчання під керівництвом не можливе, і застосовується навчання без нагляду. Його мета – знайти природні групи в даних і групувати їх, а потім віднести нові дані до вивчених груп. Алгоритм кластеризації, що підвищує метод опорних векторів, називається кластеризацією опорних векторів і часто застосовується у промислових застосунках, де дані не мають міток або частину даних спершу класифікують, перед тим як вони будуть відправлені на класифікацію. Це відомо як переднє оброблення [18, 19]. Кластеризацію опорних векторів можна використовувати для розв'язання різних практичних завдань [18, 19].

МОВ корисні для класифікації тексту, зменшуючи потребу у мітках у стандартних індуктивних контекстах. Класифікація зображень також може бути здійснена за допомогою МОВ. Експериментальні результати свідчать, що МОВ досягає вищої точності розпізнавання вже після трьох-чотирьох раундів пошуку збігів порівняно з традиційними методами поліпшення виявлення. Це стосується й систем сегментації зображень, включаючи модифіковану версію МОВ на основі методу пріоритетів [18, 19]. Алгоритм МОВ широко використовується в біології та інших науках. Послідовні тести на основі вагових коефіцієнтів МОВ пропонуються як спосіб інтерпретації моделі МОВ. Ваги МОВ раніше використовувалися для прогнозування точності моделей. Прогностична інтерпретація машинних моделей опорних векторів для визначення особливостей, які модель використовує для прогнозування, є відносно новою областю досліджень [18, 19].

Генетичний алгоритм – це метод пошуку, який використовує принципи еволюції для вирішення проблем оптимізації та моделювання. Основні риси

цього методу полягають у використанні кросинговеру, що регулює поєднання варіантів рішень, аналогічно до кросинговеру в біологічних системах [18, 19].

Проблему кодують у формі таблиці, що відображає аналогію зі складом хромосом. Ця структура часто отримує назву хромосома [18, 19]. Початкова популяція фіксованої кількості хромосом або особин вибирається випадковим чином і представляється у вигляді таблиці. Кожну особину оцінюється за допомогою функції придатності, яка присвоює кожній особині визначене значення відповідності для виживання. Далі основуючись на отриманих значеннях відповідності, відбираються особини для подальшого схрещування (відбору) [18, 19].

Хромосоми при генетичному пошуку піддаються генетичним операціям, зазвичай це схрещування і мутація, що породжує нові покоління особин. Після цього нові покоління оцінюються за допомогою цих генетичних операцій, і відбір та трансформація завершуються [18, 19].

Отже, ми моделюється еволюційний процес, що складається з кількох життєвих циклів (поколінь), поки алгоритм не відповідає критеріям завершення [18, 19].

Ці критерії можуть бути: пошук глобальних або оптимальних рішень, скорочення кількості поколінь у процесі еволюції, мінімізація марно витраченого часу на еволюцію [18, 19].

Генетичні алгоритми можуть застосовуватись для пошуку рішень у великих і складних областях пошуку. Ці алгоритми допомагають вирішувати різноманітні задачі, такі як оптимізація функцій, вдосконалення запитів до баз даних, робота з різнокольоровою графікою, розробка та тренування штучних нейронних мереж, функції верстки для різних застосувань, розробка графіків, координація діяльності, продуктивний термін служби, для моделювання складності білкових структур [18, 19].

Іншим методом машинного навчання є наївний байєсівський класифікатор, що використовує прості ймовірнісні розрахунки на основі

теореми Байєса та припущення про функціональну незалежність параметрів моделі [18, 19].

Якщо значення змінних однозначно вказують на категорію спостереження, то байєсівський класифікатор повідомляє ймовірність його віднесення до цієї категорії. У складніших випадках, коли спостереження можуть належати до різних класів з різною ймовірністю, результатом класифікатора стає вектор, що містить ймовірності віднесення до кожного класу [18, 19].

Байєсівський класифікатор вважається досить ефективним інтелектуальним засобом. Він не може уточнити вихідні дані, оскільки у всіх невизначених ситуаціях він дає відповідь, вимірюючи ступінь цієї невизначеності [18, 19].

Для створення віршованих форм необхідно відокремити слова, на які можна застосувати цей метод. Він є простим у засвоєнні та використанні, а також відкриває можливості для подальших досліджень під час роботи. Якщо є словники груп слів, важливо вивчити взаємозв'язки слів, щоб надати значення віршованим формам [18, 19].

Перевага такого підходу полягає в тому, що необхідний обсяг вибірки зменшується дуже швидко, не враховуючи ускладнень. Недоліком є те, що модель працює оптимально лише при умові виконання припущення про функціональну незалежність [18, 19].

Однак на практиці навіть невелике відхилення від незалежності призводить лише до дрібної втрати точності при використанні генеративних моделей, і навіть при значній залежності між змінними результати роботи класифікатора відображають справжній зв'язок зображення з класом. У той же час переваги таких класифікаторів, як швидкість роботи, простота використання та можливість масштабування, а також помірні вимоги до пам'яті, часто переважають їх недоліки [18, 19].

2.4 Розробка моделі взаємодії користувача з програмним забезпеченням генерації текстової інформації

З метою можливості розробки програмного забезпечення для генерації текстової інформації створимо відповідну модель у вигляді діаграми прецедентів.

При розробленні моделі взаємодії користувача з програмним забезпеченням для генерації текстової інформації враховуємо те, що користувачі програми мають можливості генерації текстової інформації, зокрема, у вигляді поетичних форм, використання інтелектуальних засобів як базис для створення текстового контенту, роботи з програмою як в консольному, так і графічному режимі, створення словнику з базового тексту, задавання параметрів генерованого тексту (наприклад, кількість рядків генерованої поезії).

Основний успішний сценарій користувача при роботі з програмним забезпеченням для генерації текстової інформації можна описати таким чином:

- користувач програмного забезпечення для генерації текстової інформації запускає програму, обираючи один з двох режимів програми (консольний чи графічний);

- після запуску програми для генерації текстової інформації користувачу відображається головна форма (у випадку графічного режиму роботи програми);

- користувач програми для генерації текстової інформації бачить головну форму та завантажує генеративну модель;

- користувач вводить параметри генерованого тексту (наприклад, кількість рядків генерованої поезії);

- користувач здійснює генерацію текстової інформації, наприклад, у вигляді поетичних форм, шляхом натискання відповідної кнопки у програмі;

- користувач бачить результат роботи програми, зокрема створений текстовий контент (наприклад, поезію);
- користувач завершує роботу з програмним забезпеченням для генерації текстової інформації.

Враховуючи сценарії роботи користувачів при роботі з програмним забезпеченням для генерації текстової інформації, модель взаємодії користувача з програмним забезпеченням для розв'язання задачі генерації текстової інформації, зображено на рис. 2.1.

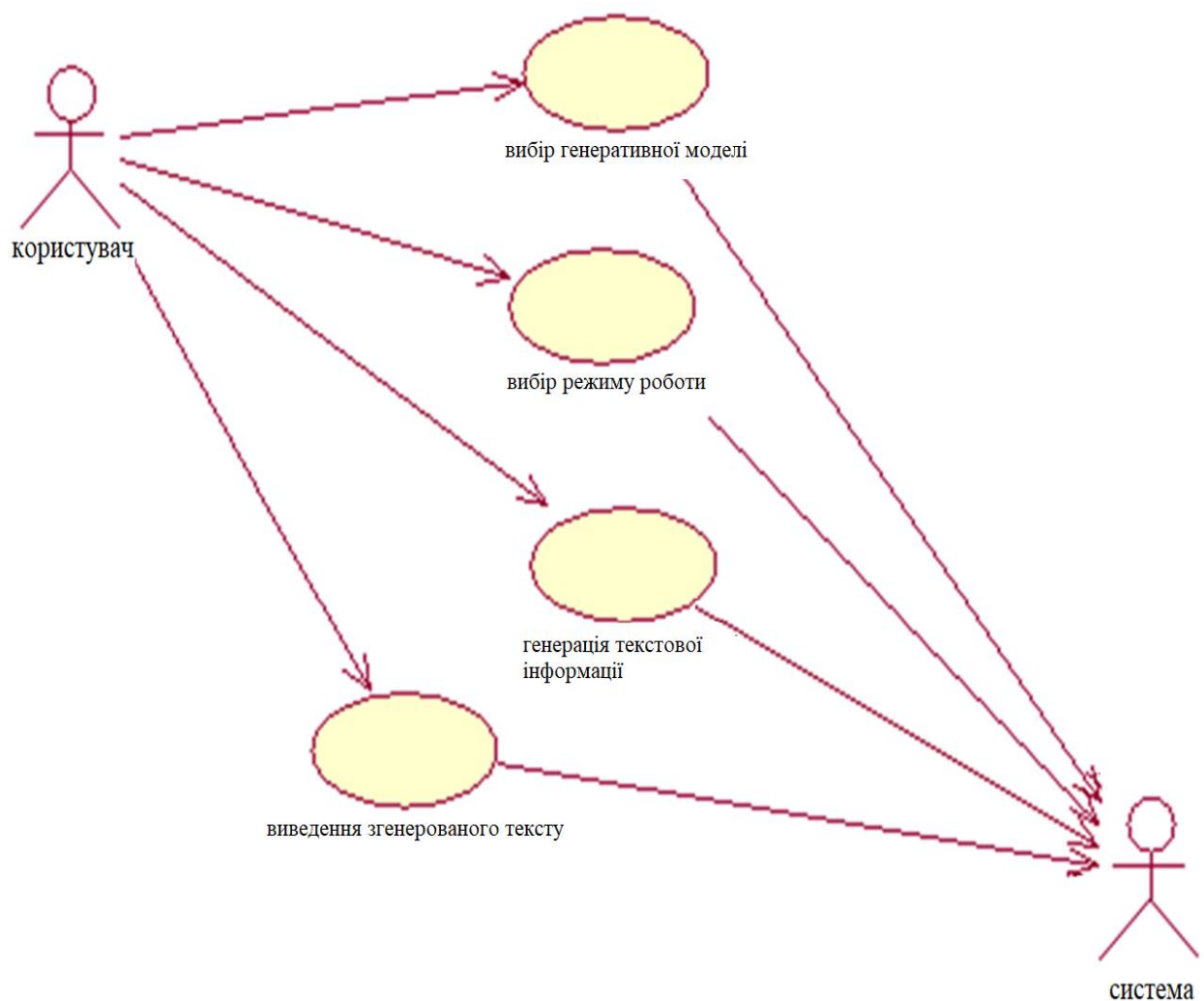


Рисунок 2.1 – Модель взаємодії користувача з програмним забезпеченням для генерації текстової інформації

Таким чином, створено модель взаємодії користувача з програмним забезпеченням для генерації текстової інформації, яка подана у вигляді діаграми прецедентів, та дозволяє розроблювати програмне забезпечення для генерації текстової інформації.

2.5 Висновки за розділом 2

Для реалізації програмного забезпечення для розв'язання завдання генерації текстової інформації обрано мову програмування Python, яка є мовою програмування загального призначення, спрямована на покращення продуктивності розробника і зрозумілість коду та дозволяє ефективно працювати з текстовими даними.

Для створення програмного забезпечення для розв'язання задачі генерації текстової інформації обрано середовище розробки PyCharm, що являє собою ефективний інструмент для створення програм, написаних на мові програмування Python та включає в себе основні класи для роботи з генеративним штучним інтелектом, потрібні для розробки програмних застосунків генерації текстової інформації.

Створено модель взаємодії користувача з програмним забезпеченням для генерації текстової інформації, яка подана у вигляді діаграми прецедентів, та дозволяє розроблювати програмне забезпечення для генерації текстової інформації.

3 ОПИС ПРОГРАМИ

3.1 Структура програмного забезпечення для генерації текстової інформації

Структуру програмного забезпечення для генерації текстової інформації подано на рис. 3.1.



Рисунок 3.1 – Структура програмного забезпечення для генерації текстової інформації

Як видно з рисунку, програмне забезпечення для генерації текстової інформації складається з таких компонентів:

- модуль парсингу словнику слів на основі наявної текстової інформації `dictnry`;
- модуль генерації текстової інформації `txtgener`;
- модуль забезпечення інтерфейсу користувача `intrfcLMV`;
- головний модуль програмного забезпечення для генерації текстової інформації `MUTG`.

Модуль, який відповідає за парсинг словника, призначений для формування словнику слів на основі наявної текстової інформації.

Модуль генерації текстової інформації використовує наявні генеративні моделі, а також базові запити користувача для створення текстового контенту.

Модуль забезпечення інтерфейсу користувача відповідає за підтримку взаємодії між користувачем та програмним забезпеченням для генерації текстової інформації

Відзначимо, що програмне забезпечення для генерації текстової інформації містить у собі й інші програмні модулі.

3.2 Функціонування програмного забезпечення для генерації текстової інформації

Схема функціонування програмного забезпечення для генерації текстової інформації наведена на рис. 3.2.

Як видно з рис. 3.2 на першому етапі відбувається формування словнику зі словами. Вхідні дані надходять у вигляді посилання на сторінку в Інтернет з цим словником. Проводиться вибір кожної сторінки словника, після чого з кожної сторінки відбирається слово з позначеним наголосом, можливі закінчення та слова, які пов'язані з основним словом.

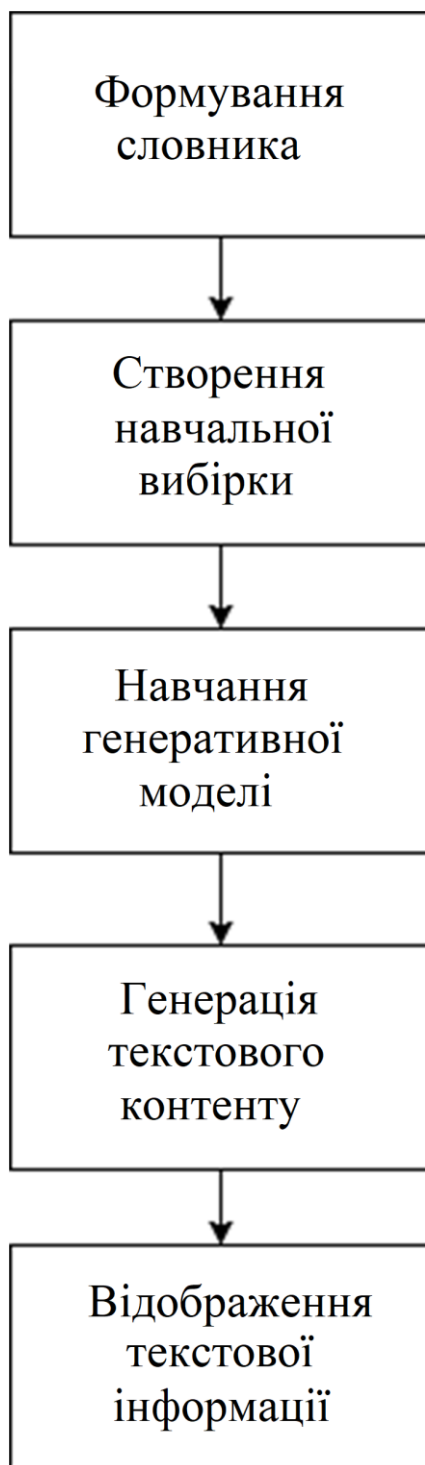


Рисунок 3.2 – Схема функціонування програмного забезпечення для генерації текстової інформації

Усі ці дані заносяться до бази даних. На виході отримуємо оновлений словник зі словами та заповнену базу даних.

На другому етапі відбувається створення навчальної вибірки. З тексту твору випадковим чином вибираємо кілька тисяч слів, які відсутні у словнику.

Потім користувач повинен ввести аналогічне за значенням слово, що відрізняється лише закінченням, префіксом або суфіксом. Отримана вибірка випадково розподіляється на дві частини: 80% для тренування та 20% для перевірки навчання. Як результат сформовано два набори пар слів - один для навчання, інший для перевірки ефективності навчання.

Далі виконується навчання генеративної моделі. На цьому етапі маємо два набори пар слів з попереднього етапу та словник із першої стадії. Кожна пара слів першого набору проходить аналіз, де перше слово порівнюється із другим за кількома критеріями: присутність префіксу, спільність кореневих літер, наявність відповідного закінчення. Потім в словнику знаходиться слово, яке максимально відповідає цим критеріям. Це допомагає визначити важливість та ймовірність кожного параметру та їх комбінацій. Далі проводиться перевірка класифікації на другому наборі слів. Як результат отримуються ймовірності параметрів слів та їх комбінацій.

Як базис для синтезу текстового контенту доцільно використовувати рекурентні нейронні мережі (рис. 3.3), що мають у своєму складі нейрони із зворотними зв'язками, а як входи приймають інформацію про базовий текст, на основі якого необхідно створювати новий текст.

Після цього відбувається генерація текстового контенту на основі наявної генеративної моделі. На цьому етапі як вхідну інформацію програмна система використовує розраховані ймовірності з попереднього етапу та текст твору. Тут відбувається аналіз літературного твору, вивчення взаємозв'язків та взаємного розташування слів. Для визначення ваги використовують позицію слова у тексті, його місце в абзаці, частоту та взаємне розташування. Якщо слово не міститься у словнику, воно класифікується за допомогою класифікатора.

На останньому етапі відбувається виведення створеного текстового контенту у зручному для користувача вигляді.

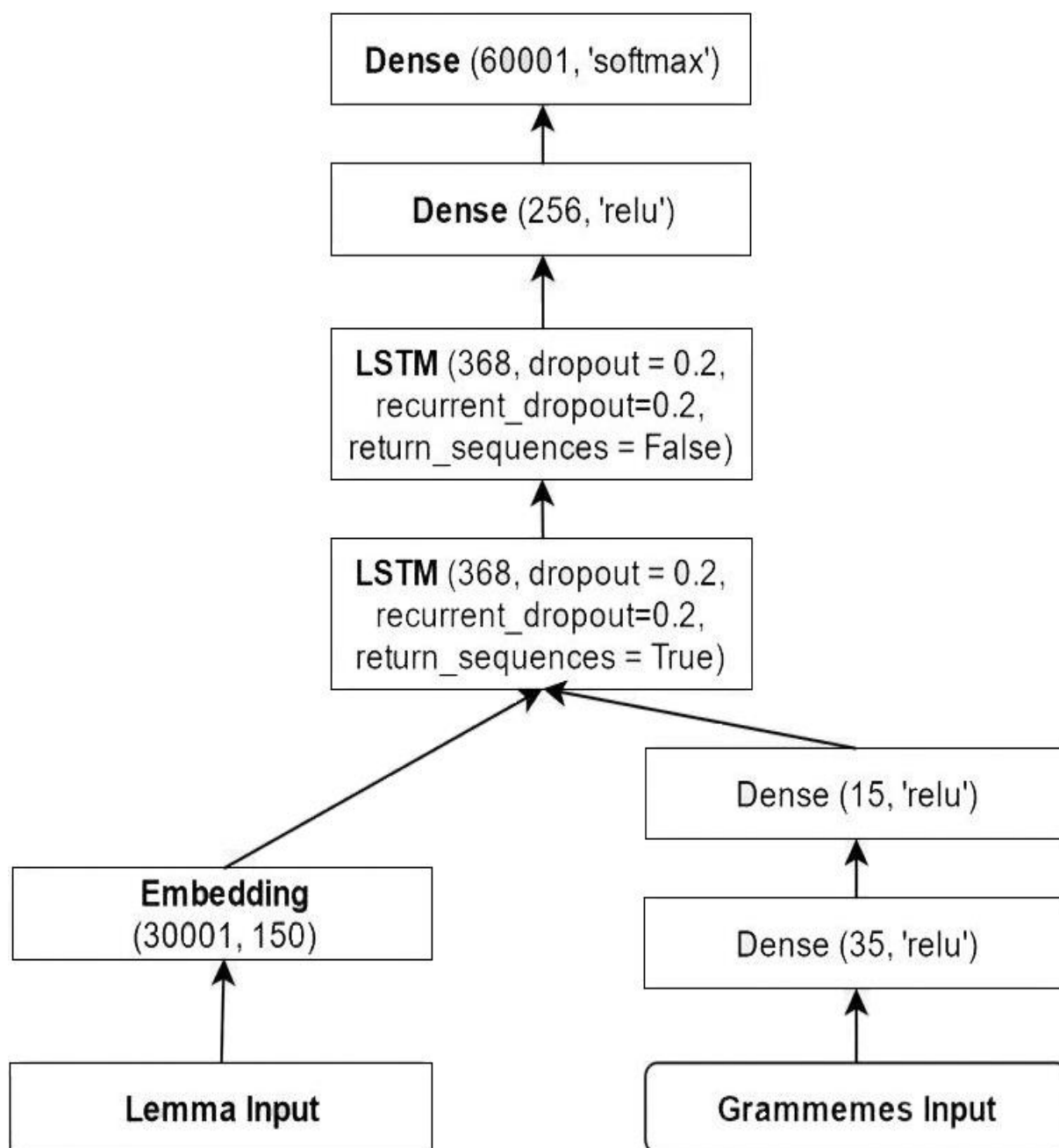


Рисунок 3.3 – Модель для генерації текстової інформації

3.3 Основні рішення щодо розробки програмного забезпечення для генерації текстової інформації

Фізичну структуру програмного забезпечення для генерації текстової інформації подамо у вигляді дерева, що містить набір основних файлів та каталогів програми (рис. 3.4).

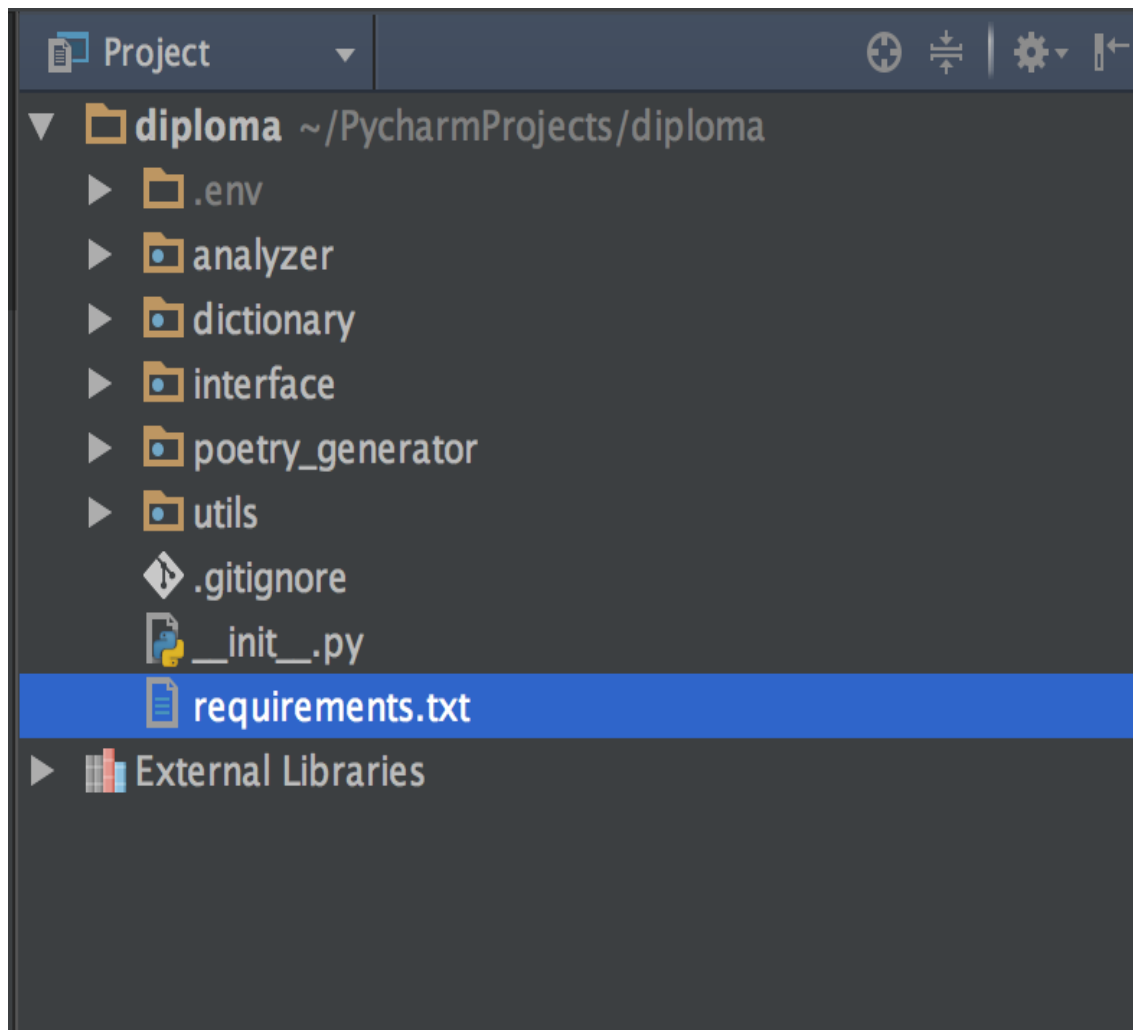


Рисунок 3.4 – Фізична структуру програмного забезпечення для генерації текстової інформації

Реалізоване програмне забезпечення для генерації текстової інформації містить такі основні модулі:

- модуль парсингу словнику слів на основі наявної текстової інформації dictnry;
- модуль генерації текстової інформації txtgener;
- модуль забезпечення інтерфейсу користувача intrfcLMV;
- головний модуль програмного забезпечення для генерації текстової інформації MUTG.

Призначення модулів програмного забезпечення для генерації текстової інформації описано у табл. 3.1.

Таблиця 3.1 – Опис модулів проекту

Назва	Призначення	Файли
dictnry	Синтез словнику з наявного набору слів	scrpy.cfg dictnry_Prsr/scrpy.cfg dictnry_Prsr/ustanovk.py dictnry_Prsr/Elemns.py dictnry_Prsr/mdlwrs.py dictnry_Prsr/Pplns.py dictnry_Prsr/Pwks/dictnry.py
intrfcLMV	Забезпечення взаємодії з користувачем	intrfcLMV.py templates/indx.html static/main.css static/main.js
txtgener	Генерація текстового контенту	hener.py txtstrat/ampg.py txtstrat/base.py txtstrat/iamb.py txtstrat/treed.py txtstrat/dctl.py
koefkor	Обчислення числових показників взаємозв'язку слів	koefkor.py koefkor_rukvd.py
dictnry_generat	Генерація контенту	generat.py generat_navching.py serlz.py slovoman.py
dopomdop	Допоміжний модуль з додатковими модулями та структурами	classes.py db.py fltr_db.py fncts.py local_ustanovk.py ustanovk.py navchsample.py

Опис основних файлів проекту програмного забезпечення для генерації текстової інформації описано у табл. 3.2.

Таблиця 3.2 – Опис файлів проєкту програмного забезпечення для генерації текстової інформації

Назва модулю	Назва файла	Структури даних та функції, розташовані у відповідному програмному модулі
dictnry	Elems.py	DictnryPrsrElem
	mdlwrs.py	DictnryPrsrPwkMdlwr
	Pplns.py	DictnryPrsrPpln
	Pwks/dictnry.py	DictnryPwk
intrfcLMV	Stvoren.py	hener indx
	Static/main.js	hener openLogo showPreldrer hidePreldrer onChngCnt henerUstanovkForCplets syntezUstanovk syntezSetting clearUstanovk
txtgener	hener.py	hener_tofl hener_random_txt
	txtstrat/ampg.py	AmpgTxtStrat
	txtstrat/base.py	BaseTxtStrat
	txtstrat/dctl.py	DctlTxtStrat
	txtstrat/iamb.py	IambTxtStrat
	txtstrat/treed.py	TreedTxtStrat
koefkor	Koefkor_rukvd.py	KoefkorRukvd
	Koefkor.py	gtabzcs spltb_pnct gtpredlzhs gtslvs_in_predlzh gtdictnry_slv

Продовження таблиці 3.2.

Назва модулю	Назва файла	Структури даних та функції, розташовані у відповідному програмному модулі
koefkor	Koefkor.py	vstvslvs vstvslv vstvkoefkor vstvkoefkors dobkoefkor dobpredlzh_koefkor dobabzc_koefkor crltabzc crcfc
dictnry_generat	Generat.py	gtchnc gener gtvertties gtsstnavching_stnctes set_vertties init_vertties tstgenert
	Generat_navching.py	navch chckprfx chckending gtnavching_stnctes wrtvertties_tofl nachs_generat_navching
	serlz.py	dsrlz_vertties_ffl dsrlz_json_ffl
	slovoman.py	SlvRukvd
dopomdop	classes.py	Slv Hypt Vertties Koefkor
	db.py	cnctto_db vstvdictnrycmnd vstvnew_dictnrycmnd vstvkoefkorcmnd vybotrall_slvscmnd delkskd_slvscmnd delslvscmnd vybotrall_slvs vybotrall_koefkors delslv

Продовження таблиці 3.2.

Назва модулю	Назва файла	Структури даних та функції, розташовані у відповідному програмному модулі
dopomdop	fltr_db.py	fltr_dictnry_tblc
	fncts.py	is_ukr_slv gtslv_verttiesfl
	navchsample.py	gtslvs gener_dopomer

Для роботи класифікатора необхідний словник слів та навчальні дані у вигляді пар слів: слово для навчання – слово, яке є найближчим за сенсом та граматиною, що вже присутнє у словнику. Кожну пару слів аналізують, порівнюючи перше слово із другим за рядом параметрів, таких як префікси, кількість спільних кореневих літер, належне закінчення тощо. У мові є певна кількість префіксів, тому цей параметр визначається просто. Словник містить різні закінчення кожного слова, тому це також важливий параметр. Кількість спільних кореневих літер – складний параметр для аналізу через неможливість простого підрахунку літер. Також важливо враховувати особливості мови, наприклад, часте записування однієї літери як інша та інші морфологічні відмінності.

Після аналізу встановлюється, які слова у словнику мають схожі параметри. Це дозволяє визначити значимість та ймовірність кожного параметру та їх поєднань.

Для класифікації слова порівнюється з усіма словами у словнику, обираючи семантичну групу, де отримана ймовірність за методом Байєса найвища, якщо ця ймовірність перевищує мінімальне значення, знайдене в експерименті. Кожна група – це семантична категорія слів, які мають спільний смисл, відрізняючись закінченням, частиною мови або відмінюванням.

Система отримує низку параметрів для створення текстового контенту, наприклад, вірша, включаючи тип поезії, кількість рядків, кількість слів і багато іншого.

Спочатку система вибирає перше слово з вказаних параметрів або генерує випадкове слово, враховуючи усі умови, такі як наголос і кількість складів.

На кожному кроці система намагається знайти шлях у графі відносин таким чином, щоб кожне слово відповідало вибраному типу поезії і відповідало римі. Однак через велику кількість слів система робить припущення щодо відповідності слова, тому рима може бути порушена.

Для створення словника використовується словник відповідної мови. Цей словник, як правило, містить десятки тисяч слів, вказані наголоси для кожного слова, різні можливі закінчення та альтернативні варіанти. У ролі навчального та тестового матеріалу виступає літературний твір. Для створення навчальної та тестової вибірки з літературного твору випадковим чином відбираються тисячі слів, які відсутні у словнику. Потім користувачу потрібно ввести слово, що має аналогічне значення до введеного, але відрізняється закінченням, префіксом або суфіксом. Ця вибірка розбивається на дві частини випадковим чином у співвідношенні 80% до 20%. Перша частина слугує для навчання, а друга – для перевірки.

Бажано використовувати великі літературні тексти, щоб забезпечити репрезентативність і різноманіття вибірки. Класифікатор може бути навчений на близько декілька тисяч слів, що є достатньою кількістю для правильної роботи, але точність може зростати з більшою кількістю слів.

3.4 Проєктування інтерфейсу програмного забезпечення для генерації текстової інформації

Проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для генерації текстової інформації з урахуванням вимог технічного завдання.

Згідно технічного завдання програмне забезпечення для генерації текстової інформації працює у двох режимах: консольному та візуальному.

Приклад інтерфейсу при консольному режимі роботи програми (процес генерації вибірки) наведено на рис. 3.5.

```
(.env) MacBook-Pro-Nikita:diploma quebane$ python -m utils.word_for_training
Please, enter path to file with words
/Users/quebane/Documents/diploma/warandpeace2.txt
Please, enter path to file to save words
/Users/quebane/PycharmProjects/diploma/utils/data/dictionary_classifier3.json
Propositions for познакомились
познакомился 64646

0 of 500

Propositions for достало
недостало 64644
достать

Propositions for приказывал
приказываю 64315
приказывать 58592 аю, аешь
```

Рисунок 3.5 – Приклад інтерфейсу при консольному режимі роботи програмного забезпечення для генерації текстової інформації

Приклад інтерфейсу при консольному режимі роботи програми (процес отримання словника) наведено на рис. 3.6.

```
10-30 13:03:25 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://povto.ru/russkie/slovari/tolkovie/ozhegova/tolkovii-slovar-ozhegova-bukva-sh.htm>
{'accent': 5,
 'id': 2568,
 'possible_endings': ['еню', 'елишь', 'елишь', 'леный', 'ен', 'еня'],
 'word': 'шевелить'}
10-30 13:03:25 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://povto.ru/russkie/slovari/tolkovie/ozhegova/tolkovii-slovar-ozhegova-bukva-sh.htm>
{'accent': 5,
 'id': 2569,
 'possible_endings': ['енось', 'илишься', 'ели'],
 'word': 'шевелиться'}
10-30 13:03:25 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://povto.ru/russkie/slovari/tolkovie/ozhegova/tolkovii-slovar-ozhegova-bukva-sh.htm>
{'accent': 5, 'id': 2570, 'possible_endings': ['у'], 'word': 'шевелю'}
10-30 13:03:25 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://povto.ru/russkie/slovari/tolkovie/ozhegova/tolkovii-slovar-ozhegova-bukva-sh.htm>
{'accent': 4, 'id': 2571, 'possible_endings': ['а'], 'word': 'шевелю'}
```

Рисунок 3.6 – Приклад інтерфейсу при консольному режимі роботи програми (процес отримання словника)

Приклад інтерфейсу роботи з програмою у візуальному режимі при генерації тексту наведено на рис. 3.7.

Poetry Generator Settings

Poetry Type: Random Count of Couplets: 4

Row Settings

Count of Row 1: 4
 Count of Row 2: 4
 Count of Row 3: 7
 Count of Row 4: 4

Generate

Result:

Вкусить темная блаженство все жизнь,
 Надежды свои как под ним,
 Страдать как жизнь все нем,
 Разлуки, свиданья, вольности

Искал как под ним последним,
 Погибнешь нигде вторично как все,
 Глотая руки сжав,
 Гром эскадра пальба

Рисунок 3.7 – Приклад інтерфейсу роботи з програмою у візуальному режимі при генерації тексту

3.5 Висновки за розділом 3

Розроблено програмне забезпечення для розв’язання задачі генерації текстової інформації. Запропоновано структуру та описано основні модулі програмного забезпечення для генерації текстової інформації.

Описано функціонування програмного забезпечення для генерації текстової інформації. Описано особливості реалізації програмного забезпечення для генерації текстової інформації. Виконано проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для генерації текстової інформації.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Програма призначена для розв'язання задачі генерації текстової інформації. Програма написана на мові Python, яка є мовою програмування загального призначення, спрямована на покращення продуктивності розробника і зрозумілість коду та дозволяє ефективно працювати з текстовими даними. В якості середовища розробки для створення програми обрано PyCharm, що являє собою ефективний інструмент для створення програм, написаних на мові програмування Python та включає в себе основні класи для роботи з генеративним штучним інтелектом, потрібні для розробки програмних застосунків генерації текстової інформації.

Програмне забезпечення для розв'язання задачі генерації текстової інформації забезпечує виконання таких функцій:

- можливість генерації текстової інформації, зокрема, у вигляді поетичних форм;
- використання інтелектуальних засобів як базис для створення текстового контенту;
- можливість роботи з програмою як в консольному, так і графічному режимі;
- можливість створення словнику з базового тексту;
- можливість задавання параметрів генерованого тексту (наприклад, кількість рядків генерованої поезії).

Для експлуатації програмного забезпечення для розв'язання задачі генерації текстової інформації необхідно таке апаратне та програмне забезпечення:

- оперативна пам'ять об'ємом 2Гб і більше;
- процесор із тактовою частотою 2,4 GHz і більше;
- вільне місце на диску 500 Мб або більше;

– встановлена операційна система.

Функціональні характеристики розробленого програмного забезпечення для розв’язання задачі генерації текстової інформації наведено у технічному завданні (додаток А). Фрагмент тексту програмного забезпечення для розв’язання задачі генерації текстової інформації наведено у додатку Б.

4.2 Характеристики програми для розв’язання задачі генерації текстової інформації

Програмне забезпечення для розв’язання задачі генерації текстової інформації може працювати як у консольному, так і візуальному режимах. Програмне забезпечення для розв’язання задачі генерації текстової інформації може використовуватись в різних операційних системах, оскільки воно є крос платформним. Для коректного функціонування програми для розв’язання задачі генерації текстової інформації вміст директорії з програмним забезпеченням не можна змінювати.

4.3 Інструкція по експлуатації програми

4.3.1 Звернення до програми

Для запуску програмного забезпечення для розв’язання задачі генерації текстової інформації необхідно запустити головний файл, що знаходиться в кореневій директорії програми.

4.3.2 Вхідні й вихідні дані

Вхідними даними до програмного забезпечення для розв’язання задачі генерації текстової інформації є файл з словником слів, навчальні дані у вигляді пар слів, а також текстовий файл, що містить літературний твір, який відображає взаємозв’язок між цими словами. Крім того, у випадку генерації

текстового контенту у вигляді поезії, система отримує деякі додаткові параметри, наприклад, тип поезії, кількість рядків, кількість слів.

Вихідними даними програмного забезпечення для розв’язання задачі генерації текстової інформації є створений текстовий контент.

4.3.3 Повідомлення

Користувач програмного забезпечення для розв’язання задачі генерації текстової інформації може отримати такі повідомлення: повідомлення про необхідність введення вхідних даних, а також про завершення генерації текстового контенту.

4.4 Виконання програмного забезпечення для розв’язання задачі генерації текстової інформації

До запуску програми користувач повинен визначитися, у якому режимі роботи він хоче використовувати програмне забезпечення для генерації текстової інформації.

Процес генерації вибірки при консольному режимі роботи програми наведено на рис. 4.1.



```
(.env) MacBook-Pro-Nikita:diploma quebane$ python -m utils.word_for_training
Please, enter path to file with words
/Users/quebane/Documents/diploma/warandpeace2.txt
Please, enter path to file to save words
/Users/quebane/PycharmProjects/diploma/Utils/data/dictionary_classifier3.json
Propositions for познакомился
познакомился 64646

0 of 500

Propositions for достало
недостало 64644
достать

Propositions for приказывал
приказывал 64315
приказывать 58592 аю,аешь
```

Рисунок 4.1 – Процес генерації вибірки при консольному режимі роботи програми

Процес отримання словнику даних при роботі у консольному режимі програми наведено на рис. 4.2.

```
10-30 13:03:25 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://povto.ru/russkie/slovari/tolkovie/ozhegova/tolkovii-slovar-ozhegova-bukva-sh.htm>
{'accent': 5,
 'id': 2568,
 'possible_endings': ['еню', 'елишь', 'елишь', 'леный', 'ен', 'еню'],
 'word': 'шевелить'}
10-30 13:03:25 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://povto.ru/russkie/slovari/tolkovie/ozhegova/tolkovii-slovar-ozhegova-bukva-sh.htm>
{'accent': 5,
 'id': 2569,
 'possible_endings': ['енюсь', 'илимся', 'ели'],
 'word': 'шевелиться'}
10-30 13:03:25 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://povto.ru/russkie/slovari/tolkovie/ozhegova/tolkovii-slovar-ozhegova-bukva-sh.htm>
{'accent': 5, 'id': 2570, 'possible_endings': ['и'], 'word': 'шевелю'}
10-30 13:03:25 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://povto.ru/russkie/slovari/tolkovie/ozhegova/tolkovii-slovar-ozhegova-bukva-sh.htm>
{'accent': 4, 'id': 2571, 'possible_endings': ['а'], 'word': 'шевелют'}
```

Рисунок 4.2 – Процес отримання словнику даних при роботі у консольному режимі

У випадку роботи з програмним забезпеченням для розв’язання задачі генерації текстової інформації за допомогою візуального інтерфейсу користувачу необхідно ввести додаткові параметри для генерації тексту, зокрема, тип поезії, кількість рядків, кількість слів (рис. 4.3). Після цього користувач отримує створений текст.

Poetry Generator Settings

Poetry Type Count of Couplets

Row Settings

Count of Row 1

Count of Row 2

Count of Row 3

Count of Row 4

Result:

Вкусить темная блаженство все жизнь,
Надежды свои как под ним,
Страдать как жизнь все нем,
Разлуки, свиданья, вольности

Искал как под ним последним,
Погибнешь нигде вторично как все,
Глотая руки сжав,
Гром эскадра пальба

Рисунок 4.3 – Приклад інтерфейсу роботи з програмою у візуальному режимі при генерації тексту

4.5 Тестування програмного забезпечення для генерації текстової інформації

Виконано тестування програмного забезпечення для генерації текстової інформації.

Виявлено, що програма для генерації текстової інформації функціонує правильно та злагоджено. Вона реалізує всі функціональні вимоги та успішно виконує свою основну задачу – розв’язання задачі генерації текстової інформації.

Розроблене програмне забезпечення дозволяє розв’язувати завдання генерації текстової інформації.

4.6 Висновки за розділом 4

Описано програмне забезпечення для розв’язання задачі генерації текстової інформації. Виконано тестування розробленого програмного забезпечення для розв’язання задачі генерації текстової інформації. Результати тестування програмного забезпечення показали, що розроблена програма дозволяє розв’язувати задачу генерації текстової інформації.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи магістра було проаналізовано та досліджено процес розробки програмного забезпечення для генерування текстів.

Визначено, що у наш час широко використовуються програмні засоби для генерації текстової інформації, що допомагають створювати різні тексти, взаємодіяти з користувачами, відповідати на різні запитання та допомагати у вирішенні різних проблем. Показано, що генерація тексту – це метод, яким штучний інтелект створює текст, що наслідує природне спілкування людини. Відзначено, що значення генерації тексту зростає в різних областях, таких як обробка природної мови, створення контенту, обслуговування клієнтів і розробка програмного забезпечення. Показано, що в базисі створення тексту лежать мовні моделі. Проаналізовано практичні застосування генерації тексту за допомогою засобів генеративного штучного інтелекту. Визначено переваги та обмеження програмного забезпечення створення текстів. Проаналізовано процес розробки програмного забезпечення для створення текстів на основі генеративних моделей. Відзначено, що генеративний інтелект штучний інтелект представляє собою важливу галузь штучного інтелекту та машинного навчання, яка використовується для створення програмного забезпечення, здатного генерувати штучний контент, наприклад, текстові матеріали, зображення, аудіо та відео. Проаналізовано техніки генеративного штучного інтелекту для синтезу генеративних мовних моделей.

За результатами проведеного аналізу зроблено висновок, що у наш час існує досить багато програмних засобів для генерації текстової інформації. Проте деякі програми для генерації тексту можуть створювати стереотипні або загальні висловлювання, що робить їх менш корисними для унікальних або специфічних завдань, деякі програми можуть іноді генерувати неправильну або недостовірну інформацію, що може призвести до поширення некоректної інформації. Інші подібні програми для досягнення оптимальних результатів та

уникнення неправильних висновків програми часто потребують постійного нагляду та втручання від фахівців для покращення генеративної моделі. Тому актуальною є розробка програмного забезпечення для генерації текстової інформації.

Сформульовано функціональні вимоги до програмного забезпечення для генерації текстової інформації.

Для реалізації програмного забезпечення для розв'язання завдання генерації текстової інформації обрано мову програмування Python, яка є мовою програмування загального призначення, спрямована на покращення продуктивності розробника і зрозумілість коду та дозволяє ефективно працювати з текстовими даними.

Для створення програмного забезпечення для розв'язання задачі генерації текстової інформації обрано середовище розробки PyCharm, що являє собою ефективний інструмент для створення програм, написаних на мові програмування Python та включає в себе основні класи для роботи з генеративним штучним інтелектом, потрібні для розробки програмних застосунків генерації текстової інформації.

Новизна роботи полягає в тому, що створено модель взаємодії користувача з програмним забезпеченням для генерації текстової інформації, яка подана у вигляді діаграми прецедентів, та дозволяє розроблювати програмне забезпечення для генерації текстової інформації.

Практичне значення роботи полягає в тому, що розроблено програмне забезпечення для генерації текстової інформації.

Запропоновано структуру та описано основні модулі програмного забезпечення для генерації текстової інформації.

Описано функціонування програмного забезпечення для генерації текстової інформації. Описано особливості реалізації програмного забезпечення для генерації текстової інформації. Виконано проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для генерації текстової інформації.

Виконано тестування розробленого програмного забезпечення для розв'язання задачі генерації текстової інформації. Результати тестування програмного забезпечення показали, що розроблена програма дозволяє розв'язувати задачу генерації текстової інформації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. AI Text Generation: Top 12 Use Cases & 2 Case Studies in 2023 [Electronic resource]. – Access mode : <https://research.aimultiple.com/ai-text-generation/>.
2. Practical text generation using GPT-2, LSTM and Markov Chain [Electronic resource]. – Access mode : <https://towardsdatascience.com/text-generation-gpt-2-lstm-markov-chain-9ea371820e1e>.
3. What is Text Generation? [Electronic resource]. – Access mode : <https://www.danycamp.com/blog/what-is-text-generation>.
4. Generative AI: 7 Steps to Enterprise GenAI Growth in 2023 [Electronic resource]. – Access mode : <https://research.aimultiple.com/generative-ai/>.
5. Text Generation with Diffusion Language Models: A Pre-navching Approach with Continuous Abzс Denoise [Electronic resource]. – Access mode : <https://arxiv.org/pdf/2212.11685.pdf>.
6. Text Generation [Electronic resource]. – Access mode : <https://h2o.ai/wiki/text-generation/>.
7. 12 Bst AI Text Generator Software in 2023 [Electronic resource]. – Access mode : <https://businessolution.org/ai-text-generator>.
8. The top AI text generators in 2023 [Electronic resource]. – Access mode : <https://zapier.com/blog/ai-text-generator/>.
9. 7 Bst AI Text Generator Tools & Software To Use In 2023 [Electronic resource]. – Access mode: <https://digitalmarketingtipsy.com/tools/ai-text-generator/>.
10. Free AI Writing & Text Generation Tools [Electronic resource]. – Access mode : <https://www.grammarly.com/ai-writing-tools>.
11. Jasper [Electronic resource]. – Access mode : <https://www.jasper.ai/>.
12. Writesonic [Electronic resource]. – Access mode : <https://writesonic.com/>.
13. Anyslv [Electronic resource]. – Access mode : <https://anyslv.com/?fpr=digitalmarketingtipsy>.

14. Programiz [Electronic resource]. – Access mode :
<https://www.programiz.com/python-programming/online-compiler/>.
15. Using the Python Interpreter [Electronic resource]. – Access mode :
<https://docs.python.org/3/tutorial/interpreter.html>.
16. Explore Plugins for JetBrains Products [Electronic resource]. – Access mode :
<https://plugins.jetbrains.com/pycharm>.
17. PyCharm Community Edition [Electronic resource]. – Access mode :
<https://pycharm-community-edition.fr.softonic.com/>.
18. Machine Learning [Electronic resource]. – Access mode :
<https://es.mathworks.com/discovery/machine-learning.html>.
19. O que é Machine Learning [Electronic resource]. – Access mode :
<https://www.alura.com.br/artigos/machine-learning>.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення може використовуватися для розв'язання задачі генерації текстової інформації.

А.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Дослідження та програмна реалізація методів генерації текстової інформації», затверджене наказом Національного університету «Запорізька політехніка» № 437 від 14 листопада 2023 р.

А.2 Призначення розробки

Програмний продукт призначений для розв'язання задачі генерації текстової інформації.

А.3 Основні вимоги до програми, що розробляється

А.3.1 Вимоги до функціональних характеристик

Програмне забезпечення для розв'язання задачі генерації текстової інформації забезпечує виконання таких функцій:

- можливість генерації текстової інформації, зокрема, у вигляді поетичних форм;
- використання інтелектуальних засобів як базис для створення текстового контенту;
- можливість роботи з програмою як в консольному, так і графічному режимі;
- можливість створення словнику з базового тексту;

– можливість задавання параметрів генерованого тексту (наприклад, кількість рядків генерованої поезії).

A.3.2 Вимоги до інтерфейсу програми

Інтерфейс програмного забезпечення для генерації текстової інформації повинен бути зручним для користувачів. Повинна бути забезпечена можливість роботи як в консольному режимі, так і в візуальному режимі.

A.3.3 Вимоги до надійності

Програмне забезпечення для генерації текстової інформації повинно забезпечити надійне функціонування.

A.3.4 Умови експлуатації

Для експлуатації програмного забезпечення для розв'язання задачі генерації текстової інформації необхідна наявність персонального комп'ютера у користувача.

A.3.5 Вимоги до складу та параметрів технічних засобів

Для експлуатації програмного забезпечення для розв'язання задачі генерації текстової інформації необхідно таке апаратне та програмне забезпечення:

- оперативна пам'ять об'ємом 2Гб і більше;
- процесор із тактовою частотою 2,4 GHz і більше;
- вільне місце на диску 500 Мб або більше;

– встановлена операційна система.

A.3.6 Вимоги до маркування і пакування

Програма для розв’язання задачі генерації текстової інформації може бути записана на будь-якому носії інформації.

На пакуванні повинна бути назва програми – «Програмне забезпечення для генерації текстової інформації».

A.4 Вхідні дані до роботи

Вхідними даними до програмного забезпечення для розв’язання задачі генерації текстової інформації є файл з словником слів, навчальні дані у вигляді пар слів, а також текстовий файл, що містить літературний твір, який відображає взаємозв’язок між цими словами. Крім того, у випадку генерації текстового контенту у вигляді поезії, система отримує деякі додаткові параметри, наприклад, тип поезії, кількість рядків, кількість слів.

Вихідними даними програмного забезпечення для розв’язання задачі генерації текстової інформації є створений текстовий контент.

ДОДАТОК Б
Фрагмент тексту програми

```

import re
import time

from analyzer.dictnry_generat import generat
from analyzer.dictnry_generat.slovoman import SlvRukvd
from dopomdop import db
from dopomdop.classes import Slv
from dopomdop.fncts import is_ukr_slv

PNCTS = ['...', '.', '?', '!']
NORMLZ_SLV_REGEX = re.compile('[^\w\-\-]')

MRKFR_SLV = 10
MRKFR_FRONT_SLV = 8
MRKFR_PREDLZH_SLV = 4
MRKFR_ABZC = 1
MIN_GENERT_CHNC = 30

KOEFKOR = dict()

SLV_FRINS = []

def gtabzcs(text):
    return text.split('\n\n')

def spltb_pnct(predlzhs, pnct):
    new_predlzhs = []
    for predlzh in predlzhs:
        new_predlzhs.extend(predlzh.split(pnct))
    return new_predlzhs

def gtpredlzhs(abzc):
    predlzhs = [abzc]
    for pnct in PNCTS:
        predlzhs = spltb_pnct(predlzhs, pnct)
    return predlzhs

def gtslvs_in_predlzh(predlzh):
    slvs = predlzh.split(' ')
    slvs = map(lambda x: NORMLZ_SLV_REGEX.sub('', x).lower(),
slvs)
    return list(fltr(lambda x: is_ukr_slv(x), slvs))

```

```

def gtdictnry_slv(slv):
    dictnry_slv = SlvRukvd.gtslv_by_slv(slv)
    if dictnry_slv:
        return dictnry_slv
    slv = Slv(slv=slv)
    parent = generat.gener(slv)
    if parent and generat.gtchnc(slv, parent) > MIN_GENERT_CHNC:
        slv.prnt = parent.gtgroup_id()
    slv.id = SlvRukvd.gtnext_id()
    SLV_FRINS.append(slv)
    print('ADDED SLV {0}'.format(slv.slv))
    SlvRukvd.dobslv(slv)
    return slv

def vstvslvs():
    dbcnctc = db.cnctto_db()
    for slv in SLV_FRINS:
        vstvslv(slv, dbcnctc)
    dbcnctc.close()

def vstvslv(slv, dbcnctc):
    crsr = dbcnctc.crsr()
    vstvcmnd = db.vstvvnew_dictnrycmnd(slv.id, slv.slv, 'null',

slv.prnt or 'null',

[], 'true')
    try:
        crsr.execute(vstvcmnd)
    except psycopg2.IntegrityError:
        dbcnctc.rollback()
    else:
        dbcnctc.commit()
    crsr.close()

def vstvkoeffkor(key, value, dbcnctc):
    crsr = dbcnctc.crsr()
    vstvcmnd = db.vstvkoeffkorcmnd(key[0], key[1], value)
    try:
        crsr.execute(vstvcmnd)
    except psycopg2.IntegrityError:
        dbcnctc.rollback()
    else:
        dbcnctc.commit()
    crsr.close()

```

```

def vstvkoefkors():
    dbcnctc = db.cnctto_db()
    for key in KOEFKOR.keys():
        vstvkoefkor(key, KOEFKOR[key], dbcnctc)
    dbcnctc.close()

def dobkoefkor(slv_a, slv_b, value):
    if (slv_a.id, slv_b.id) not in KOEFKOR:
        KOEFKOR[(slv_a.id, slv_b.id)] = 0
    KOEFKOR[(slv_a.id, slv_b.id)] += value

def dobpredlzh_koefkor(slvs_in_predlzh):
    for i, slv_a in enumerate(slvs_in_predlzh):
        for j, slv_b in enumerate(slvs_in_predlzh):
            if i == j:
                continue
            if i == j + 1:
                dobkoefkor(slv_a, slv_b, MRKFR_SLV)
            elif i + 1 == j:
                dobkoefkor(slv_a, slv_b, MRKFR_FRONT_SLV)
            else:
                dobkoefkor(slv_a, slv_b, MRKFR_PREDLZH_SLV)

def dobabzc_koefkor(slvs_in_predlzhs):
    slv_ids = set()
    for slvs_in_predlzh in slvs_in_predlzhs:
        for slv in slvs_in_predlzh:
            slv_ids.add(slv.id)
    for slv_a_id in slv_ids:
        for slv_b_id in slv_ids:
            if slv_a_id == slv_b_id:
                continue
            dobkoefkor(SlvRukvd.gtslv_by_id(slv_a_id),
SlvRukvd.gtslv_by_id(slv_b_id),
                        MRKFR_ABZC)

def crltabz(abzc):
    predlzhs = gtpredlzhs(abzc)
    print('{0} predlzhs'.format(len(predlzhs)))
    slvs_in_predlzhs = []
    for i, predlzh in enumerate(predlzhs):
        slvs_in_predlzh = gtslvs_in_predlzh(predlzh)
        print('{0} slvs at {1}'.format(len(slvs_in_predlzh), i))

```

```

        slvs_in_predlzh = list(map(lambda x: gtdictnry_slv(x),
slvs_in_predlzh))
        dobpredlzh_koefkor(slvs_in_predlzh)
        slvs_in_predlzhs.append(slvs_in_predlzh)
        dobabzc_koefkor(slvs_in_predlzhs)

def crcfc():
    generat.init_vertties()
    pthtofl = input('Введіть шлях до тексту\n')
    with open(pthtofl) as file:
        txt = file.read()
        abzcs = gtabzcs(txt)
        klk_of_abzcs = len(abzcs)
        nachn_time = time.time()
        try:
            for i, abzc in enumerate(abzcs):
                print('{0} of {1} abzcs'.format(i, klk_of_abzcs))
                crltabzc(abzc)
                crnttm = time.time()
                esttm = ((crnttm - nachn_time) / (i + 1)) *
(klk_of_abzcs - i)
                print('Поточний час: {0} s. Час очікування {1}
s'.format((crnttm - nachn_time), esttm))
            except Exception as ex:
                print(ex)
                print('Помилка' * 1000)
        vstvsivs()
        vstvkoefkors()

class KoefkorRukvd:
    koefkors = None
    koefkor_by_slv_id = dict()

    @classmethod
    def init_koefkor(cls):
        cls.koefkors = db.vybotrall_koefkors()
        cls.koefkors = list(fltr(lambda x: x.slv_a_id !=
x.slv_b_id, cls.koefkors))
        for koefkor in cls.koefkors:
            if not cls.koefkor_by_slv_id.get(koefkor.slv_a_id):
                cls.koefkor_by_slv_id[koefkor.slv_a_id] = []

        cls.koefkor_by_slv_id[koefkor.slv_a_id].append(koefkor)

    @classmethod
    def gtmxslv_koefkor(cls, slv_a, klk_of_bst_koefkor,
ispslvs=None, min_length=0,
nusslvs=None):

```

```

mxkoefkors = []
if not nusslvs:
    nusslvs = []
nusslvs = list(map(lambda x: x.id, nusslvs))
if ispslvs:
    ispslvs = list(map(lambda x: x.id, ispslvs))
for koefkor in cls.koefkor_by_slv_id.get(slv_a.id, []):
    slv_b = SlvRukvd.gtslv_by_id(koefkor.slv_b_id)
    if len(slv_b.slv) < min_length or slv_b.id in
nusslvs:
        continue
    if ispslvs and slv_b.id not in ispslvs:
        continue
    if len(mxkoefkors) < klk_of_bst_koefkor:
        mxkoefkors.append(koefkor)
    elif koefkor.value > mxkoefkors[0].value:
        mxkoefkors[0] = koefkor
    mxkoefkors.sort(key=lambda x: x.value)
if not mxkoefkors:
    return None
return random.choice(mxkoefkors)

@classmethod
def gtrandom_koefkor(cls):
    return random.choice(cls.koefkors)

def gtchnc(slv, dictnry_slv):
    hsprfx = chckprfx(slv, dictnry_slv)
    hsending = chckending(slv, dictnry_slv)
    smlttr = slv.percent_of_smlttr(dictnry_slv)
    vertty = Vertties.gtvertty(hsprfx, smlttr, hsending)
    chnc_of_vertty = Vertties.gtchnc_of_vertty(hsprfx, smlttr,
hsending)
    return vertty

def gener(slv):
    mxchnc = 0
    mxslv = None
    if SlvRukvd.gtslv_by_slv(slv.slv):
        return SlvRukvd.gtslv_by_slv(slv.slv)
    for dictnry_slv in SlvRukvd.gtslvs():
        chnc = gtchnc(slv, dictnry_slv)
        if chnc > mxchnc:
            mxchnc = chnc
            mxslv = dictnry_slv
    return mxslv

```

```

def gtvertties():
    vertty_path = input('Введіть шлях\n')
    with open(vertty_path) as verttyfl:
        vertties = dsrlz_vertties_ffl(verttyfl)
        return vertties

def gttstnavching_stnctes():
    tstnavching_stnctes_path = input('Введіть шлях \n')
    with open(tstnavching_stnctes_path) as tstnavchingfl:
        tstnavching_stnctes = dsrlz_json_ffl(tstnavchingfl)
        return tstnavching_stnctes

def set_vertties(vertties):
    for key in list(vertties['vertties'].keys()):
        for klk in range(vertties['klk'][key]):
            dany = json.ldrs(key)
            Vertties.dobhpyt(vertties['vertties'][key], dany[0],
dany[1], dany[2])

def init_vertties():
    vertties = gtvertties()
    set_vertties(vertties)

def tstgenert():
    init_vertties()
    tstnavching_stnctes = gttstnavching_stnctes()
    klk_of_positive = 0
    for i, tstnavching_stncte in enumerate(tstnavching_stnctes):
        navching_slv = Slv(slv=tstnavching_stncte['slv'])
        result_slv = SlvRukvd.gtslv_by_id(tstnavching_stncte['result'])
        result_genert = gener(navching_slv)
        print(i, 'yes' if result_genert.gtgroup_id() ==
result_slv.gtgroup_id() else 'no',
navching_slv.slv, result_genert.slv,
result_slv.slv)
        if result_genert.gtgroup_id() == result_slv.gtgroup_id():
            klk_of_positive += 1
        else:
            print(gtchnc(navching_slv, result_slv),
gtchnc(navching_slv, result_genert))
            print('_____')
            print('{0} of {1}'.format(klk_of_positive,
len(tstnavching_stnctes)))

```

```

def navch(navching_stnctes):
    for i, navching_stncte in enumerate(navching_stnctes):
        same_hypt_slvs = 1
        navching_slv = Slv(slv=navching_stncte['slv'])
        result_slv =
SlvRukvd.gtslv_by_id(navching_stncte['result'])
        hsprfx = chckprfx(navching_slv, result_slv)
        hsendng = chckending(navching_slv, result_slv)
        smlttr = navching_slv.percent_of_smlttr(result_slv)
        for slv in SlvRukvd.gtslvs():
            if slv.id == result_slv.id or slv.gtgroup_id() ==
result_slv.gtgroup_id():
                continue
            if hsprfx == chckprfx(navching_slv, slv) and \
                hsendng == chckending(navching_slv,
slv) and \
                    smlttr ==
navching_slv.percent_of_smlttr(slv):
                same_hypt_slvs += 1
                Vertties.dobhypt(100.0 / same_hypt_slvs, hsprfx, smlttr,
hsendng)
                print(i)
                Vertties.gtvertties()

def chckprfx(navching_slv, result_slv):
    if not navching_slv.hsprfx():
        return False
    if navching_slv.hsprfx() and not result_slv.hsprfx():
        return True
    if not result_slv.chcksame_prfxes(navching_slv):
        return True
    return False

def chckending(navching_slv, result_slv):
    return result_slv.chckpossible_ending(navching_slv)

def gtnavching_stnctes():
    navching_stnctes_path = input('Введіть шлях\n')
    with open(navching_stnctes_path) as navchingfl:
        navching_stnctes = dsrlz_json_ffl(navchingfl)
        return navching_stnctes

def wrtvertties_tofl(vertties):
    print(vertties)

```

```

def nachn_generat_navching():
    navching_stnctes = gtnavching_stnctes()
    vertties = navch(navching_stnctes)
    wrtvertties_tofl(vertties)

if __name__ == '__main__':
    nachn_generat_navching()

def dsrlz_json_ffl(file):
    dany = file.read()
    return json.ldrs(dany)

def dsrlz_vertties_ffl(file):
    dany = file.read()
    dany = json.ldrs(dany)
    return dany

class SlvRukvd:
    slvs_by_id = dict()
    slvs_by_group = dict()
    slvs = []
    slvs_by_slv = dict()
    mxid = 0

    def init_slvs(cls):
        cls.slvs = db.vybotrall_slvs()
        for slv in cls.slvs:
            cls.init_slv(slv)

    def init_slv(cls, slv):
        cls.slvs_by_id[slv.id] = slv
        cls.chckmxid(slv)
        cls.slvs_by_slv[slv.slv] = slv
        if slv.prnt:
            if not cls.slvs_by_group.get(slv.prnt):
                cls.slvs_by_group[slv.prnt] = []
            cls.slvs_by_group[slv.prnt].append(slv)

    def dobslv(cls, slv):
        cls.slvs.append(slv)
        cls.init_slv(slv)

    def chckmxid(cls, slv):

```

```

        if slv.id > cls.mxid:
            cls.mxid = slv.id

    def gtnext_id(cls):
        return cls.mxid + 1

    def gtslv_by_id(cls, id):
        return cls.slvs_by_id[id]

    def gtslv_by_slv(cls, slv):
        return cls.slvs_by_slv.get(slv)

    def gtslvs(cls):
        return cls.slvs

    def gtgroups_klk(cls):
        return len(cls.slvs_by_group)

SlvRukvd.init_slvs()

class DictnryPrsrPpln(object):
    dbcnctc = None

    def open_Pwk(self, Pwk):
        self.dbcnctc = db.cnctto_db()

    def close_Pwk(self, Pwk):
        self.dbcnctc.close()

    def NORMLZ_Elem(self, Elem):
        Elem['slv'] = Elem['slv'].lower().strip()
        if not Elem['possible_endings']:
            Elem['possible_endings'] = []
        return Elem

    def process_Elem(self, Elem, Pwk):
        Elem = self.NORMLZ_Elem(Elem)
        crsr = self.dbcnctc.crsr()
        vstvcmd = db.vstvdictnrycmd(Elem['id'], Elem['slv'],
Elem.get('accent', 'null'),
Elem.get('prnt', 'null'),
Elem['possible_endings'])
        try:
            crsr.execute(vstvcmd)
        except psycopg2.IntegrityError:
            self.dbcnctc.rollback()

```

```

        else:
            self.dbcnctc.commit()
            crsr.close()
            return Elem

STRGS = [IambTxtStrat, TreedTxtStrat]

STRGS_NAMES = dict(Iamb=IambTxtStrat, Treed=TreedTxtStrat)

def hener_random_txt(strat=None, klk_of_rows=4):
    strat_cls = STRGS_NAMES.get(strat)
    if not strat_cls:
        strat_cls = random.choice(STRGS)
    print('Strat {0}'.format(strat_cls.getstrat_name()))
    strat = strat_cls(SlvRukvd.gtslvs(), klk_of_rows)
    strat.hener()
    print(strat)
    return strat

class AmpgTxtStrat(BaseTxtStrat):
    def fltr_slvs(self, slvs):
        def fltr_slv(slv):
            actnmbr = SlvRukvd.gtactnmbr(slv)
            return actnmbr and actnmbr % 2 != 0

        slvs = list(fltr(fltr_slv, slvs))
        return slvs

class BaseTxtStrat(ABC):
    rows = []

    def __init__(self, slvs, klk_of_rows):
        self.slvs = self.fltr_slvs(slvs)
        self.klk_of_rows = klk_of_rows

    def gtklk_of_slvs_in_row(self):
        return random.choice(SLVS_IN_TXT_KLK)

    def gtfirst_slv(self):
        if len(self.rows) == 0:
            return self.gtrandomfrst_slv()
        lsrw = self.rows[-1][:]
        random.shuffle(lsrw)
        bst_slv = self.hener_slv(lsrw, 500)
        if bst_slv:
            return bst_slv
        return self.gtrandomfrst_slv()

```

```

def gtrandomfrst_slv(self):
    all_slvs = list(map(lambda x: x.id, self.slvs))
    while True:
        slv = self.gtslv()
        if slv.id in all_slvs:
            return slv

def gtslv(self):
    random_koefkor = KoefkorRukvd.gtrandom_koefkor()
    return SlvRukvd.gtslv_by_id(random_koefkor.slv_a_id)

def hener_slv(self, row, klk_of_bst_koefkor):
    mxkoefkor = None
    for slv in reversed(row):
        koefkor = KoefkorRukvd.gtmxslv_koefkor(slv,
        klk_of_bst_koefkor, min_length=3,
        nusslvs=row, ispslvs=self.slvs)
        if koefkor:
            mxkoefkor = koefkor
            break
    if mxkoefkor:
        return SlvRukvd.gtslv_by_id(mxkoefkor.slv_b_id)
    return self.gtslv()

def fltr_slvs(self, slvs):
    return slvs

class DctlTxtStrat(BaseTxtStrat):
    def fltr_slvs(self, slvs):
        def fltr_slv(slv):
            actnmbr = SlvRukvd.gtactnmbr(slv)
            return actnmbr and actnmbr % 2 == 0

        slvs = list(fltr(fltr_slv, slvs))
        return slvs

    @classmethod
    def getstrat_name(cls):
        return 'Dctl'

class IambTxtStrat(BaseTxtStrat):
    def fltr_slvs(self, slvs):
        def fltr_slv(slv):
            actnmbr = SlvRukvd.gtactnmbr(slv)
            return actnmbr and actnmbr % 2 != 0

        slvs = list(fltr(fltr_slv, slvs))

```

```
        return slvs

    @classmethod
    def getstrat_name(cls):
        return 'Iamb'
```

ДОДАТОК В
Слайди презентації

Міністерство освіти та науки України
Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Дослідження та програмна реалізація методів генерації текстової інформації

Арсен ШАПОВАЛ,

ст. групи КНТ-112м

керівник Євгеній ГОФМАН,

к.т.н., доцент кафедри ПЗ

Рисунок В.1 – Слайд 1

Об'єкт, предмет та мета роботи

- **Об'єкт дослідження** – процес розробки програмного забезпечення для генерування текстів.
- **Предмет дослідження** – методи та програмні засоби для розв'язання завдань створення текстової інформації.
- **Метою роботи є** дослідження та розробка програмного забезпечення для розв'язання завдань створення текстової інформації.

2

Рисунок В.2 – Слайд 2

Завдання роботи

Для досягнення поставленої мети у кваліфікаційній роботі бакалавра необхідно розв'язати такі задачі:

- виконати аналіз предметної області та програмних засобів для генерації текстової інформації;
- розробити модель взаємодії користувача з програмним забезпеченням для генерації текстової інформації;
- здійснити проєктування програмного забезпечення для генерації текстової інформації;
- створити програмне забезпечення для генерації текстової інформації;
- виконати тестування розробленого програмного забезпечення для генерації текстової інформації.

3

Рисунок В.3 – Слайд 3

Порівняння існуючих аналогів

Критерій порівняння	Jasper	Writesonic	Anyslv
Зручність роботи з програмою	+–	+	+
Засоби оцінювання продуктивності генерації тексту	–	–	+
Генеративні моделі для створення тексту	GPT-3	GPT-3, GPT-3.5, GPT-4	GPT-3, T5, CTRL
Кількість інтелектуальних засобів для підтримки роботи користувача	50	90	100
Інтеграція з граматичними засобами	+	–	+

4

Рисунок В.4 – Слайд 4

Порівняння мов програмування

Критерій порівняння мов програмування	Мова програмування		
	Python	Matlab	C++
Зручність розробки програмного забезпечення для генерації текстової інформації	+	–	–
Робота з текстовими даними	+	+–	+
Засоби спадкування	+	+–	+
Засоби генеративного штучного інтелекту	+	+–	+–
Наявність бібліотек для машинного навчання	+	+	+–

5

Рисунок В.5 – Слайд 5

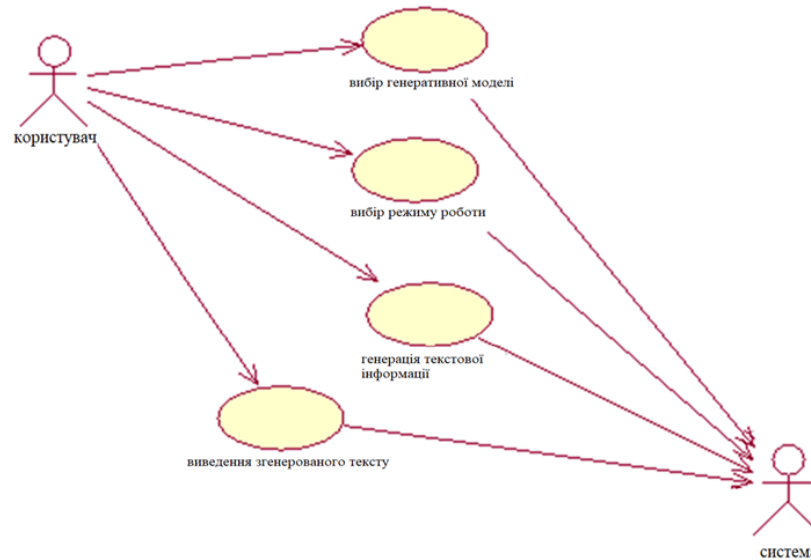
Обґрунтування вибору середовища розробка

Критерій порівняння середовищ розробки	Середовище розробки		
	PyCharm	Visual Studio	Sublime
Зручність розробки на мові Python	+	+–	+
Засоби для аналізу коду	+	+	+–
Засоби візуального налагодження	+	+	–
Зручність для створення ПЗ для генерації текстової інформації	+	–	–
Проста навігація у проєкті та вихідному коді	+	–	–

6

Рисунок В.6 – Слайд 6

Модель взаємодії користувача з програмним забезпеченням для генерації текстової інформації



7

Рисунок В.7 – Слайд 7

Структура програмного забезпечення для генерації текстової інформації



8

Рисунок В.8 – Слайд 8

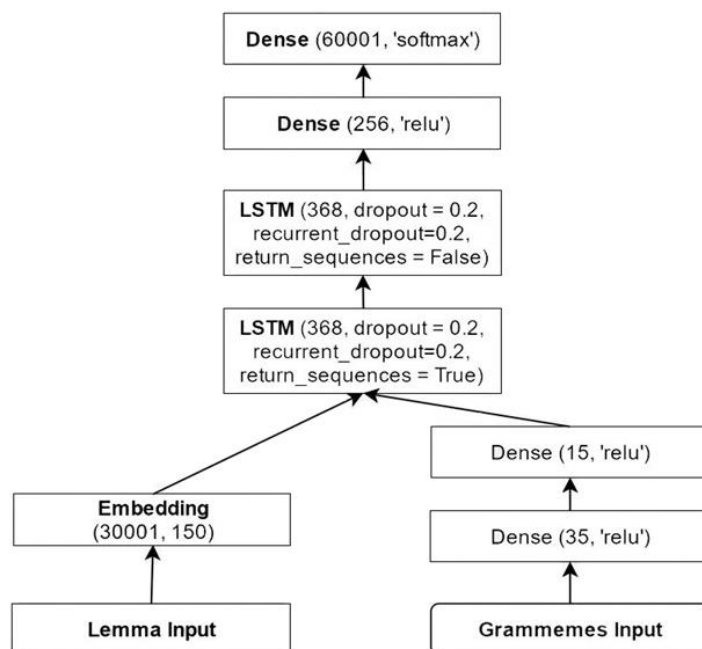
Схема функціонування програми



9

Рисунок В.9 – Слайд 9

Модель генерації текстової інформації



10

Рисунок В.10 – Слайд 10

Интерфейс застосунку. Консольний режим

```
(.env) MacBook-Pro-Nikita:diploma quebane$ python -m utils.word_for_training
Please, enter path to file with words
/Users/quebane/Documents/diploma/warandpeace2.txt
Please, enter path to file to save words
/Users/quebane/PycharmProjects/diploma/utils/data/dictionary_classifier3.json
Propositions for познакомились
познакомился 64646

0 of 500

Propositions for достало
недостало 64644
достать

Propositions for приказывал
приказывая 64315
приказывать 58592 аю,аешь

10-30 13:03:25 [scrapy.core.scrapy] DEBUG: Scraped from <200 http://povto.ru/russkie/slovani/tolkovie/azhegova/tolkovi-i-slovar-azhegova-bukva-sh.htm>
{'accent': 's',
 'id': 2568,
 'possible_endings': ['еня', 'ення', 'ення', 'ення', 'ен', 'ен'],
 'word': 'вееення'}
10-30 13:03:25 [scrapy.core.scrapy] DEBUG: Scraped from <200 http://povto.ru/russkie/slovani/tolkovie/azhegova/tolkovi-i-slovar-azhegova-bukva-sh.htm>
{'accent': 's',
 'id': 2569,
 'possible_endings': ['еня', 'ення', 'ення', 'ення', 'ен', 'ен'],
 'word': 'вееення'}
10-30 13:03:25 [scrapy.core.scrapy] DEBUG: Scraped from <200 http://povto.ru/russkie/slovani/tolkovie/azhegova/tolkovi-i-slovar-azhegova-bukva-sh.htm>
{'accent': 's', 'id': 2570, 'possible_endings': ['а'], 'word': 'веееня'}
10-30 13:03:25 [scrapy.core.scrapy] DEBUG: Scraped from <200 http://povto.ru/russkie/slovani/tolkovie/azhegova/tolkovi-i-slovar-azhegova-bukva-sh.htm>
{'accent': 'a', 'id': 2571, 'possible_endings': ['а'], 'word': 'веееня'}
```

11

Рисунок В.11 – Слайд 11

Интерфейс застосунку. Візуальний режим

Poetry Generator Settings

Poetry Type Count of Couplets

Row Settings

Count of Row 1

Count of Row 2

Count of Row 3

Count of Row 4

Result:

Вкусить темная блаженство все жизнь,
Надежды свои как под ним,
Страдать как жизнь все нем,
Разлуки, свидания, вольности

Искать как под ним последним,
Погнись нигде вторично как все,
Глотая руки скаж,
Гром эскадра пальба

12

Рисунок В.12 – Слайд 12

Висновки

В ході виконання дипломної кваліфікаційної роботи магістра було проаналізовано та досліджено процес розробки програмного забезпечення для генерування текстів.

Для створення програмного забезпечення для розв'язання завдання генерації текстової інформації обрано мову програмування Python, яка є мовою програмування загального призначення, спрямована на покращення продуктивності розробника і зрозумілість коду та дозволяє ефективно працювати з текстовими даними.

Для створення програмного забезпечення для розв'язання задачі генерації текстової інформації обрано середовище розробки PyCharm, що являє собою ефективний інструмент для створення програм, написаних на мові програмування Python та включає в себе основні класи для роботи з генеративним штучним інтелектом, потрібні для розробки програмних застосунків генерації текстової інформації.

13

Рисунок В.13 – Слайд 13

Висновки

Новизна роботи полягає в тому, що створено модель взаємодії користувача з програмним забезпеченням для генерації текстової інформації, яка подана у вигляді діаграми прецедентів у нотації UML, та дозволяє розроблявати програмне забезпечення для генерації текстової інформації.

Практичне значення роботи полягає в тому, що розроблено програмне забезпечення для генерації текстової інформації.

Описано функціонування програмного забезпечення для генерації текстової інформації. Описано особливості реалізації програмного забезпечення для генерації текстової інформації. Виконано проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для генерації текстової інформації.

Усі завдання випускної дипломної роботи магістра повністю виконано.

14

Рисунок В.14 – Слайд 14