Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
магістр
(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ ПОБУДОВИ ЧАТ-БОТІВ З ДИНАМІЧНОЮ ОБЛАСТЮ ВИКОРИСТАННЯ RESEARCH AND SOFTWARE IMPLEMENTATION OF METHODS FOR CREATING CHAT-BOTS WITH A DYNAMIC FIELD OF APPLICATION

Виконав: студент(ка) 2 курсу, групи КНТ-129м
Спеціальності 121 Інженерія програмного забезпечення
(код і найменування спеціальності)
Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Тарасенко Д.О.
(прізвище та ініціали)
Керівник Колпакова Т.О.
(прізвище та ініціали)
Рецензент Казурова А.Є.
(прізвище та ініціали)

2020

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІІРЕ, ФКНТ
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.
_____С.О. Субботін
"____" _____20___року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Тарасенка Дмитра Олександровича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів побудови чат-ботів з динамічною областю використання. Research and Software Implementation of Methods For Creating Chat-bots with a Dynamic Field of Application.

керівник проєкту (роботи) Колпакова Тетяна Олексіївна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від "03" листопада 2020 року № 301
2. Строк подання студентом проєкту (роботи) 5 грудня 2020 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Дослідження проблеми розробки контексту чат-бота за допомогою методів машинного навчання. 2. Дослідження засобів генерації відповідей. 3. Проект програмної системи контексту нейронної мережі та чат-бота. 4. Реалізація програмної системи контексту нейронної мережі та чат-бота. 5. Охорона праці та безпека у надзвичайних ситуаціях. 6. Економічне обгрунтування дипломної роботи. 7. Додатки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Колпакова Т.О., доцент		
Організаціно-економічна частина	Пожуєва Т.О., професор		
Охорона праці та цивільна безпека	Коробко О.В., ст. викладач		
Нормоконтролер	Дейнега Л.Ю., ст. викладач		

7. Дата видачі завдання “ 4 ” вересня 2020 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	Дослідження проблеми розробки контексту чат-бота за допомогою методів машинного навчання.	2–3 тижні	Розділ 1
3	Дослідження засобів генерації відповідей.	4–5 тижні	Розділ 2
4	Проект програмної системи контексту нейронної мережі та чат-бота.	6 тиждень	Розділ 3
5	Реалізація програмної системи контексту нейронної мережі та чат-бота.	7–8 тижні	Розділ 4
6	Охорона праці та безпека у надзвичайних ситуаціях	10 тиждень	Розділ 5
7	Економічне обгрунтування дипломної роботи	11 тиждень	Розділ 6
8	Оформлення пояснювальної записки та документів до неї.	12-13 тиждень	Додатки
9	Нормоконтроль та рецензування.	14–15 тижні	
10	Захист роботи.	16 тиждень	

Студент(ка) _____ Тарасенко Д.О.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Колпакова Т.О.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 147 с., 9 табл., 42 рис., 2 дод., 61 джерел.

МАШИННЕ НАВЧАННЯ, ГЛИБИННЕ НАВЧАННЯ, НЕЙРОННА МЕРЕЖА, ЧАТ-БОТ, ФРЕЙМВОРК, КОНТЕКСТ, PYTHON.

Об’єкт дослідження – електронні системи для створення чат-ботів та аналізу вхідних даних.

Предмет дослідження – можливість використання технології машинного навчання при побудові чат-ботів з динамічною областю використання.

Мета роботи – створення програмної системи контексту нейронної мережі у чат-ботах.

Матеріали, методи та технічні засоби: структурне та об’єктно-орієнтоване програмування, фреймворк Keras, мова програмування Python та бібліотека python-telegram-bot для розробки чат-ботів.

Результати. Створено застосунок чат-боту з використанням контексту на базі існуючих даних.

Висновки. Розроблено програмну систему контексту нейронної мережі та чат-бота з використанням алгоритмів обробки природньої людської мови.

Галузь використання – взаємодія бізнесу з клієнтами через месенджери за допомогою чат-ботів направлених на аналіз повідомлень та коротких текстів, що мають жорстку потребу у контексті.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 147 pages, 42 figures, 9 tables, 2 appendixes, 61 sources.

MACHINE LEARNING, DEEP LEARNING, NEURAL NETWORK, CHAT-BOT, FRAMEWORK, CONTEXT, PYTHON.

Object of study are electronic systems for creating chatbots and analyzing input data.

Subject of study is the possibility of using machine learning technology in the construction of chatbots with a dynamic scope.

The purpose of the work is to create a software system of the neural network context in chatbots.

Materials, methods, and tools: structural and object-oriented programming, Keras framework, Python programming language and python-telegram-bot library for chatbot development.

Results. Created a chatbot application using the context based on existing data.

Conclusions A software system for the context of a neural network and a chatbot has been developed using algorithms for processing natural human speech.

The field of use is a business interaction with customers through messengers using chatbots aimed at analyzing messages and short texts that are in dire need of context.

ЗМІСТ

С.

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CPU	– Central Processing Unit;
DNN	– Deep Neural Network;
GPU	– Graphics Processing Unit;
IDE	– Integrated Development Environment,
IGE	– Infinite Garble Extension;
LSTM	– Long Short-Term Memory;
NLP	– Natural Language Processing;
NLTK	– Natural Language Toolkit;
NMT	– Neural Machine Translation;
PEP	– Python Enhancement Proposal;
RNN	– Recurrent Neural Network;
TPU	– Tensor Processing Unit;
ГП	– Графічний процесор;
ОРС	– Оптичне розпізнавання символів;
ОС	– Операційна система;
ПЗ	– Програмне забезпечення;
ПК	– Персональний комп’ютер;
ТП	– Тензорний процесор;
ЦП	– Центральний процесор;

ВСТУП

Виникнення мережі Інтернет та бурхливе зростання доступної текстової інформації значно прискорило розвиток наукової галузі, яка існує вже багато десятків років і відомої як автоматична обробка текстів (Natural Language Processing) і комп'ютерна лінгвістика (Computational Linguistics). В рамках цієї області запропоновано багато цікавих та перспективних ідей з автоматичної обробки текстів природною людською мовою.

На сьогоднішній день тема розробки чат-ботів є актуальною через те, що вони можуть бути дуже простим інтерфейсом до складних систем. Чат-боти часто використовуються в системах діалогу для різних практичних цілей, включаючи сервіси обслуговування клієнтів або отримання інформації. Деякі чат-боти використовують складні системи обробки природної людської мови, але більшість використовує простіші системи.

Метою і завданням кваліфікаційної роботи магістра є виявлення можливості розробки та застосування контексту нейронної мережі у чат-ботах. Аналізуючи предметну область виявити та представити недоліки існуючих електроних систем для побудови контексту чат-ботів та виявити основні і критичні фактори, на які слід звертати увагу, при побудові подібних систем.

У якості прикладу розробити систему, яка б демонструвала необхідність використання нейронних мереж та системи контексту в чат-ботах.

Предметом дослідження є можливість використання технології машинного навчання при побудові чат-ботів з динамічною областю використання.

Об’єкт дослідження – електронні системи для створення чат-ботів та аналізу вхідних даних.

В роботі було використано наступні методи дослідження:

– аналітичний метод дослідження використовувався для попереднього аналізу предметної області та проблем, які могли з’явитись під час роботи.

Шляхом розкладання об’єкту досліджень на складові частини та аналізу проблем і недоліків кожної, можна зробити висновки про першочергові властивості шуканого результату;

– методи порівняння та оцінювання були використані для виявлення недоліків вже існуючих аналогів та після закінчення розробки системи для виявлення її переваг;

– методи вимірювання та розрахунку були використані для проведення чисельних порівнянь вже існуючих аналогів з поточною системою.

Наукова новизна одержаних результатів полягає в тому, що був створений прототип системи, яка може користуватись наявними можливостями контексту в зв'язці з нейронною мережею. Була визначена актуальність використання цієї системи в загальному використанні чат-ботів для видачі релевантних відповідей користувачеві, на основі діалогу і загального контексту, заданого адміністратором (оператором) чат-бота.

Користувач повинен мати можливість за допомогую інтерфейсу чат-боту додати дані до контексту застосунку, використовучи як прості, так і складні речення. Застосунок повинен виконувати наступні дії: додати дані до контексту, отримати відповідь з використанням даних з контексту на запит.

Практичне значення одержаних результатів полягає в тому, що було проаналізовано проблему жорсткої залежності від чітко визначеної структури вхідних даних моделі рекурентної нейронної мережі, та виконано спробу розширити загальну область використання, у якій може працювати будь-який сервіс на основі рекурентної нейронної мережі, направлений на аналіз повідомлень та коротких текстів, що має жорстку потребу у контексті. Для вирішення цієї проблеми будуть використані алгоритми обробки природньої людської мови [1].

1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ РОЗРОБКИ КОНТЕКСТУ ЧАТ-БОТА ЗА ДОПОМОГОЮ МЕТОДІВ МАШИННОГО НАВЧАННЯ

1.1 Загальні відомості про машинне навчання

Машинне навчання (англ. machine learning) – це підгалузь інформатики (зокрема, м'яких та гранульованих обчислень), яка еволюціонувала з досліджень, які займалися розпізнаванням образів та теорією обчислювального навчання в галузі штучного інтелекту [2]. 1959 року Артур Семюель визначив машинне навчання як «Галузь досліджень, яка дає комп'ютерам здатність навчатися без того, щоби їх явно програмували». Машинне навчання досліджує вивчення та побудову алгоритмів, які можуть навчатися з даних, і виконувати передбачувальний аналіз на них. Такі алгоритми діють шляхом побудови моделі зі зразкового тренувального набору вхідних спостережень, щоби здійснювати керовані даними прогнози або ухвалювати рішення, виражені як виходи, замість того, щоби суворо слідувати статичним програмним інструкціям .

Сучасне машинне навчання має тісний зв’язок та часто перетинається з обчислювальною статистикою. Також воно має тісний зв'язок з математичною оптимізацією, теорією матриць та лінійною алгеброю, що допомогає забезпечити цю галузь методами, теорією та прикладними областями. Машинне навчання застосовують в ряді обчислювальних задач, в яких розробка та програмування явних алгоритмів є нездійсненними. Найчастіші області застосування включають відфільтровування спаму, оптичне розпізнавання символів (ОРС), комп'ютерний зір та різномантітні пошукові системи. Також є можливість поєднання машинного навчання з іншою підгалуззю, яка більше фокусується на дослідницькому аналізі даних, вона є відомою як навчання без учителя.

В області аналізу даних машинне навчання - це метод, який використовується для винайдення складних моделей та алгоритмів, що служать

для прогнозування - у комерційних додатках воно відоме як прогнозована аналітика. Ці аналітичні моделі дозволяють дослідникам, дослідникам даних, інженерам та аналітикам "виробляти надійні, повторювані рішення та результати" та виявляти "приховані уявлення", вивчаючи історичні взаємозв'язки та тенденції в даних.

1.2 Аналіз проблеми контексту діалогу з чат-ботом

В даний час існують такі рішення, як Dialogflow та Amazon Lex, за домогою яких можна створювати чат-боти через веб-інтерфейс, але вони можуть працювати лише в їх вузькому діапазоні і не можуть добре опрацьовувати нові формати даних, якщо це не визначено оператором. Отже, якщо структуру речень або порядок слів буде змінено, тоді сервіс може не генерувати або генерувати неправильні відповіді.

1.3 Огляд існуючих рішень

Dialogflow [3] – це безкоштовний сервіс для розробки чат-ботів для будь-яких платформ, за допомогою відкритого API, що допомогає створити абстракцію та дозволяє використовувати сервіс без прив’язки до однієї платформи. Він надає можливість створювати чат-ботів, які зберігають базову інформацію про контекст діалогу (рис. 1.1), але місця та шаблони речень мають бути заздалегідь задані.

Також сервіс надає кілька шаблонів чат-ботів, наприклад, підтримку базового діалогу, пошук у мережі за допомогою Google, бронювання готелів, тощо.

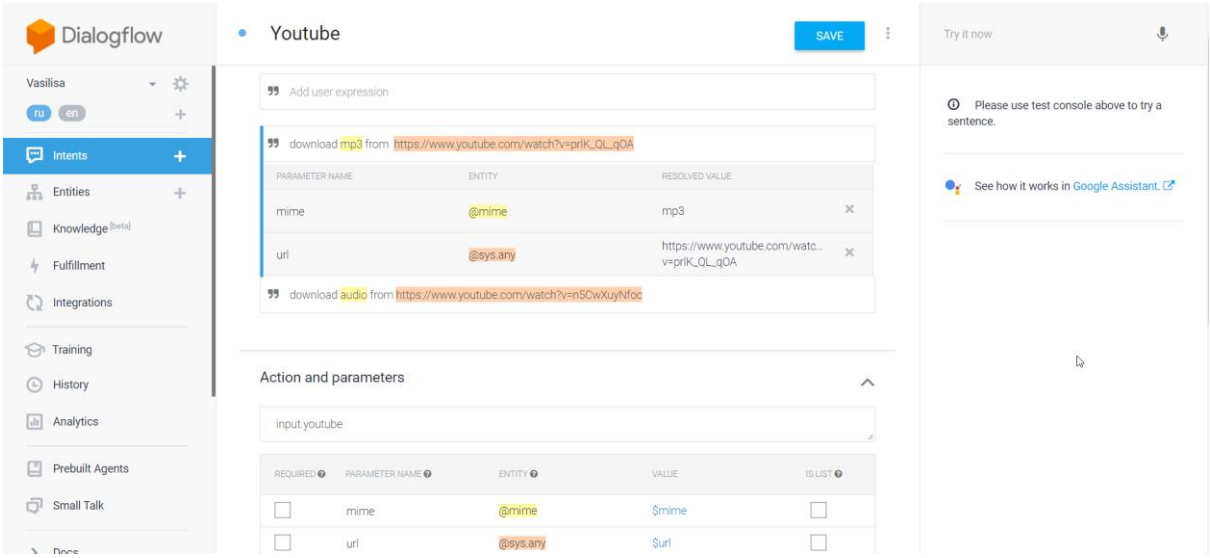

Рисунок 1.1 – Розробка шаблону діалога на сервісі Dialogflow

AllenNlp [4] – це бібліотека з відкритим кодом для створення та оцінки моделей глибокого навчання для вирішення проблем нейролінгвістичного програмування (NLP) та для розробки програм NLP. AllenNLP будується та підтримується Алленським інститутом штучного інтелекту у співпраці з Університетом Вашингтона.

AllenNLP розробив декілька моделей з відкритим кодом для машинного розуміння, текстового залучення, семантичної ролі, маркування роздільної здатності, названого розпізнавання об'єктів, розбору виборчих округів, аналізу залежностей, відкритого вилучення інформації та event2mind (модель виводу між подіями та психічними станами) (рис. 1.2).

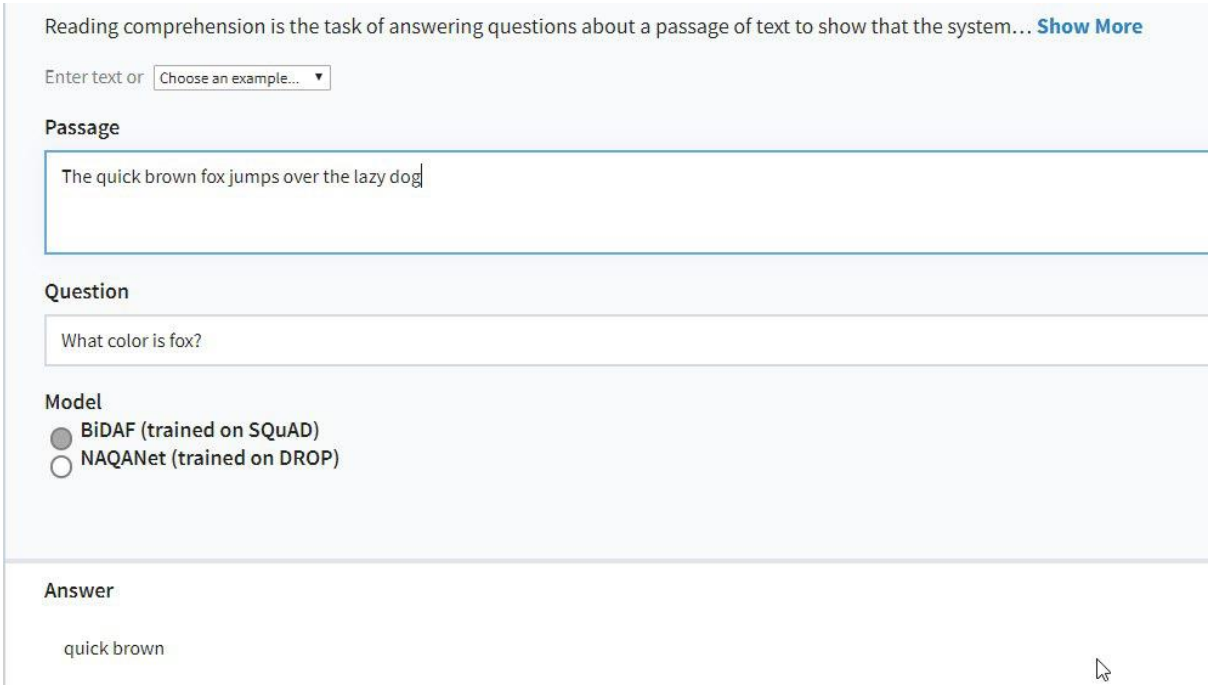

Рисунок 1.2 – Приклад роботи AllenNLP

1.4 Висновки з розділу 1

В даному розділі були розглянуті аналоги та рішення для часткового вирішення проблем, що виникають у роботі. Проаналізувавши їх, можна сказати, що ці сервіси працюють у вузькому обсязі свого призначення, і якщо оператор їх не вказує, то новий формат інформації не може бути оброблений.

2 ДОСЛІДЖЕННЯ ЗАСОБІВ ГЕНЕРАЦІЇ ВІДПОВІДЕЙ

2.1 Платформи для побудови чат-ботів

Чат-бот [5] – це комп’ютерна програма, розроблена на основі нейронної мережі та технології машинного навчання, і може використовувати слухові або текстові методи для ведення розмов. Чат-боти використовуються для досягнення цілей або для розваги. Чат-боти зазвичай використовуються в системах діалогу з різними практичними цілями, до складу яких входять пошук інформації або обслуговування клієнтів. Деякі чат-боти можуть використовувати більш складні системи за допомогою яких буде оброблюватись людська мова, але більшість зазвичай використовують простіші системи.

2.1.1 Чат-боти в Telegram

Telegram – це месенджер, який є програмнм забезпечення для смартфонів, планшетів та ПК з різними операційними системами, яке дозволяє обмінюватися повідомленнями які можуть включати текст, різноманітними файли, зокрема графічні файли та відеофайли, а також безкоштовно телефонувати іншим користувачам програми.

Після реєстрації обліковий запис користувача прив'язується до номера мобільного телефону: щоб пройти авторизацію, потрібно ввести код який прийде в СМС. Ці коди мають обмежені терміни дії, що дає можливість користувачу позбавитись необхідності запам'ятовувати свій пароль.

Телеграм [6] розроблений на технології шифрування листування під назвою MTProto. Він спочатку був експериментом компанії Digital Fortress, який призначений для тестування MTProto на великих навантаженнях.

Спеціально для месенджера було створено протокол MTProto, що даж можливість використання декількох протоколів шифрування. Під час

авторизації зазвичай використовуються алгоритми DH-2048 та RSA-2048, за для шифрування, під час процесу передачі повідомлень в мережу вони підлягають шифруванню AES з ключем, який відомий серверу та клієнту. Для того чтоб створити нового чат-бота є спеціальний бот (рис. 2.1).

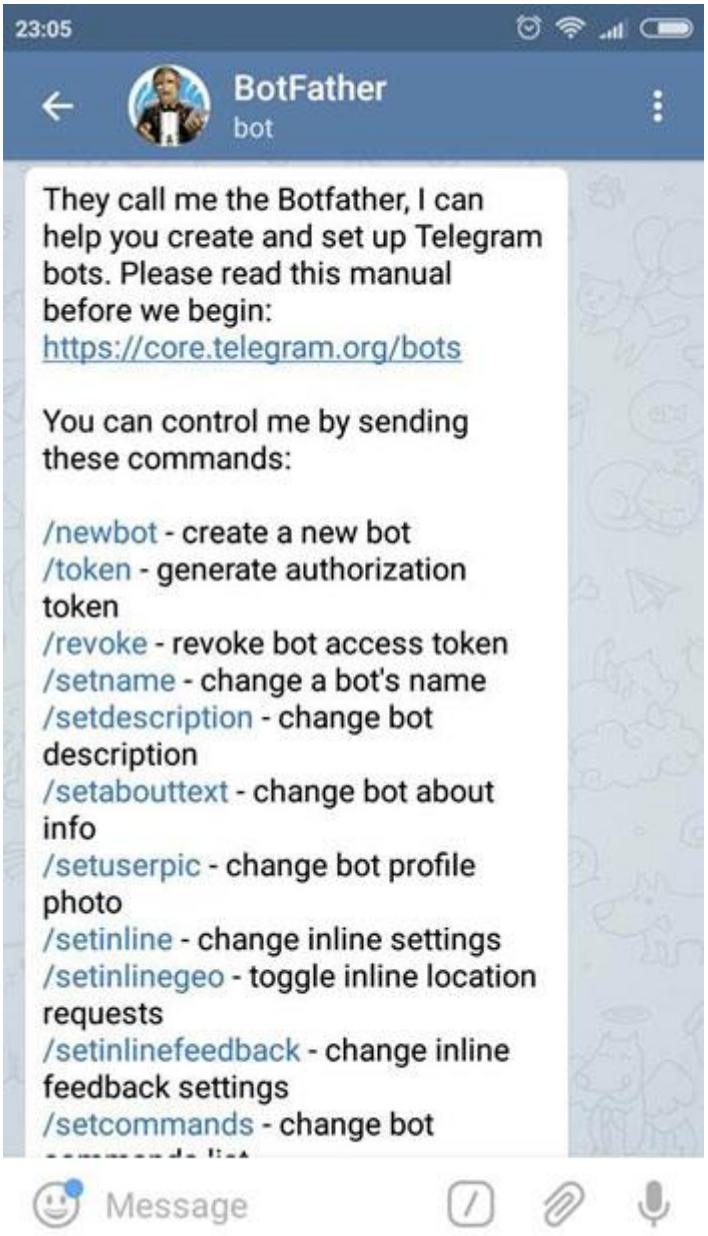

Рисунок 2.1 – Приклад роботи месенджеру Telegram та сервісу чат-ботів BotFather

Чат-бот створений в Телеграм може виконувати наступні задачі [7]:

– розсилати нотифікації чи новини;

– інтегруватися с іншими сервісами (Github, Youtube, тощо);

– отримувати платежі від користувачів Телеграм;

– створювати одно- та багатокористувацькі ігри;

– допомагати користувачам знаходити один одного за інтересами та інше.

2.1.2 Чат-боти в Facebook Messanger

Система обміну миттєвими повідомленнями Facebook Messenger (укр. Фейсбук Месенджер) – це додаток який надає можливість обміну повідомленнями, створений Facebook. Фейсбук месенджер інтегрований із системою обміну повідомленнями на головному веб-сайті Facebook і базується на відкритому протоколі MQTT [8]. Станом на квітень 2017 року щомісячна аудиторія месенджера становила 1 мільярд. Чат-боти Facebook засновані на особистих повідомленнях із загальнодоступною сторінкою від імені користувача..

Тому для створення бота потрібно буде створити сам додаток для доступу до API (рис. 2.2), і публічну сторінку, з якої будуть спілкуватися користувачі.

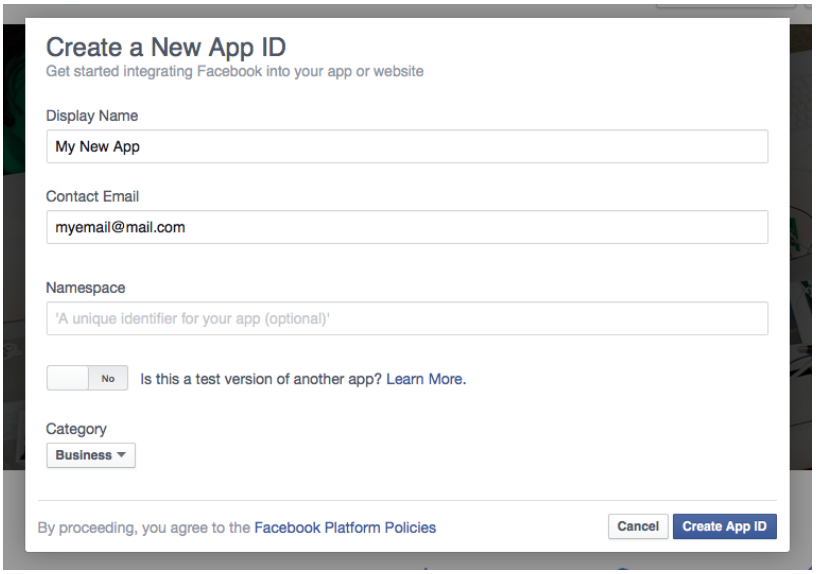

Рисунок 2.2 – Створення нового застосунку для Facebook API

Виділяють такі основні напрями чат-ботів створених в Facebook [9]:

– розсилання нотифікацій чи новин;

– технічна підтримка (відповіді на розповсюдженні запитання);

– продаж (сегментування клієнтів через «питання-відповідь» та запропонування найбільш релевантних товарів);

– інтегрування з іншими сервісами (Youtube, Spotify, тощо).

2.1.3 Обгрунтування вибору платформи для побудови чат-боту

Для створення нейронних мереж було обрано платформу Telegram. Цей вибір було зроблено на підставі наступних особливостей платформи:

– простота початкової ініціалізації чат-боту;

– зручний API – інтуїтивно зрозумілий для користувача Telegram API дозволяє просто і без зайвих проблем досягнути необхідної функціональності;

– наявність зручних Python-клієнтів для Telegram API.

2.2 Теорія методів машинного навчання

Основна мета системи що навчається - узагальнення з особистого досвіду. Узагальнення в цьому контексті - це здатність навчальної машини точно працювати над новими, невідомими прикладами та завданнями після набуття досвіду в навчальному наборі даних. Приклади навчання походять із загальновідомого розподілу ймовірностей (який вважається репрезентативним для простору справ), і система навчання повинна будувати загальну модель цього простору, що дозволяє досить точно передбачати нові випадки.

Обчислювальний аналіз алгоритмів машинного навчання [2] та їх ефективність є розділом теоретичної інформатики, відомим як обчислювальна

теорія навчання. Оскільки навчальні набори обмежені, а майбутнє непевне, теорія навчання, як правило, не гарантує правильності алгоритмів. Натомість імовірнісні рамки правильності є досить поширеними. Одним із способів кількісно визначити помилку узагальнення є компрометація зміщення та дисперсії.

Те, наскільки добре модель, натренована наявними прикладами, передбачує виходи для невідомих випадків, називається узагальненням. Щоб узагальнення було найкращим, складність його гіпотези повинна відповідати складності функції, яка лежить в основі даних. Якщо гіпотеза є менш складною за цю функцію, то ми недовчилися. Тоді ми підвищуємо складність, і похибка тренування знижується. Але якщо наша гіпотеза є занадто складною, то ми перевчилися. Після цього ми повинні знайти гіпотезу, яка має мінімальну похибку тренування.

На додачу до рамок продуктивності, теоретики обчислювального навчання досліджують часову складність та здійсненність навчання. В теорії обчислювального навчання обчислення вважається здійсненним, якщо його може бути виконано за поліноміальний час. Є два види результатів часової складності. Позитивні результати показують, що певного класу функцій може бути навчено за поліноміальний час. Негативні результати показують, що певних класів не може бути навчено за поліноміальний час.

Існує багато подібного між теорією машинного навчання та статистичним висновуванням, хоча вони використовують відмінні терміни.

2.3 Основні підходи у машинному навчанні

Том Мітчелл представив широко цитоване більш офіційне визначення: "Комп'ютерна програма, як кажуть, вчиться на досвіді E щодо певного класу T-проблем і вимірює продуктивність P, якщо її ефективність у T-задачах, виміряна P, покращується з досвідом E". визначення є видатним, оскільки воно

визначає машинне навчання через принципово оперативні, а не когнітивні терміни, таким чином, слідуючи пропозиціям Алана Тьюрінга в його роботі "Комп'ютери та розум" на питання "Чи можуть машини мислити?" замінити питанням "Чи можуть машини робити те, що ми може (як мислячі істоти)? "[10].

Завдання машинного навчання [11], як правило, поділяються на три великі категорії, залежно від характеру "сигналу", який вивчає система, або "зворотного зв'язку", доступного для системи навчання. Цими категорями є:

– навчання з учителем (кероване навчання, англ. supervised learning). Комп’ютер представлений прикладами входів та їх бажаних результатів, наданих «викладачем», і метою є навчити загальне правило, яке відображає входи та виходи;

– навчання без учителя (спонтанне навчання, англ. unsupervised learning). На вивчення алгоритму це не впливає, залишаючи його шукати структуру у своєму внеску. Навчання без вчителя може бути самоціллю (виявлення прихованих закономірностей у даних) або засобом досягнення цілі (ознаки навчання);

– навчання з підкріпленням (англ. reinforcement learning). Комп’ютерна програма взаємодіє з динамічним середовищем, в якому вона повинна досягти певної мети (наприклад, керування автомобілем), без того, щоб вчитель явно сказав їй, якщо вона наближається до мети. Інший приклад - навчання грі через гру з суперником.

Між керованим та спонтанним навчанням є напівавтоматичне навчання (англ. semi-supervised learning), в якому вчитель дає неповний тренувальний сигнал: тренувальний набір, в якому відсутні деякі (часто численні) цільові виходи. Окремим випадком цього принципу є трансдукція (англ. transduction), коли під час навчання відомий повний набір випадків задачі, крім частини цілей, яких бракує.

Метод опорного вектора - це класифікатор, який ділить свій вхідний простір на дві області, розділені лінійною межею. Тут він вчиться розрізняти чорні та білі кола.

Серед інших завдань машинного навчання - мета-навчання - навчається на основі власного попереднього досвіду. Розвиваюче навчання, призначене для навчання роботів, генерує власні послідовності навчальних ситуацій (їх також називають навчальними програмами) для накопичення репертуару нових навичок шляхом автономного самонавчання та соціальної взаємодії з вчителями-людьми. та використання провідних механізмів, таких як активне навчання, дозрівання, рухова синергія та наслідування.

Інша класифікація завдань машинного навчання виникає при розгляді бажаного результату роботи системи за допомогою машинного навчання:

– у класифікації (англ. classification) вхідні дані поділяються на два або більше класів [12], і система студента повинна генерувати модель, яка призначає безпрецедентні вхідні дані одному або декільком (багатозначна класифікація) цих класів. Зазвичай вони намагаються вирішити це контрольовано. Прикладами класифікації є спам-фільтри, в які вводяться електронні листи (або щось інше), а класи - "спам" та "не спам";

– у регресії (англ. regression), також кероване завдання, результати суцільні, не дискретні;

– у кластеруванні (англ. clustering) набір входів повинно бути поділено на групи. На відміну від класифікації, групування не відомо заздалегідь, що зазвичай робить його спонтанним навчальним завданням. Оцінка щільності визначає розподіл вхідних даних у певних просторах. Зменшення розміру може спростити введення, відображаючи в меншому просторі. Це завдання передбачає моделювання теми, в якій програма надає перелік документів людською мовою і має завдання знайти, які документи охоплюють подібні теми.

2.3.1 Навчання з учителем

Навчання з учителем [13] (англ. Supervised learning) – Метод машинного навчання, при якому існуючий набір зразків "стимул-реакція" використовується для примусової тестової системи для визначення "реакції" на "стимул", який не належить до існуючого набору зразків. Це кібернетичний експеримент з кібернетичної точки зору.

Між входами та визначенними як еталонні виходами (стимул-реакція) може існувати деяка залежність, але вона заздалегідь не відома. Відомий лише остаточний набір прецедентів - пара "стимул-реакція", яка називається навчальним зразком. На основі цих даних необхідно відновити залежність (побудувати модель взаємозв'язку стимул-реакція, придатну для прогнозування), тобто побудувати алгоритм, який може дати досить точну відповідь на будь-який об'єкт. Для вимірювання точності відповідей, а також при навчанні прикладів можна ввести якісний функціонал.

Формально задача навчання може бути сформульована наступним чином: існує деяка сукупність «стимулів» x і «реакцій на стимули» y, необхідно визначити залежність між x та y таку, що в межах припустимої помилки e буде справедливим $f(x) - e \leq y \leq f(x) + e$ [13].

2.3.2 Типологія завдань навчання з учителем

Існують наступні типи вхідних даних:

– ознаковий опис – найпоширеніший випадок. кожен об'єкт описується набором своїх характеристик, що називаються ознаками. Ознаки можуть бути числовими або нечисловими;

– матриця відстаней між об'єктами. Кожен об'єкт описується відстанями до всіх інших об'єктів навчальної вибірки. З цим типом вхідних

даних працюють деякі методи, зокрема, метод *k* найближчих сусідів, метод парзенівського вікна, метод потенційних функцій;

– часовий ряд або сигнал, що є послідовністю вимірювань в часі. Кожне вимірювання може представлятися числом, вектором, а в загальному випадку – ознаковим описом досліджуваного об'єкта в даний момент часу;

– зображення або відеоряд.

Зустрічаються й складніші випадки [13], коли вхідні дані подаються у вигляді графів, текстів, результатів запитів до бази даних тощо. Як правило, вони приводяться до першого або другого випадку шляхом попередньої обробки даних та виділяння ознак.

Також можна виокремити такі типи відгуків:

– коли множина можливих відповідей нескінченна (відповіді є дійсними числами або векторами), говорять про задачі регресії та апроксимації;

– коли множина можливих відповідей звичайна, говорять про задачі класифікації та розпізнавання образів;

– коли відповіді характеризують майбутню поведінку процесу або явища, кажуть про задачі прогнозування.

2.3.3Вироджені види систем управління підкріпленням

Система підкріплення з керуванням по реакції (*R*-керована система) характеризується тим, що інформаційний канал від зовнішнього середовища до системи підкріплення не функціонує. Дана система, незважаючи на наявність системи управління, відноситься до спонтанного навчання, оскільки випробовувана система навчається автономно, під дією лише своїх вихідних сигналів незалежно від їх «правильності». При такому методі навчання для управління зміною стану пам'яті не потрібно ніякої зовнішньої інформації.

Система підкріплення з керуванням по стимулах (*S*-керована система) характеризується тим, що інформаційний канал від випробовуваної системи до

системи підкріплення не функціонує. Незважаючи на не функціонування каналу від виходів випробовуваної системи, відноситься до навчання з учителем, оскільки в цьому випадку система підкріплення (вчитель) змушує випробувану систему виробляти реакції згідно певного правила, хоча й не береться до уваги наявність істинних реакцій випробовуваної системи.

2.3.4 Навчання без учителя

Навчання без вчителя (англ. Unsupervised learning, самоосвіта, спонтанне навчання) – Одним із методів машинного навчання є розв’язання методу, за допомогою якого тестова система вчиться виконувати завдання спонтанно, без втручання експериментаторів. З кібернетичної точки зору, навчання без вчителя є одним із видів кібернетичних експериментів. Як правило, це стосується лише проблеми відомих описів наборів об'єктів (навчальних зразків), і необхідно виявити внутрішні взаємозв'язки, залежності та закономірності, що існують між об'єктами.

Коли кожного учня змушують «правильно відповісти», і необхідно знайти зв’язок між стимулом та системною реакцією, навчання без вчителя [14] часто контрастує з навчанням з учителем.

Незважаючи на численні прикладні досягнення, викладання з викладачем критикували за його біологічну неправдоподібність. Важко уявити механізм навчання в мозку, який порівнює бажані та фактичні значення результатів, виконуючи корекцію зворотного зв'язку. Якщо припустити подібний механізм у мозку, звідки беруться бажані результати? Навчання без вчителя - це набагато більш правдоподібна модель навчання в біологічній системі. Розроблений Кохоненом та багатьма іншими, він не вимагає цільового вектора для результатів, тому його не потрібно порівнювати з певними ідеальними відповідями.

Щоб побудувати теорію і відійти від кібернетичного експерименту, різні теорії намагаються формалізувати експеримент із навчанням без вчителя математично. Існує безліч різних підтипів встановлення та визначення цієї формалізації. Один з яких знайшов своє відображення в теорії розпізнавання зразків.

Цей відхід від експерименту та побудова теорії пов'язані з різними поглядами експертів. Відмінності, зокрема, виявляються у відповіді на запитання: "Чи є єдино можливими принципами адекватного опису зображень різного характеру, або такий опис завжди є завданням для професіоналів, що мають конкретні знання?".

У першому випадку виробництво має бути спрямоване на виявлення загальних принципів використання апріорної інформації [15] при складанні адекватного опису зображень. Важливо, що тут апріорна інформація про зображення різного характеру різна, а принцип їх обліку однаковий. У другому випадку проблема отримання опису виходить за рамки загального твердження, і теорія навчання машинам розпізнавання зображень з точки зору статистичної теорії вивчення розпізнавання зображень може бути зведена до проблеми мінімізації середнього ризику в спеціальному класі правила прийняття рішень.

В теорії розпізнавання образів розрізняють в основному три підходи до даної проблеми:

– евристичні методи;

– математичні методи;

– лінгвістичні (синтаксичні) методи.

Експериментальна схема навчання без вчителя часто використовується в теорії розпізнавання зразків, машинного навчання. Водночас, залежно від підходу, це формалізується в тому чи іншому математичному понятті. І лише в теорії штучних нейронних мереж проблема вирішується експериментально, використовуючи той чи інший тип нейронних мереж. При цьому, як правило, отримана модель може не мати інтерпретацій, які іноді відносять до недоліків

нейронних мереж. Однак отримані результати не гірші, і при бажанні можуть бути інтерпретовані за допомогою спеціальних методів.

2.4 Алгоритми оптимізації цільової функції нейронних мереж

Градієнтний спуск [16] – це один з найпопулярніших алгоритмів для оптимізації та найпоширеніший спосіб оптимізації нейронних мереж. Градієнтний спуск – це спосіб мінімізувати цільову функцію *J(θ)*, параметризовану параметрами моделі $\theta \in R^d$, оновлюючи параметри в протилежному напрямку градієнта цільової функції $\nabla_\theta J(\theta)$ відносно до параметрів. Швидкість навчання η визначає розмір кроків, які ми беремо для досягнення (локального) мінімуму. Є три варіанти градієнтного спуску, які відрізняються тим, скільки даних використовується для обчислення градієнта цільової функції. Залежно від обсягу даних досягається компроміс між точністю оновлення параметрів та часом, необхідним для виконання оновлення.

2.4.1 Градієнтний спуск

Звичайний градієнтний спуск [16] вираховує градієнт функції витрат відносно параметрів θ для всього набору даних:

$$\theta = \theta - \eta * \nabla_\theta J(\theta). \tag{2.1}$$

Оскільки необхідно підрахувати градієнти для цілого набору даних для виконання всього одного оновлення, градієнтний спуск може бути дуже повільним і не підходящим для наборів даних, що не поміщаються у пам’ять. Звичайний градієнтний спуск також не дозволяє оновлювати модель на льоту, використовуючи нові приклади (лістинг 1).

Лістинг 1. Звичайний градієнтний спуск

```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(loss_function, data, params)
  params = params - learning_rate * params_grad
```

Оновлення параметрів відбувається у напрямку отриманих градієнтів на відстань, що визначається значенням параметру «швидкість навчання». Звичайний градієнтний спуск гарантовано сходиться до глобального мінімуму для опуклих поверхонь помилок та до локального мінімуму для неопуклих поверхонь.

2.4.2Стохастичний градієнтний спуск

Стохастичний градієнтний спуск[17] виконує оновлення параметрів для кожного прикладу $x^{(i)}$ та мітки $y^{(i)}$:

$$\theta = \theta - \eta * \nabla_\theta J(\theta; x^{(i)}; y^{(i)}). \quad (2.2)$$

Звичайний градієнтний спуск виконує надлишкові обчислення для великих наборів даних, так як виконується повторне обчислення градієнтів для схожих прикладів перед кожним оновленням параметрів. Стохастичний градієнтний спуск, в свою чергу, уникає цієї надлишковості завдяки виконанню одного оновлення за раз. Тому такий підхід зазвичай набагато швидший та дозволяє навчатися на нових даних. Проте, часті оновлення з високим значенням дисперсії призводять до значних коливань цільової функції.

В той час, як звичайний градієнтний спуск сходиться до локального мінімуму, коливання цільової функції стохастичного градієнтного спуску дозволять перестрибнути до нового, потенційно кращого локального мінімуму. З іншої сторони – це значно ускладнює сходження до точного мінімуму,

оскільки алгоритм буде продовжувати перестрибувати. Тим не менш, ця проблема вирішується поступовим зменшенням швидкості навчання, що дозволяє досягти майже гарантованого сходження до глобального або локального мінімумів для опуклих та неопуклих поверхонь відповідно.

Лістинг 2. Стохастичний градієнтний спуск

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function,
example, params)
    params = params - learning_rate * params_grad
```

2.4.3 Градієнтний спуск міні-партій

Поєднання двох попередніх методів виконує оновлення параметрів для кожної міні-партії з n екземплярів для навчання за формулою:

$$\theta = \theta - \eta * \nabla_\theta J(\theta; x^{(i;i+n)}; y^{(i;i+n)}). \quad (2.3)$$

Такий підхід, по-перше, зменшує коливання оновлень параметрів, що призводить до більш стабільного сходження, і, по-друге, дозволяє використання високо оптимізованих матричних операцій, що часто використовуються в бібліотеках глибокого навчання , що, в свою чергу робить обчислення градієнтів для міні-партій [18] дуже ефективним. Найбільш розповсюджений розмір партії коливається від 50 до 256, але це значення залежить від типу мережі і сфери застосування.

Лістинг 3. Градієнтний спуск міні-партій

```
for i in range(nb_epochs):
```

```
        np.random.shuffle(data)
        for batch in get_batches(data, batch_size=50):
          params_grad = evaluate_gradient(loss_function,
batch, params)
          params = params - learning_rate * params_grad
```

2.4.4 Градієнтний спуск з імпульсом

Градієнтний спуск має проблеми з переміщенням в ущелинах, тобто в таких ділянках, де поверхня є значно крутішою в одному вимірі, ніж у іншому, що є розповсюдженою ситуацію біля локального мінімуму. У цих сценаріях, градієнтний спуск коливається по всьому схилу ущелини, роблячи лише ті кроки, що дуже коливаються, до локального мінімуму, як на Рисунок 2.3.

Рисунок 2.3 – Градієнтний спуск без імпульсу

Імпульс – це метод, що дозволяє прискорити градієнтний спуск у доречному напрямку і пом’якшує коливання [19], як показано на Рисунок .

Це досягається додаванням фракції γ вектору оновлення з минулого кроку до теперішнього вектору оновлень за формулами:

$$\nu_t = \gamma \nu_{t-1} + \eta \nabla_\theta J(\theta); \tag{2.4}$$

$$\theta = \theta - \nu_t. \tag{2.5}$$

Значення імпульсу збільшується для напрямків, градієнти яких направлені у одну і ту ж сторону та зменшується для напрямків, градієнти яких змінюють свій напрям. Як результат, досягається швидше сходження і зменшується ступінь коливань.

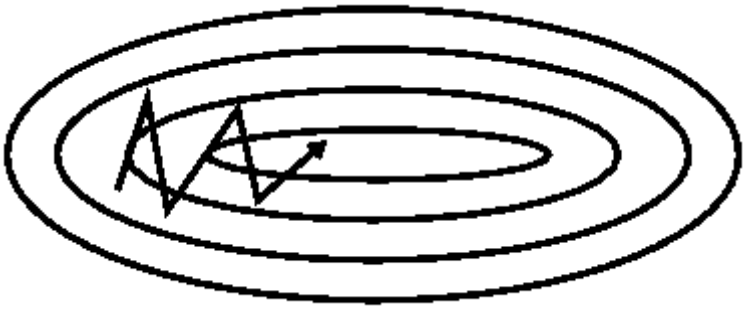

Рисунок 2.4 – Градієнтний спуск з імпульсом

2.4.5Градієнтний спуск з імпульсом Нестерова

Імпульс Нестерова [19] покращує звичайний імпульс за рахунок прогнозування зміни поверхні функції для наступних кроків і використання цього знання при встановленні значення імпульсу для теперішнього кроку. Значення імпульсу γv_{t-1} використовується для оновлення параметрів θ. Обчислення $\theta - \gamma v_{t-1}$ надає наближене значення наступної позиції параметрів, тобто грубу оцінку того, якими будуть параметри після наступного кроку. Таким чином існує можливість ефективного обчислення градієнтів не відносно поточних параметрів θ, а відносно майбутньої позиції параметрів за формулою:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}); \quad (2.6)$$

Тоді як звичайний імпульс [20] спочатку обчислює поточний градієнт (малий синій вектор на Рисунок 2.5 –), а потім робить великий стрибок у напрямку оновленого накопиченого градієнта (великий синій вектор), імпульс

Нестерова спершу робить великий стрибок у напрямку попереднього накопиченого градієнта (коричневий вектор) вимірює градієнт, а потім робить корекцію (червоний вектор), що призводить до повного оновлення (зелений вектор). Це попереджувальне оновлення забороняє просуватися надто швидко, і це призводить до покращеної віддачі.

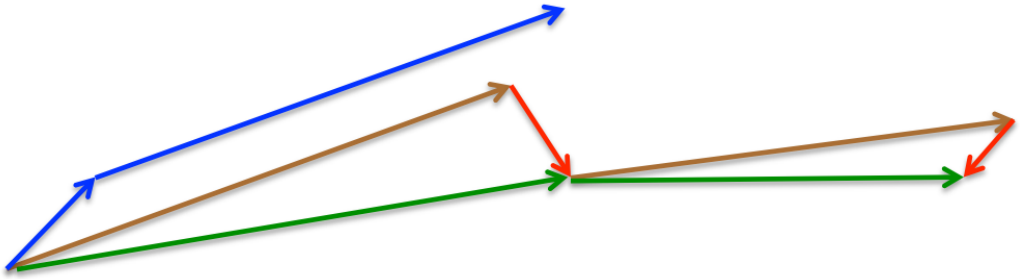

Рисунок 2.5 – Імпульс Нестерова

2.4.6 Алгоритм оптимізації цільової функції Adagrad

Алгоритм Adagrad [20] заснований на градієнтному спуску, який адаптує значення швидкості навчання до значень параметрів, виконуючи більші оновлення для параметрів, що зустрічаються відносно рідко, та менші – для параметрів, що зустрічаються частіше. Алгоритм добре підходить для оптимізації цільової функції на розріджених даних.

Попередні методи виконували оновлення всіх параметрів θ за раз, використовуючи однакове значення швидкості навчання η для кожного θ_i. Так як Adagrad використовує різні значення швидкості навчання для кожного параметра θ_i, в кожний момент часу t, то оновлення параметрів відбувається за наступною формулою:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}+\epsilon}} * g_{t,i} \,, \tag{2.7}$$

де G_t – це діагональна матриця, де кожен елемент i,i – це сума квадратів градієнтів відносно θ_i в момент часу t, а ϵ – згладжувальне значення для уникнення ділення на 0. Так як G_t містить суму квадратів минулих градієнтів відносно всіх параметрів θ на діагоналі, є можливість векторизувати реалізацію завдяки виконанню поелементного множення матриці G_t на вектор g_t за наступною формулою:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t. \tag{2.8}$$

Одна з основних переваг алгоритму Adagrad полягає в тому, що він прибирає необхідність вручну підбирати значення швидкості навчання. Більшість реалізацій використовують незмінне стандартне значення 0.01.

Основним недоліком алгоритму є накопичення квадратів градієнтів у знаменнику. Так як кожне нове додане значення є додатнім, накопичена сума продовжує зростати впродовж навчання, що призводить до поступового зменшення швидкості навчання, яке може стати дуже малим з плином часу, і, в певний момент, алгоритм не здатен буде здобути будь-які додаткові знання з даних.

2.4.7Алгоритм оптимізації цільової функції RMSprop

RMSprop [21] – неопублікований адаптивний метод оптимізації, запропонований Джефом Хінтоном. Він був розроблений з метою розв’язання проблеми затухаючого значення швидкості навчання методу Adagrad. Оновлення параметрів відбувається за формулами:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2; \tag{2.9}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t. \tag{2.10}$$

RMSprop [22] ділить швидкість навчання на експоненційно затухаюче середнє значення квадратичних градієнтів. Хорошим початковим значенням швидкості навчання η вважається 0.001.

RMSprop лежить у царині методів адаптивного навчання, які набувають все більшої популярності в останні роки, але також отримують певну критику [23]. Він знаменитий тим, що не публікується, але дуже відомий; Найбільш глибокі рамки навчання включають його реалізацію поза межами.

Існує два способи впровадження RMSprop. По-перше, слід розглядати це як адаптацію алгоритму Rprop для міні-пакетного навчання. Це було початковою мотивацією для розробки цього алгоритму. Інший спосіб – розглянути його схожість з Адаградом [24] та розглянути RMSprop як спосіб вирішити його радикально зменшувані показники навчання.

Почнемо з розуміння Rprop – алгоритму, який використовується для повної пакетної оптимізації. Rprop [25] намагається вирішити проблему, що градієнти можуть сильно відрізнятися за величиною. Деякі градієнти можуть бути крихітними, а інші – величезними, що спричиняє дуже складну проблему – намагаючись знайти єдиний глобальний рівень навчання для алгоритму. Якщо використовувати повноцінне навчання, то можна впоратися з цією проблемою, лише використовуючи знак градієнта. Тоді можна гарантувати, що всі оновлення ваги однакового розміру. Це коригування дуже допомагає з точками сідла та плато, якщо кроки досить великі навіть при крихітних градієнтах. Зауважте, що не можна цього зробити, адже збільшуючи рівень навчання, оскільки кроки, які ми робимо з великими градієнтами, будуть ще більшими, що призведе до розбіжності. Rprop поєднує ідею лише використання знака градієнта з ідеєю адаптації розміру кроку індивідуально для кожної ваги.

Для регулювання розміру кроку для деякої ваги використовується наступний алгоритм:

Якщо ознаки двох останніх градієнтів для ваги мають однаковий знак, це означає, що обрано правильний напрямок, і потрібно прискорити його невеликою часткою, тобто збільшити розмір кроку мультиплікативно (наприклад, у 1,2 раза). Якщо вони різні, це означає, що зроблено занадто великий крок і тому слід зменшити розмір кроку мультиплікативно (наприклад, в 0,5).

Потім обмежується розмір кроку між деякими двома значеннями. Ці значення дійсно залежать від програми та набору даних, хороші значення, які можуть бути за замовчуванням, становлять 50 і мільйонний, що є гарним початком.Тепер ми можемо застосувати оновлення ваги.

Rprop насправді не працює, коли використовуються дуже великі набори даних і тоді потрібно проводити оновлення міні-пакетних ваг. Причина, по якій це не працює, полягає в тому, що вона порушує центральну ідею стохастичного градієнтного спуску, яка є, при достатньо малій швидкості навчання, і це в середньому градієнти на наступних міні-партіях. Використовуючи вагу градієнта 0,1 на дев'яти міні-партіях, та вагу градієнта -0,9 на десятих міні-партіях очікуєеться, що ці градієнти приблизно відміняють один одного, щоб залишитися приблизно однаковим. Але це не те, що відбувається з Rprop. За допомогою Rprop збільшується вага в 9 разів і зменшується лише один раз, тому вага збільшується набагато більше.

Щоб поєднати надійність Rprop, ефективність, яка отримується від міні-партій, і усереднення по міні-партіях, що дозволяє комбінувати градієнти правильно, треба дивитися на Rprop з іншої точки зору. Rprop еквівалентний використанню градієнта, але також ділиться на величину градієнта, тому результатом є однакова величина, незалежно від того, наскільки великий цей градієнт малий. Проблема міні-партій полягає в тому, що кожен раз ділиться на різні градієнти. Центральною ідеєю RMSprop є збереження ковзного середнього квадратичного градієнта для кожної ваги. З математичними рівняннями правило оновлення має наступний вигляд.

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)(\frac{\delta C}{\delta w})^2; \quad (2.11)$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}}\frac{\delta C}{\delta w}. \quad (2.12)$$

Як видно з вищенаведеного рівняння, ми адаптуємо швидкість навчання шляхом ділення на корінь градієнта квадрата, але оскільки у нас є лише оцінка градієнта на поточну міні-партію, замість цього потрібно використовувати ковзну середню. Значення за замовчуванням для ковзного середнього параметра, який використовується, становить 0,9. Він працює дуже добре для більшості програм [26].

2.4.8 Алгоритм Adam

Adam [27] – ще один метод, що обчислює адаптивні значення швидкості навчання для кожного параметра. На додачу до зберігання експоненційно затухаючих середніх значень минулих квадратичних градієнтів v_t, як це було у RMSprop, Adam також зберігає експоненційно затухаючі середні значення минулих градієнтів m_t за наступними формулами:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \quad (2.13)$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 , \quad (2.14)$$

де m_t та v_t - це оцінки першого моменту (середнього значення) та другого моменту (нецентрована дисперсія) градієнтів відповідно[19]. Так як ці значення на початку ініціалізовані нульовими векторами, спостерігалося їх наближення до нуля, особливо під час початкових кроків алгоритму, коли

швидкості розпаду низькі (отже, β_1 та β_2 близькі до 1). Протидією цьому є обчислення корекції наближення за наступними формулами:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}; \tag{2.15}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}. \tag{2.16}$$

Після чого ці скореговані значення використовуються при оновленні параметрів, тобто формулою оновлення параметрів для Adam є:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t+\epsilon}}\hat{m}_t. \tag{2.17}$$

Хорошими початковими значеннями є $\beta_1 = 0.9$, $\beta_2 = 0.999$ та $\epsilon = 10^{-8}$. На даний момент, Adam є оптимальним вибором оптимізатора з адаптивним значенням швидкості навчання, оскільки він не потребує підбору гіперпараметрів і, в середньому, показує найкращі результати на практичних задачах без витрати значної кількості часу на його налаштування.

2.4.9Алгоритм Nadam

Як було зазначено раніше, алгоритм Adam може розглядатися як поєднання RMSprop та імпульсу. RMSprop надає експоненційно затухаючі середні значення минулих квадратів градієнтів ν_t, в той час як імпульс враховує експоненційно затухаюче середнє минулих градієнтів mt. Також було доведена перевага імпульсу Нестерова над звичайним імпульсом.

Nadam [28] в свою чергу поєднує Adam та імпульс Нестерова. Імпульс Нестерова [20] дозволяє виконувати більш точний крок у напрямку градієнта завдяки оновленню параметрів кроком імпульсу перед власне обчисленням

градієнта. Але замість того, щоб застосовувати крок імпульсу двічі – перший раз для оновлення градієнту g_t та другий раз для оновлення параметрів θ_{t+1} – вектор майбутніх імпульсів використовується для оновлення поточних параметрів за формулами:

$$g_t = \nabla_{\theta_t} J(\theta_t) \, . \tag{2.18}$$

$$m_t = \gamma m_{t-1} + \eta g_t \, . \tag{2.19}$$

$$\theta_{t+1} = \theta_t - (\gamma m_t + \eta g_t) \, . \tag{2.20}$$

Таким чином, імпульс Нестерова додається завдяки заміні скоригованих прогнозів вектору імпульсу попереднього кроку часу $\widehat{m}_{t-1}$ на скориговані прогнози поточного вектору імпульсу $\widehat{m}_t$, що призводить до наступного правила оновлення для Nadam:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}(\beta_1 \widehat{m}_t + \frac{(1-\beta_1)g_t}{1-\beta_1^t}). \tag{2.21}$$

Теоретично, Nadam досягає мінімуму функції швидше за Adam, проте це досить новий оптимізатор, який ще не був добре досліджений і перевірений на реальних задачах машинного навчання, а тим більше для задач синтезу зображень. Тому у подальших дослідженнях буде використовуватись алгоритм оптимізації Adam.

2.4.10 Обгрунтування вибору алгоритма оптимізації

Після дослідження сучасних методів оптимізації нейроних мереж, для використання у дипломній роботі було обрано алгоритм RMSprop, через швидкість та надійність його роботи. Також RMSprop один за найпопулярніших

алгоритмів оптимізації нейроних мереж, та широко використовується у задачах з генерацією тексту.

2.5 Фреймворки для глибинного навчання

Сучасні архітектури нейронних мереж складаються з багатьох шарів і потребують значної швидкодії і коректної реалізації математичних операцій, тому під час розробки моделей глибоко навчання використовуються фреймворки, що абстрагують значну частину математики, що дозволяє сконцентруватись на аналізі даних та розробці підходящої моделі. Також, використання фреймворків які вже є досить популярними дозволяє уникнути значної кількості випадкових помилок, оскільки більшість з них добре протестована значною кількістю користувачів зі всього світу.

У таблиці 2.1 наведено опис і порівняння найбільш популярних фреймворків для глибинного навчання.

Таблиця 2.1 – Порівняння фреймворків глибинного навчання

Назва	**Від-крите**	**Платформа**	**Підтри мка CUDA**	**Реку-рентні мережі**	**Згорт-кові мережі**	**Пара-лельне виконання**
1	2	3	4	5	6	7
Apache Singa	Так	Linux, Mac OS X, Windows	Так	Так	Так	Так

Продовження таблиці 2.1

1	2	3	4	5	6	7
Caffe	Так	Linux, Mac OS X, Windows	Так	Так	Так	?
Deeplearning4j	Так	Linux, Mac OS X, Windows, Android (багато платформне)	Так	Так	Так	Так
Dlib	Так	багато платформне	Так	Ні	Так	Так
Keras	Так	Linux, Mac OS X, Windows	Так	Так	Так	Так
Microsoft Cognitive Toolkit	Так	Windows, Linux (OSX в планах через Docker)	Так	Так	Так	Так
MXNet	Так	Linux, Mac OS X, Windows,AWS, Android,iOS, JavaScript	Так	Так	Так	Так
Neural Designer	Ні	Linux, Mac OS X, Windows	Ні	Ні	Ні	?
OpenNN	Так	багатоплатформне	Ні	Ні	Ні	?

Продовження таблиці 2.1

1	2	3	4	5	6	7
Tensor Flow	Так	Linux, Mac OS X, Windows	Так	Так	Так	Так
Theano	Так	багатоплатформне	Так	Так	Так	Так
Torch	Так	Linux, Mac OS X, Windows,Android, iOS	Так	Так	Так	Так
Mathematica	Ні	Windows, Mac OS X, Linux, Хмарні обчислення	Так	Так	Так	Так

2.5.1 Огляд фреймворку TensorFlow

TensorFlow [29] – бібліотека машинного навчання з відкритим кодом, розроблена Google для задоволення її потреб у системах, здатних будувати та навчати нейромережі для виявлення та розшифровки зображень та кореляцій, подібних до навчання та розуміння, що використовуються людьми. В даний час він використовується як для досліджень, так і для розробки продуктів Google, часто замінюючи свого закритого попередника DistBelief. Спочатку TensorFlow був розроблений командою Google Brain для внутрішнього використання Google, доки він не вийшов під відкритою ліцензією Apache 2.0 9 листопада 2015 р. [31].

Хоча посилальна реалізація працює на окремих пристроях, TensorFlow може працювати на декількох процесорах та графічних процесорах (включаючи додаткові розширення CUDA для загальних обчислень GPU). TensorFlow

доступний для 64-розрядних Linux, macOS, Windows та для мобільних обчислювальних платформ, включаючи Android та iOS.

Розрахунки TensorFlow виражаються як стани графіку потоку даних. Назва TensorFlow походить від операцій, які такі нейронні мережі виконують на багатовимірних наборах даних. Ці багатовимірні масиви називаються "тензорами". У червні 2016 року Джефф Дін від Google заявив, що TensorFlow згадав 1500 сховищ на GitHub, лише 5 з яких були від Google.

У травні 2016 року Google оголосив про свій блок обробки тензорів (TPU), спеціалізований чіп, побудований спеціально для машинного навчання та адаптований до TensorFlow. TP - це програмований прискорювач ШІ, розроблений для забезпечення високої продуктивності в низькоточній арифметиці (наприклад, 8-розрядної) і спрямований на використання або виконання моделей, а не на їх навчання. Google оголосив, що вони використовують TP у своїх центрах обробки даних вже більше року, і виявив, що вони забезпечують набагато краще оптимізовану продуктивність на ват для машинного навчання [32].

Серед програм, основою яких є TensorFlow, є програмне забезпечення для автоматизованого опису зображень, таке як DeepDream. 26 жовтня 2015 року Google офіційно запровадив RankBrain, який підтримує TensorFlow. Зараз RankBrain обробляє значну кількість записів пошуку, замінюючи та доповнюючи традиційні статичні алгоритми які працюють на основі результатів пошуку.

TensorFlow надає API для різних мов програмування, основним з яких є Python, що має найкращу документацію та підтримку, а також отримує найшвидші оновлення. Також підтримуються API для C++, Haskell, Java, Go [33].

2.5.2 Огляд фреймворку Keras

Keras[34] – це відкрита нейромережева бібліотека, написана мовою Python. Вона вміє працювати поверх TensorFlow та Theano. Спроектовану для усунення швидких експериментів із вимірюванням глибинного навчання, вона розподілена в той час, щоб вона була мінімальною, модульною та розширюваною. Її було створено як частину дослідницького зусилля проекту ONEIROS (англ. Open-end Neuro-Electronic Intelligent Robot Operating System), її основним автором та підтримкою є Франсуа Шоллє (фр. François Chollet), інженер Google.

І хоч команда TensorFlow Google вирішила підтримку Keras в основній бібліотеці TensorFlow, Шоллє сказав, що Keras [35] було замислено радше як інтерфейс, аніж як наскрізна система машинного навчання. Вона представляє високорівневий, інтуїтивний набір абстракцій, який робить простішим формування нейронних мереж, не залежно від тилової бібліотеки наукових викладів.

Ця бібліотека містить численні реалізації широко розповсюджених будівельних блоків нейронних мереж, таких як шари, цільові та передавальні функції, оптимізатори та безліч інструментів для спрощення роботи із зображеннями та текстом. Їх код розміщений на GitHub, спільна підтримка форумів включає сторінку запитів GitHub, канал Gitter і канал Slack.

2.5.3 Огляд фреймворку Theano

Theano – бібліотека чисельного обчислення в Python. Обчислення в Theano виражаються NumPy-ським синтаксисом і компілюються для ефективних паралельних обчислень як на звичайних CPU, так і на GPU.

Theano [36] є відкритим проектом, основним розробником якого є група машинного навчання в Монреальському університеті.

Theano являє собою препроцесор на мові типу python для системи обчислень з багатовимірними масивами даних (тензорами), що поєднує в собі математичні пакети Mathematica і MATLAB. Вхідна мова для Theano близька до мови symPy – мови символьних виразів для Mathematica, і NumPy для MATLABa.

Основні математичні методи, операції, і структури даних, підтримувані Theano [37]:

– робота з тензорами через структуру numpy.ndarray і підтримка безлічі тензорних операцій;

– робота з розрідженими матрицями через структури scipy;

– численні методи лінійної алгебри, включаючи досить складні;

– можливість в режимі роботи створювати нові операції з графами;

– численні операції про перетворення графів;

– підтримка мови python версій 2 і 3;

– використання архітектури CUDA для мультипроцессорної роботи з тензорами;

– підтримка стандарту Basic Linear Algebra Subprograms (BLAS) для процедур лінійної алгебри;

Планується використання і підтримка наступних середовищ – C/C++, CUDA, OpenCL, PTX, CAL, AVX.

2.5.4 Огляд фреймворку Torch

Torch – це відкрита бібліотека для машинного навчання, система створена для наукових обчислень та мова сценаріїв розроблена на основі мови програмування Lua. Пропонує широкий спектр алгоритмів для глибинного машинного навчання і використовує мову сценаріїв LuaJIT та реалізацію мовою C в основі.

Основним пакетом бібліотеки Torch [38] є torch. Він забезпечує гнучкий N-вимірний масив, або тензор, який підтримує основні процедури для індексування, розшаровування, транспозиції, приведення типів, зміни розмірів, розподілення зберігання та клонування. Цей об'єкт використовується більшістю інших пакетів, і відтак є центральним об'єктом бібліотеки. Тензор також підтримує математичні операції, такі як max, min, sum, статистичні розподіли, такі як рівномірний, нормальний та поліноміальний, та операції основних підпрограм лінійної алгебри, такі як скалярний добуток, матрично-векторне множення, матрично-матричне множення, матрично-векторний скалярний добуток та матричний скалярний добуток.

Пакет torch [39] також спрощує об'єктно-орієнтоване програмування та послідовність, надаючи різні допоміжні функції, що використовуються в його пакеті. Функцію torch.class (classname, parentclass) можна використовувати для створення фабрии об'єктів (класів). Під час виклику конструктора torch ініціалізує та використовує визначену користувачем мета-таблицю для встановлення таблиці Lua, яка використовує таблицю Lua як об'єкт.

Якщо об'єкти, створені фабрикою torch, не містять посилань на об'єкти, які неможливо сортувати (наприклад, Lua та userdata Lua), вони також можуть бути послідовними. Однак якщо обгорнуто дані userdata в таблицю (або мета-таблицю), яка забезпечує методи read () і write (), ви можете слідувати їм.

Програмний пакет nn використовується для побудови нейронних мереж. Він розділений на модульні об'єкти через загальний інтерфейс модуля. Модулі мають прямий () та зворотний () методи, які дозволяють їм здійснювати пряме і зворотне поширення відповідно. Модульні лінкери (такі як послідовний, паралельний та Concat) можуть використовуватися для підключення модулів для створення складної, орієнтованої на завдання графіки. Найпростіші модулі (такі як Linear, Tanh та Max) є основними компонентними модулями. Модульний інтерфейс може автоматично розрізняти кроки.

Функції втрат мають реалізацію як підкласи класу Criterion, що має інтерфейс, подібний до Module. Також він має методи forward() і backward() для обчислення втрат і відповідно зворотного поширення градієнтів. Критерії є корисними для тренування нейронної мережі на класичних задачах

Він також має клас StochasticGradient для навчання нейронних мереж із використанням методу стохастичного градієнта, хоча пакет Optim надає набагато більше функцій у цьому відношенні, таких як регуляція втрат моменту та градієнта.

Torch використовується дослідницькою групою ШІ Facebook, IBM, Yandex та дослідницьким інститутом IDIAP. Torch розширено для використання на Android та iOS. Він використовувався для реалізації апаратних реалізацій потоків даних, подібних до тих, що існують у нейронних мережах.

Facebook випустив набір модулів розширення як програмне забезпечення з відкритим кодом.

2.5.5 Огляд фреймворку Caffe

Caffe [40] – це система для глибинного навчання, первинно розроблена Янці Дзя (англ. Yangqing Jia) як частина його докторської праці в Каліфорнійському університеті в Берклі. Вона є відкритою, з ліцензією BSD. Її написано мовою C++ з інтерфейсом для Python.

Caffe підтримує багато різних типів архітектур глибинного навчання, орієнтованих на класифікацію та сегментування зображень, також підтримує конструкції повноз'єднаних нейронних мереж. Caffe підтримує прискорення на основі ГП із застосуванням CuDNN Nvidia.

Caffe застосовують в академічних дослідницьких проектах, стартапних прототипах та навіть у великомасштабних промислових застосуваннях у баченні, мовленні та мультимедіа. Yahoo! також інтегрувала caffe з Apache

Spark для створення caffeonspark, що інтегрує систему caffe з Apache Spark для розподіленого глибинного навчання.

У квітні 2017 року Facebook анонсувала Caffe2, що включає нові властивості, такі як рекурентні нейронні мережі.

2.5.6 Огляд фреймворку MXNet

MXNet [41] – це сучасна основа для відкритих джерел, яка використовується для навчання та розгортання глибоких нейронних мереж. Він масштабований, що дозволяє швидко тренувати модель і підтримує гнучке програмування та кілька мов (C ++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl, Wolfram Language) [22].

Бібліотека MXNet є портативною і може масштабуватися для декількох графічних процесорів та декількох машин. MXNet підтримується великими постачальниками публічних хмарних сховищ, включаючи AWS та Azure. Компанія Amazon вибрала MXNet як глибоку навчальну основу вибору на AWS. Наразі MXNet підтримує Intel, Dato, Baidu, Microsoft, Wolfram Research та науково-дослідні установи, такі як Карнегі Меллон, MIT, Університет Вашингтона та Університет науки і техніки Гонконгу.

Apache MXNet – це гнучка та ультра масштабована глибока система навчання, яка підтримує сучасні умови в глибоких моделях навчання, включаючи зчеплення нейронних мереж (CNN) та довгі короткострокові мережі пам'яті (LSTM).

MXNet призначений для розподілу в динамічній хмарній інфраструктурі, використовуючи сервер розподілених параметрів і може досягти майже лінійного масштабування з кількома CPU / GPU.

MXNet підтримує як імперативне, так і символічне програмування, що спрощує задачу для розробників, які використовують імперативне програмування для початку глибокого навчання. Це також полегшує

відстеження, налагодження, збереження контрольних точок, зміну гіперпараметрів, таких як швидкість навчання або раннє зупинення.

Підтримує C++ для оптимізованого бекенду, щоб максимально використати доступний графічний процесор або центральний процесор, а також Python, R, Scala, Julia, Perl, Matlab та Javascript для простого використання інтерфейсу для розробників.

Підтримує ефективне розгортання навченої моделі на низькотехнологічних пристроях для висновків, таких як мобільні пристрої, пристрої IoT (використовуючи AWS Greengrass), без сервера (використовуючи AWS Lambda) або контейнери. Ці низькотехнологічні середовища можуть мати слабший процесор або обмежену пам'ять (ОЗП), а також мати можливість використовувати моделі, які пройшли навчання в середовищі вищого рівня (наприклад, кластер на базі GPU).

2.5.7 Обгрунтування вибору фреймворку для глибинного навчання

Було обрано фреймворк TensorFlow від компанії Google за для задачі побудови нейронних мереж. Вибір базується на таких функціях [23]:

– відкритий вихідний код - це дозволяє завжди переглядати реалізацію алгоритму та змінювати його, якщо це необхідно;

– зручний API – зручний для користувачів API мови прогармування Python дозволяє швидко і легко досягти необхідних функціональних можливостей;

– можливість навчитися на графічних прискорювачах NVIDIA - реалізація підтримки відеоадаптерів є досить складною проблемою, але може значно пришвидшити роботу мережі;

– підтримка Google – один з найважливіших моментів. Оскільки GAN – досить новий виток розвитку машинного навчання, дуже важливо мати

своєчасну підтримку та оновлення, що можуть бути необхідними у разі винайдення нових алгоритмів, методів оптимізації, тощо;

– надає можливість використовувати фреймворк Keras в якості зручного інтерфейсу.

2.6 Існуючі моделі машиного навчання

2.6.1 Модель RNN

Звичайне збереження стану нейронної мережі досягається за допомогою рекурентних нейронних мереж. Люди не починають думати з чистого аркуша кожну секунду. Стара інформація не відкидається з голови, та роздум не починається з нуля. Наші думки мають сталість.

Традиційні нейронні мережі не володіють цією властивістю, і в цьому їх головний недолік. Уявімо, наприклад, що ми хочемо класифікувати події, що відбуваються у фільмі. Незрозуміло, як традиційна нейронна мережа могла б використовувати міркування про попередні події фільму, щоб отримати інформацію про подальші.

Вирішити цю проблеми допомагають рекурентні нейронні мережі (Recurrent Neural Networks, RNN) (рис. 2.7). Це мережі, що містять зворотні зв'язки і дозволяють зберігати інформацію [42].

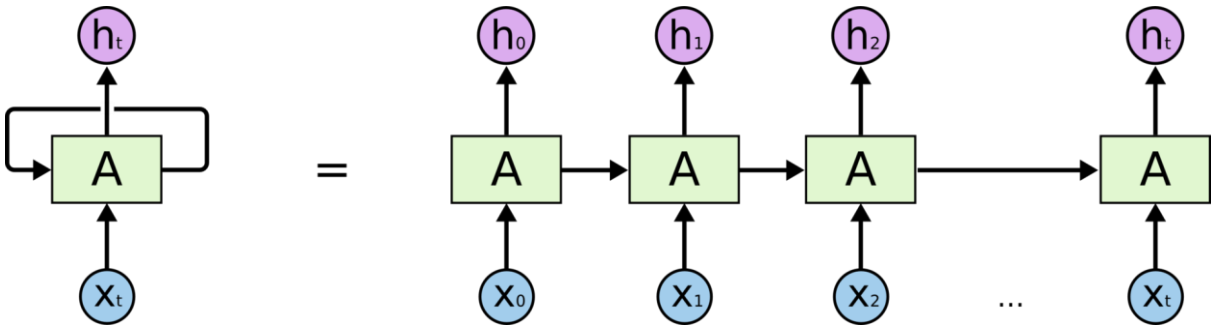

Рисунок 2.7 – Модель RNN

Однією з привабливих ідей моделі RNN є те, що вони потенційно можуть пов’язати попередню інформацію з поточним завданням, тому, наприклад, знання попереднього відеокадру може допомогти в розумінні поточного кадру. Якби моделі RNN мали таку можливість, вони були б надзвичайно корисними, але ця можливість залежить від обставин.

Іноді нам потрібна лише остання інформація для виконання поточного завдання. Розглянемо, наприклад, мовну модель, яка намагається передбачити наступне слово на основі попередніх. Якщо ми хочемо передбачити останнє слово у реченні «хмари пливуть у небі», нам не потрібен ширший контекст; в цьому випадку цілком очевидно, що останнім словом буде "небо". У цьому випадку, коли відстань між фактичною інформацією та місцем, де вона потрібна, невелика, моделі RNN можуть навчитися використовувати інформацію з минулого.

Рисунок 2.8 демонструє, як історія проходить через ланцюг вхідних нейронів.

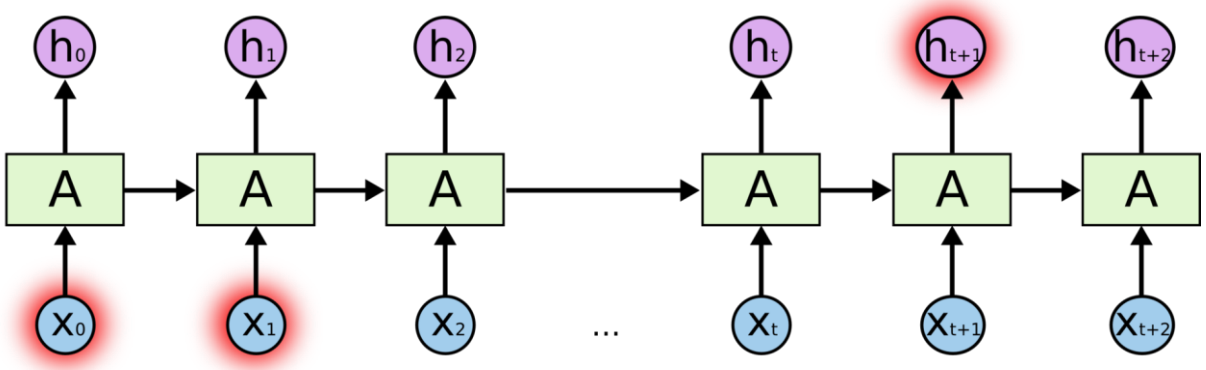

Рисунок 2.8 – Проблема довгострокових залежностей

Проблема довгострокових залежностей тут полягає у тому, что якщо цей ланцюг буде достатньо довгий, щоб дані про ті значення, що були на початку, “згасли”. Це може спричиняти проблеми в обробці текстів, великих та складних речень [43].

2.6.2 Модель LSTM

Довга короткострокова пам'ять (Long short-term memory; LSTM) [44] – особливий різновид архітектури рекурентних нейронних мереж (рис. 2.9), здатна до навчання довготривалим залежностям. Вони були представлені Зеппом Хохрайтер і Юргеном Шмідхубер (Jürgen Schmidhuber) в 1997 році, а потім вдосконалені і популярно викладені в роботах багатьох інших дослідників. Вони прекрасно вирішують цілий ряд різноманітних завдань і в даний час широко використовуються в провідних системах по всьому світу, для виконання перекладу, розробки штучних інтелектів та розпізнавання елементів на графічній площині.

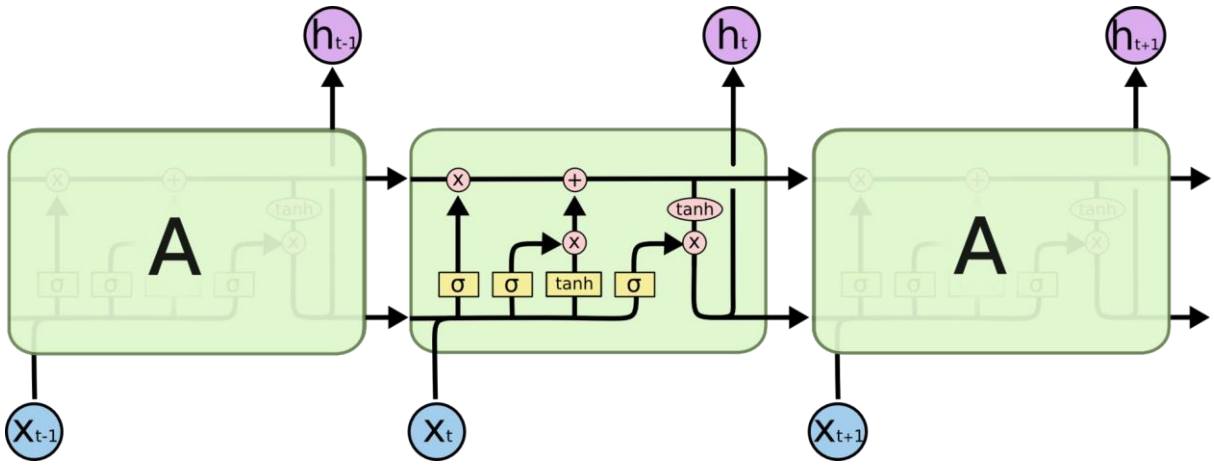

Рисунок 2.9 – Структура моделі LSTM

LSTM зчитує вхідні параметри у зворотному порядку, тому що це вводить багато короткострокових залежностей у даних, які значно спрощують проблему оптимізації. Ця, на перший погляд, проста дія вносить один з ключових технічних внесків у вирішенні проблеми аналізу довгих речень, яка є у моделях на основі RNN.

Глибокі нейронні мережі (DNN) – надзвичайно потужні моделі машинного навчання, які досягають чудової продуктивності у складних проблемах, таких як розпізнавання мовлення та візуальне розпізнавання

об’єктів. Моделі DNN є потужними, оскільки вони можуть виконувати довільні паралельні обчислення за скромну кількість кроків. Дивовижним прикладом потужності моделей DNN є їх здатність до сортування N-бітових чисел з використанням лише 2 прихованих шарів квадратичного розміру. Отже, поки нейронні мережі пов'язані зі звичайними статистичними моделями, вони вивчають складні обчислення. Крім того, великі моделі DNN можуть тренуватися з контрольованим зворотним розповсюдженням, коли мітці навчального набору вистачає інформації для визначення параметрів мережі. Таким чином, якщо існує параметр великого DNN, який досягає хороших результатів (наприклад, тому що людина може дуже швидко вирішити завдання), контрольоване зворотне розповсюдження знайде ці параметри і вирішить проблему.

Незважаючи на свою гнучкість та потужність, DNN можна застосовувати лише до проблем, вхідні дані та цільові показники може бути розумно закодований векторами фіксованої розмірності. Це суттєве обмеження, оскільки багато важливих проблем найкраще виражаються за допомогою послідовностей, довжини яких невідомо апріорі.

Наприклад, розпізнавання мови та машинний переклад – це послідовні проблеми. Аналогічно, відповідь на запитання також може розглядатися як відображення послідовності слів, що представляють питання на послідовність слів, що представляють відповідь.

Послідовності є викликом для DNN, оскільки вони вимагають, щоб розмірність входів і виходів були відомі і фіксовані. Ідея полягає у використанні одного сектора LSTM для зчитування послідовності введення, один крок за часом, для отримання великого фіксованого розмірного векторного подання, а потім використовувати інший сектор LSTM для отримання вихідної послідовності з цього вектора. Другий сектор LSTM є по суті рекуррентною мовною нейронною мережевою моделлю, за винятком того, що він обумовлений послідовністю введення. Здатність моделі LSTM успішно

вивчення даних із тимчасовими залежностями дальнього діапазону робить природним вибором для цього додатка через значне відставання між входами та відповідними їм виходами.

2.7 Генерація текстів методом Послідовність-до-послідовності

Рисунок 2.10 демонструє, як модель читає введення пропозиції "ABC" і видає "WXYZ" як вихідне речення. Модель перестає робити прогнози після виведення токену кінцевого речення [45].

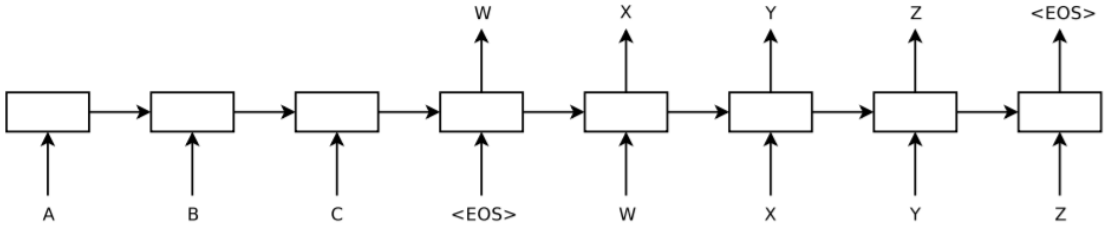

Рисунок 2.10 – Схема Sequence-to-Sequence моделі

В базовій моделі, що була описана вище, кожен вхід повинен бути закодований в вектор стану фіксованого розміру, так як це єдине, що передається декодеру. Щоб дати декодеру більш прямий доступ до даного введення, був представлений механізм уваги. Ми не будемо вдаватися в деталі механізму уваги (для цього можна ознайомитися з роботою по посиланням); досить сказати, що він дозволяє декодеру заглядати в дані введення на кожному кроці декодування. Багатошаровий сегмент послідовності до послідовності з LSTM з клітинами та механізмом уваги в декодері виглядає як на рисунку 2.11.

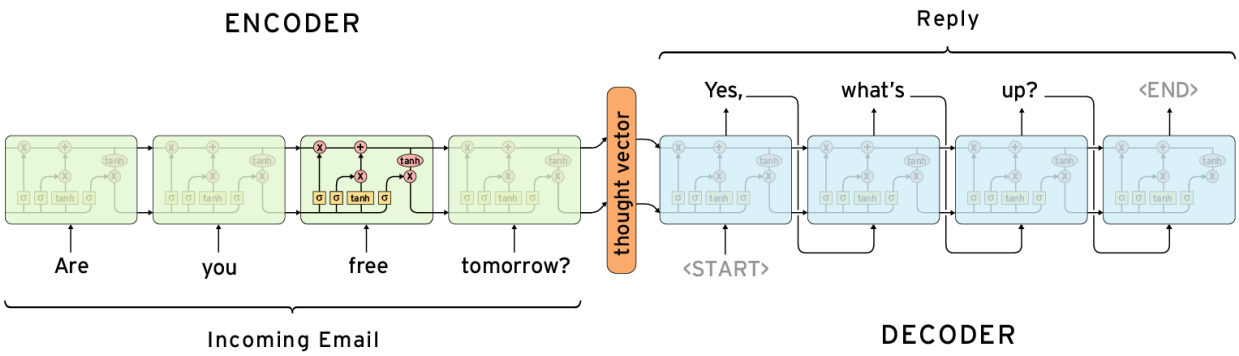

Рисунок 2.11 – Приклад генерації текстів методом Послідовність-до-послідовності з використанням моделі LSTM

Навчання моделей методом Послідовність-до-послідовності (далі – seq2seq) [46] мали великий успіх у вирішенні різних завдань, таких як машинний переклад, розпізнавання мови та підсумовування тексту. Моделі послідовності широко використовуються у системах нейронного машинного перекладу (NMT), який був першим успішним випробувальним набором моделей seq2seq.

У загальному випадку вхідні послідовності та вихідні послідовності мають різну довжину (наприклад, машинний переклад), і вся послідовність введення необхідна для того, щоб розпочати прогнозування цілі.

RNN-шар (або його стек) виступає в якості "кодера": він обробляє вхідну послідовність і повертає власний внутрішній стан. Зауважу, що ми відкидаємо виходи кодера RNN, лише відновлюючи стан. Цей стан буде служити як "контекст", або "кондиціонування", декодера на наступному кроці.

Інший RNN-шар (або його стек) виступає в якості "декодера": він навчається передбачати наступні символи цільової послідовності. Зокрема, він навчений перетворити цільові послідовності в ті ж самі послідовності, але компенсувати один раз у майбутньому, навчальний процес, який називається "вчитель витісняє" в цьому контексті. Важливо те, що кодер використовує в якості вихідного стану вектори стану з кодера, тобто декодер одержує інформацію про те, що він повинен генерувати. Декодер вчиться генерувати цілі [t + 1 ...] заданих цілей [... t], зумовлених послідовністю введення.

У режимі виведення, тобто коли ми хочемо розшифрувати невідомі вхідні послідовності, виконується такий процес:

а) кодування вхідної послідовності у векторний стан;

б) початок із цільової послідовності розміру 1 (лише символ початку послідовності);

в) подача векторного стану та 1-символьної послідовності цілей до декодера, щоб отримати прогноз для наступного символу;

г) отримується наступний символ, використовуючи ці передбачення;

д) обраний символ додається до цільової послідовності;

е) процес повторюється, доки не генерується символ кінця послідовності, або досягається межа символів.

Цей процес також може бути використаний для навчання мережі seq2seq шляхом повторення прогнозів декодера у декодері.

Моделі послідовності широко використовуються у системах нейронного машинного перекладу (NMT), який був першим випробувальним набором моделей seq2seq з великим успіхом.

Спочатку ми створимо деякі базові знання про моделі seq2seq для NMT, роз'яснюючи, як побудувати та тренувати модель NMT. У другій частині буде детально описано побудову конкурентоспроможної моделі NMT з механізмом уваги. Потім ми обговорюємо поради та підказки щодо створення найкращих моделей NMT (як у швидкості, так і в якості перекладу), таких як найкращі практики TensorFlow (дозування, bucketing), двонаправлені RNN, промінь пошуку, а також збільшення кількості декількох графічних процесорів, що використовують увагу GNMT.

Ще за старих часів традиційні фразові системи перекладу виконували своє завдання, розбивши вихідні речення на кілька фрагментів, а потім перекладали їх фразами. Це призвело до розбіжностей у викладах перекладу і не дуже схоже на те, як перекладають люди. Ми читаємо весь вихідний текст,

розуміємо його значення, а потім створюємо переклад. Нейронні машинні переклади (NMT) імітують це.

Зокрема, система NMT[47] вперше зчитує вихідне речення за допомогою кодувача для побудови вектора "думки", послідовності чисел, що представляють значення пропозиції; потім декодер обробляє векторе речення, щоб випромінювати переклад, як це показано на рисунку 2.12. Це часто називають архітектурою кодера-декодера. Таким чином, NMT вирішує проблему локального перекладу в традиційному фразовому підході: він може фіксувати довгострокові залежності мов, наприклад, гендерні угоди чи синтаксичні структури. Після цього створювати набагато більш плавні переклади, як показали Google Neural Machine Translation Systems.

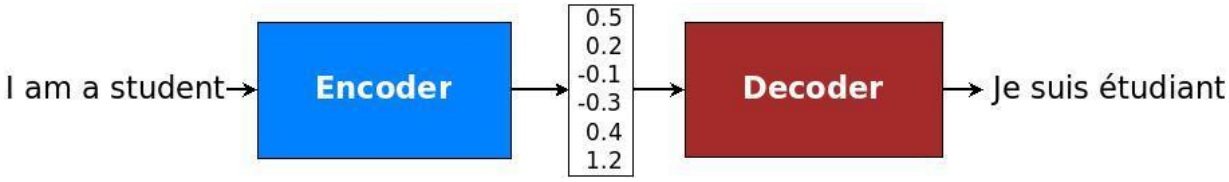

Рисунок 2.12 – Приклад моделі енкодера та декодера

NMT моделі варіюються з точки зору їх точної архітектури. Натурним вибором для послідовних даних є періодична нейронна мережа (RNN), що використовується більшістю моделей NMT. Зазвичай RNN використовується як для кодера, так і для декодера. Моделі RNN, однак, відрізняються за:

– спрямованістю – односпрямована або двонаправлена;

– глибиною – одно- або багатошарова;

– тип.

Розглядаємо як приклад глибокий багатошаровий RNN, який є однонаправленим і використовує LSTM як рекурентну одиницю. Ми показуємо приклад такої моделі на рисунку 2.12. У цьому прикладі ми створимо модель для перекладу вихідного речення "Я студент" у цільове речення "Je suis étudiant". На високому рівні модель НМТ складається з двох повторюваних

нейронних мереж: кодер RNN просто споживає слова джерела вхідного сигналу без будь-якого передбачення; декодер обробляє цільове речення під час прогнозування наступних слів.

2.8 Обробка природної мови

Обробка природної мови - загальний напрям інформатики, штучного інтелекту та математичної лінгвістики. Вивчає проблеми комп’ютерного аналізу та синтезу природної мови. Для штучного інтелекту аналіз означає розуміння мови, а синтез - генерування інтелектуального тексту. Вирішення цих проблем означатиме створення більш зручної форми взаємодії комп'ютера і людини.

Розуміння природної мови [47] іноді вважається завданням, AI-повним, оскільки для розпізнавання живої мови потрібна величезна система знань про навколишнє середовище та здатність взаємодіяти з ним. Визначення значення слова «розуміти» - одне з головних завдань штучного інтелекту. У процесі вивчення обробки природної мови були досягнуті значні результати, зокрема, розробка потужних лексикографічних систем, програм машинного перекладу, електронних словників та ін. Однак є проблема, яка ще не вирішена, вона корениться в самій природі людської мови. Проблема розуміння людської мови полягає саме в її неоднозначності. Можна виділити такі типи неоднозначностей:

– синтаксична неоднозначність: у прислів'ї «Час – не кінь, не підженеш і не зупиниш» буде абсолютно неясним для обробки природної мови те, про що саме йдеться у реченні, про коня чи про час;

– смислова неоднозначність: у питанні «Де знайти ключ до того замку?» слово замок може мати два абсолютно різні значення, якщо зважати на поставлений наголос;

– відмінкова неоднозначність: у фразах «Усі були схвильовані перед концертом» та «Не треба давати перед!» слово перед означає час або місце, що абсолютно змінює сенс фрази;

– референційна неоднозначність: у фразі «Відкрий поличку та дістань мокру парасольку, я хочу її висушити» займенник її за смисловим значенням матиме відношення до мокрої парасольки, проте для машини, у якої повністю відсутнє розуміння реальності, даний займенник відноситиметься як до полички, так і до парасольки.

Однією з проблем, що виникає в процесі обробки природної мови, можна вважати проблему синонімії, в результаті якої одне поняття може бути виражене кількома різними словами. Як результат, відповідні документи, що використовують синоніми до термінів, зазначених користувачем у запиті, можуть не бути визначені системою.

Вплив вищезазначених явищ особливо помітний при створенні систем машинного перекладу. Проблема полягає у складності встановлення конкретного відображення фактичної семантико-синтаксичної структури речення у його внутрішньому логічному поданні, яке автоматично генерується системою. Вирішення цих типів двозначностей можливо шляхом введення додаткових значень, які збільшать знання програми про певну галузь. На сьогоднішній день не існує програм, які "розуміють" усі типи неясностей у широкому діапазоні галузей, але є програми, які можуть належним чином реагувати на неясності у дуже вузьких областях.

Складність формального опису природньої мови і його обробки веде до розбиття цього процесу на окремі етапи, відповідні рівням мови. Більшість сучасних лінгвістичних процесорів відносяться до модульному типу, в якому кожному рівню / етапу аналізу або синтезу тексту відповідає окремий модуль процесора. У разі аналізу тексту окремі модулі лінгвістичного процесора виконують:

– графематическій аналіз (сегментація), тобто виділення в текстіпропозицій і словоформ, точніше токенов (т. к. в тексті можуть бути не тільки слова) – перехід від символів до слів;

– морфологічний аналіз – перехід від словоформ до їх лемм (словниковим формам лексем) або основам (ядерним частинах слова, за вирахуванням словозмінних морфем);

– синтаксичний аналіз – виявлення синтаксичних зв'язків слів і граматичної структури речень;

– автоматична обробка текстів і аналіз даних;

– семантичний і прагматичний аналіз, при якому визначається сенс фраз і відповідна реакція системи, в рамках якої працює лінгвістичний процесор.

Таким чином, лінгвістичний процесор можна розглядати як багатоетапний перетворювач, що переводить в разі аналізу тексту кожне його пропозиція у внутрішнє представлення його сенсу і навпаки в разі синтезу.

Можливі різні схеми об'єднання і взаємодії модулів розглянутих етапів, проте окремі рівні – морфологія, синтаксис і семантика зазвичай обробляються різними механізмами при вирішенні деяких прикладних задач можна обійтися без подання в процесорі всіх етапів / рівнів (наприклад, в ранніх експериментальних програмах оброблювані тексти ставилися до дуже вузьких проблемних областей з обмеженим набором слів, так що не був потрібен морфологічний і синтаксичний аналіз).

2.8.1 Обчислювальна лінгвистика

Мовознавство – це наукове вивчення мови, включаючи її граматику, семантику та фонетику. Класична лінгвістика передбачала розробку та оцінку правил мови. Був досягнутий великий прогрес про формальні методи

синтаксису та семантики, але здебільшого проблеми в розуміння природної мови протистоять чистим математичним формалізмам.

Загалом, лінгвіст – це будь-хто, хто вивчає мову, але, можливо, більш розмовно, мовознавець, що самовизначається, може бути більше зосереджений на тому, щоб бути поза місцем.

Обчислювальна лінгвістика[48] – це сучасне вивчення лінгвістики з використанням засобів інформатики. Оскільки використанням обчислювальних інструментів стало широко використовуватись в області мовознавства, може бути доречно називати цю область комп'ютерною лінгвістикою. Обчислювальна лінгвістика – це вивчення комп'ютерних систем для розуміння та породження природної мови. Однією з природних функцій для обчислювальної лінгвістики було б тестування граматик, запропонованих лінгвістами-теоретиками.

Великі дані та швидкі комп’ютери означають, що можна виявити нові та різні дані у великих наборах тексту за допомогою написання та запуску програмного забезпечення. У 1990-х роках статистичні методи та статистичне машинне навчання почало і врешті-решт замінило класичне правило, яке базується на підходах до мови принципом згори вниз, насамперед через їх кращі результати, швидкість та стійкість [32]. Зараз домінує статистичний підхід до вивчення природної мови.

Дані методи для обробки природних мов стали настільки популярними що їх слід вважати основними підходами до обчислювальної лінгвістики. Сильним фактором, що сприяє цьому розвитку, безперечно є зростання кількості доступних електронно збережених даних, до яких ці методи можуть бути застосовані.

Іншим фактором може бути певна незадоволеність виключно підходами за рукотворними правилами, завдяки спостереженій їх крихкості [30].

Статистичний підхід до природної мови не обмежується статистикою як такою, а й лише передові методи виведення, як ті, що застосовуються в прикладному машинному навчанні.

Для розуміння природної мови потрібні великі обсяги знань морфологія, синтаксис, семантика та прагматика, а також загальні знання про світ. Набуття та кодування всіх цих знань є одним із основних перешкод розвитку ефективних та надійних мовних систем. Як і в статистичних методах, методи машинного навчання автоматично отримують ці знання з анотованих або ненаголошених мовних корпусів.

2.8.2 Цілі глибинного навчання в обробці природньої мови

Методи глибокого навчання популярні насамперед тому, що вони виконують свої цілі. Деякі з перших великих демонстрації сили глибокого навчання пройшли в природній обробці мови, зокрема розпізнавання мовлення та машинному перекладі. Дослідниками та практикантами в області машинного навчання було виділено декілька ключових цілей:

– можливість заміни моделей, тобто методи глибинного навчання можуть використовуватись в існуючих системах обробки природньої мови, в якості заміни, та які будуть демонструвати співмірний рівень, або підвищення ефективності;

– нові моделі NLP, тобто, методи глибокого навчання пропонують можливість нових моделюючих підходів до складних проблем природної мови. Одним із важливих прикладів є використання періодичних нейронних мереж, які здатні вивчати та обумовлювати вихід на дуже довгих послідовностях;

– безперервне вдосконалення, тобто, ефективність глибинного навчання в обробці природньої мови ґрунтується на реальних результатах, і вдосконалення, тривають і, можливо, прискорюються;

– навчання особливостям, тобто, методи глибинного навчання можуть навчитися особливостям природньої мови, що вимагаються моделлю, а не вимагати, щоб характеристики були визначені та витягнуті експертом [32].

2.8.3 Морфологічний аналіз тексту

Однією з основ мови є його словник. Але що містить в собі словник? Так як термін «слово» є занадто багатозначним для того, щоб вживати його в строгому науковому тексті, введемо кілька понять.

Якщо з тексту витягти всі підрядка, що не містять роздільників (прогалин, деяких розділових знаків та ін.), то ми отримаємо безліч токенів. Наприклад, токеном будуть слова під'їзд або під (так як вони можуть зустрітися в тексті самі по собі), але не буде подстрока под (якщо вона, звичайно, не написана в тексті саме в такому вигляді).

Вважається, що для кожного токена існує його початкова (або нормальна) форма (також звана леммой). Від цієї початкової форми створюються всі інші форми слова шляхом флексії.

Автоматична обробка текстів і аналіз даних деяких змін цієї початкової форми. Утворення нових слів або їх форм відбувається на рівні комбінування морфов – мінімальних значущих одиниць мови. Морфи діляться на кореневі (корінь слова) і службові: префікс (приставка), суфікс, флексія (закінчення), постфікси. Носієм основного сенсу слова є корінь, а службові, в загальному випадку, надають додатковий сенс. Розбиття слова на морфи називається морфемним розбором.

Деякі службові морфи (наприклад, приставки та суфікси) відповідає за утворення нових слів, інші (наприклад, закінчення) – за освіту форм слів. Зміна форми слова прив'язується до набору граматичних параметрів (тегів): частина мови, рід, число, відмінок. Під словоформою ми будемо розуміти групу

(кортеж), що складається з токена, пов'язаної з ним початкової форми і безлічі граматичних параметрів.

Морфологічний аналіз форми по її токену. Завдання морфологічного синтезу прямо протилежна – по словоформі необхідно повернути її токен. Або більш формально:

– морфологічний аналіз – це отримання леми або основи (псевдооснови) заданого токена, а при необхідності, морфологічних параметрів;

– морфологічний синтез – це генерація потрібної словоформи слова або всієї його парадигми по нормальній формі (або основі) і морфологічним характеристикам.

Під слововживання ми будемо розуміти входження словоформи в текст. Залежно від контексту, під слововживання може розумітися або тільки рядок словоформи, або словоформа як безліч. Наприклад, фраза «Косий косою косив косою косою за піщаною косою» містить в собі 8 слововживань, 7 словоформ, 6 лексем і 4 унікальнихтокена. Це стає очевидним, якщо замість рядків записати словоформи.

<Косою, косою, {ім, чоловік. рід, од. ч., ім. п.}>

<Косою, косою, {ім, чоловік. рід, од. ч., ім. п.}>

<Косив, косити, {дієсл, 3 особа, од. ч., бавовняні. вр., чоловік. рід}>

<Косою, косою, {озн, жін. рід, од. ч., тв. п.}>

<Косою, коса, {ім, жін. рід, од. ч., тв. п.}>

<За, за, {пропоз}>

<Піщаної, піщаний, {дод, жін. рід, од. ч., тв. п.}>

<Косою, косою, {сущ, жін. рід, од. ч., тв. п.}>

Тут ми неявно використовували запис граматичних параметрів, що припускає роздільне написання імені та значення параметра. Іменем параметра може служити рід, число, час, схиляння, стисліть форми прикметника і інші ознаки слів, прийняті в даній мові. Значення параметра – це конкретне

значення, яке може приймати дана ознака. Так, наприклад, відмінок може бути називним, родовим, місцевим; рід може бути чоловічим, жіночим, середнім; число – єдиним, множинним, подвійним і т.д.

2.8.4 Фреймворки для аналізу природньої мови

Natural Language Toolkit(NLTK)[49] – це всебічна бібліотека Python для обробки природних мов та аналізу тексту. Спочатку бібліотека NLTK була призначена для викладання, але була прийнята у галузі для досліджень та розробок завдяки своїй корисності та широті охоплення. NLTK часто використовується для швидкого прототипування програм обробки тексту і навіть може використовуватися у виробничих додатках. Ця бібліотека забезпечує прості у використанні інтерфейси для більш ніж 50 корпоративних і лексичних ресурсів, таких як WordNet, а також набір текстових бібліотек для класифікації, токенізації (рис. 2.13), виведення, тегів, розбору та семантичних міркувань, обгортки для промислових бібліотек NLP, і активний дискусійний форум.

Завдяки практичному керівництву, що представляє основи програмування поряд з темами обчислювальної лінгвістики, а також повну документацію API, NLTK підходить для лінгвістів, інженерів, студентів, педагогів, дослідників та користувачів галузі. NLTK доступний для Windows, Mac OS і Linux. Найкраще, що NLTK – це безкоштовний, відкритий, спільно керований проект.

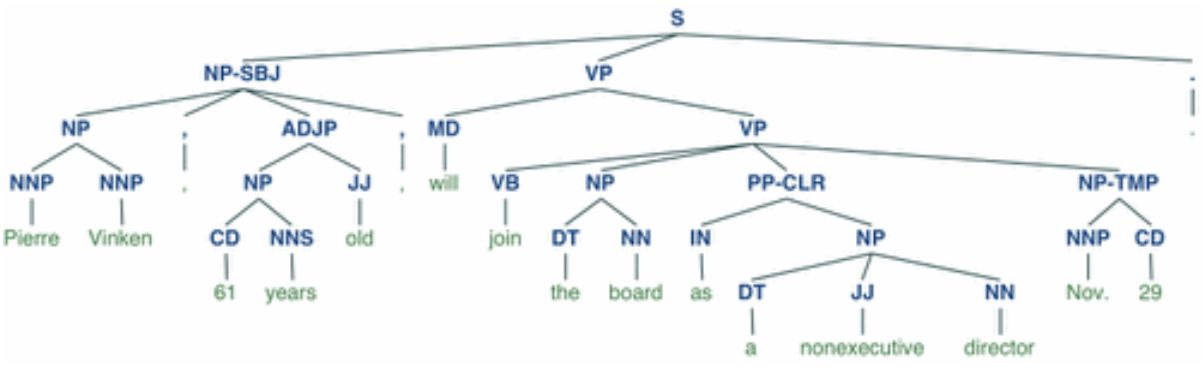

Рисунок 2.13 – Приклад роботи бібліотеки NLTK

spaCy (рис.2.14) – бібліотека програмного забезпечення з відкритим кодом для розширеної обробки природних мов, написана мовами програмування Python та Cython. Бібліотека видається за ліцензією MIT і в даний час пропонує статистичні моделі нейронної мережі для англійської, німецької, іспанської, португальської, французької, італійської, голландської та багатомовної NER, а також токенізація для різних інших мов.

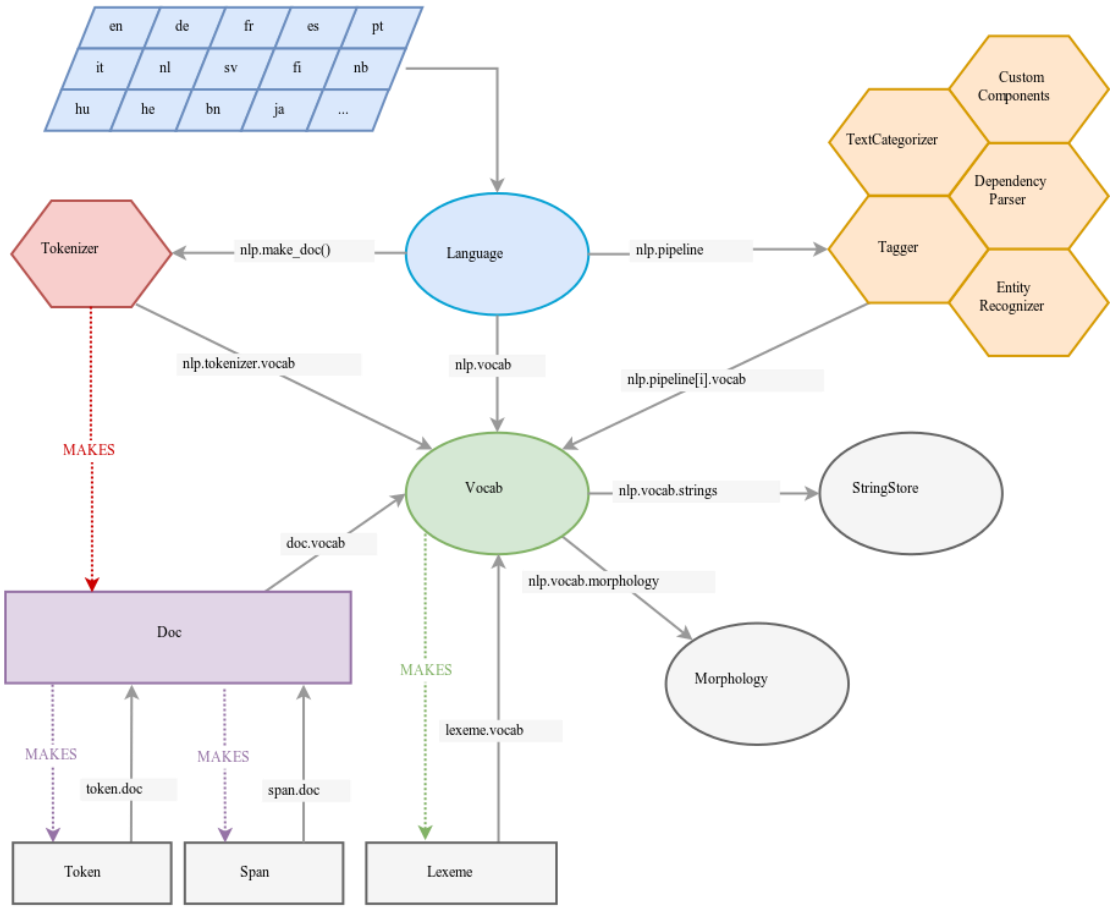

Рисунок 2.14 – Архітектура бібліотеки spaCy

Ключові функції бібліотеки:

– позначення часток мови;

– морфологія на основі правил;

– розбір залежностей;

– визнання іменованої одиниці;

– токенізація.

На відміну від NLTK, який широко використовується для викладання та досліджень, spaCy зосереджується на наданні програмного забезпечення для виробничого використання [50]. Станом на версію 1.0, spaCy також підтримує поглиблені робочі процеси , які дозволяють підключати статистичні моделі, що навчаються популярними бібліотеками машинного навчання, такими як TensorFlow, Keras, Scikit-learn або PyTorch. Машинна бібліотека машинного навчання spaCy, Thinc, також доступна як окрема бібліотека Python з відкритим кодом. У ньому представлені згорткові моделі нейронної мережі для тегування часткової мови, розбору залежностей та розпізнавання названих об'єктів, а також удосконалення API навколо моделей навчання, оновлення та побудови індивідуальних конвеєрів обробки.

Apache OpenNLP – це бібліотека Java з відкритим кодом, яка використовується для обробки текстів на природній мові. OpenNLP надає такі сервіси, як токенізація, сегментація пропозицій, теги частин мови, вилучення названих об'єктів, розподіл на частини, розбір та роздільна здатність довідки тощо.

Stanford CoreNLP – бібліотека, розроблена Stanford NLP Group на мові програмування Java, що надає набір інструментів та технологій для аналізу людської мови. Stanford CoreNLP надає NLP з методами статичного аналізу, NLP з методами глибинного навчання, інструменти обробки природньої мови на основі правил для основних проблем обчислювальної лінгвістики, які можна включити в додатки з потребами технологій обробки людської мови. Бібліотека може дати основні форми слів, їх частини мови, будь то назви компаній, людей тощо, нормалізувати дати, час та числові величини, відмітити структуру речень з точки зору словосполучень та синтаксичних залежностей, вказати які іменникові фрази стосуються одних і тих же утворень, вказують на настрої,

витягують конкретні або відкриті відносини між згадками суб'єкта, отримують цитати, про які говорять люди, тощо.

2.8.5 Обгрунтування вибору бібліотеки для аналізу природної мови

Для аналізу тексту було обрано бібліотеку spaCy. Вибір було зроблено на основі наступних особливостей [50]:

– відкритий початковий код – це дозволяє завжди подивитися реалізацію того чи іншого алгоритму і за необхідності внести до нього зміни;

– зручний API – звичний для користувача мови програмування Python API дозволяє швидко і без зайвих проблем використовувати необхідну функціональність;

– бібліотека дозволяє виконувати аналіз тексту великої кількості мов.

2.8.6 Дерево залежностей

У синтаксичному запиті частина мовлення та морфологічна інформація повертаються в поле partOfSpeech відповіді.

Для кожного речення в тексті, наданому API натуральної мови для синтаксичного аналізу, API будує дерево залежності, яке описує синтаксичну структуру цього речення. Після цього синтаксична інформація повертається в поле dependencyEdge відповіді.

Діаграма дерева залежності цього єдиного речення з інавгураційного виступу Джона Ф. Кеннеді відображається на рис. 2.15.

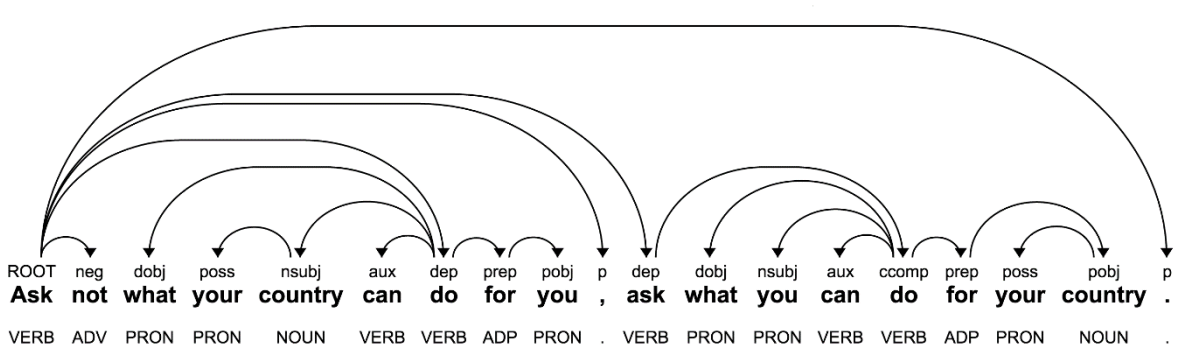

Рисунок 2.15 – Дерево залежностей

Для кожного маркера елемент dependencyEdge визначає, який інший маркер він модифікує (у полі headTokenIndex) та синтаксичний зв’язок між цим маркером та його заголовком (у полі мітки). Наприклад, ось елемент dependencyEdge для лексеми "ваш" в (першому виникненні) фразі "ваша країна":

```
 "dependencyEdge": {
    "headTokenIndex": 4,
    "label": "POSS"
},
```

Цей елемент вказує на те, що "ваш" модифікує п'ятий маркер (headTokenIndex використовує зсув на основі нуля) і що він є потужним модифікатором.

Кожне дерево залежності (рис. 2.15) включає елемент ROOT ("label": ROOT), що відповідає головному дієслову в реченні. У наведеному вище прикладі елемент ROOT є першим словом у реченні ("headTokenIndex": 0). Для ROOT слова "Ask", headTokenIndex – це власний індекс.

API Natural Language мітить синтаксичні зв’язки, використовуючи загальний набір залежностей, що застосовуються для підтримуваних мов. Означення залежностей в реченні наведено у таблиці 2.2. У прикладі текст "Голова" та мітка відображаються під маркерами, до яких вони застосовуються [51].

Таблиця 2.2 – Означення залежностей в реченні

Назва	Означення
1	2
ABBREV	Скорочення лексеми голови.
ACOMP	Прикметник словосполучення, яке функціонує як доповнення (як предмет дієслова). Це відношення зокрема включає конструкції копули `be` з прикметниковими предикатами.
ADVCL	Прислівник, що модифікує дієслово, наприклад, тимчасове застереження, наслідок, умовне застереження або пункт призначення.
ADVMOD	Прислівник (невідмовний) або дієприслівник, який слугує для зміни значення слова.
AMOD	Прикметникова фраза, яка служить для зміни значення іменникової фрази.
APPOS	Іменникова фраза, що знаходиться праворуч від іншої іменникової фрази, причому друга фраза служить для визначення або зміни першої.
ATTR	Іменна фраза на чолі з спільнокореневим дієсловом. Зауважте, що `` ATTR`` відрізняється від `` ACOMP`` тим, що залежний – це іменникова фраза, а не прикметник. Підвищенні конструкцій з номінальними предикатами також використовує відношення `` ATTR ".
AUX	Не головне дієслово, наприклад модальний допоміжний засіб або форма `` be``, `` do`` або `` have`` в перифрастичному часі. Виключає використання `` be`` як допоміжного засобу в пасивній конструкції.

Продовження таблиці 2.2

1	2
AUXPASS	Неосновне дієслово речення в пасивному голосі.
CC	Співвідношення між елементом сполучника та координаційним сполучником. Один зв`язок сполучника (зазвичай перший) розглядається як голова сполучника.
CCOMP	Залежне словосполучення із внутрішнім предметом, яке функціонує як об’єкт дієслова чи прикметника ..
CONJ	Співвідношення між двома елементами, з'єднаними координуючою сполукою, такими як `` і '' або `` або``. Голова відношення є першим сполучником, а інші сполучники залежать від нього через відношення ``conj ''.
CSUBJ	Прислівний синтаксичний предмет пункту; тобто тема сама по собі є пунктом ("Що вона сказала" у наведеному нижче прикладі).
CSUBJPASS	Ознайомчий синтаксичний предмет пасивного зауваження.
DEP	Система не в змозі визначити більш точне відношення залежності між двома словами.
DET	Співвідношення між головою іменникової фрази та її визначником.
DISCOURSE	Заперечення та інші елементи дискурсу, які чітко не пов’язані зі структурою речення, за винятком експресивного способу. Прикладами є перекрестки (`` о '', `` у-у-у '', `` привітання ''), наповнювачі (`` 'гм' ', `` а-а' '), і дискурсові маркери (`` добре '', `` як '', `` фактично '', але не `` ти знаєш '').

Продовження таблиці 2.2

1	2
DOBJ	Іменникова фраза, яка є звинувальним предметом дієслова.
EXPL	Плеонастичний номінал. В англійській мові це кілька застосувань `` it`` та `` there``: екзистенціальне `` there`` та `` it`` при використанні в екстракційних конструкціях. Експлікативний або плеонастичний іменник – це той, де номінал не задовольняє смислової ролі присудка. У мовах з експлікативом вони можуть бути розміщені в тематичних і прямих слотах об'єктів.
IOBJ	Іменникова фраза, що є дативним непрямим об’єктом дієслова.
MARK	Слово, яке вводить кінцеве або некінцеве підрядне підрозділ, наприклад, `` що " або `` чи ". Голова – голова підрядного пункту.
MWE	Одне з двох відносин (поряд із `` NN ") для складання. Він використовується для певних фіксованих граматизованих виразів із функціональними словами, які ведуть себе як одне функціональне слово. Вирази з декількома словами позначаються у плоскій, початковій структурі, в якій усі слова у виразі змінюють перше, використовуючи мітку `` MWE ".
MWV	Багатослівне словесне вираження.
NEG	Співвідношення між заперечувальним словом та словом, яке воно змінює.
NN	Будь-який іменник, який служить для зміни головного іменника.

Продовження таблиці 2.2

1	2
NPADVMOD	Іменникова фраза, що використовується як дієприслівник.
NSUBJ	Іменникова фраза, що є синтаксичним предметом пропозиції.
NSUBJPASS	Іменникова фраза, що є синтаксичним предметом пасивного сказа.
NUM	Будь-яка числова фраза, яка служить для зміни значення іменника на кількість.
NUMBER	Частина словосполучення.
P	Будь-який відрізок пунктуації в реченні.
PARATAXIS	Відношення паратаксису (з грецької для «місце пліч-о-пліч») – це відношення між словом (найчастіше основним присудком речення) та іншими елементами, розміщеними поруч, без явної координації, підпорядкування чи аргументаційного зв’язку з головним словом . Паратаксис – це дискурсовий еквівалент координації.
PARTMOD	Частифікаційний модифікатор
PCOMP	Використовується, коли доповнення прийменника є словосполученням або прийменниковим словосполученням (або іноді прислівниковою фразою).
POBJ	Голова іменникового словосполучення після прийменника або прислівників `` тут '' і `` там ''.
POSS	Присвійний визначник або генітив
POSTNEG	Поствербальна негативна частка
PRECOMP	Предикатний доповнення

Продовження таблиці 2.2

1	2
PRECONJ	Слово, яке з’являється на початку, що дуже в поєднанні, наприклад, `` або '', `` обидва '', `` ні ''.
PREDET	Слово, яке передує та змінює значення визначника іменника фрази.
PREF	Префікс
PREP	Будь-яка прийменникова фраза, яка служить для зміни значення дієслова, прикметника, іменника чи навіть іншого прийменника.
PRT	Частинка дієслова.
PS	Асоціативний або присвійний маркер
QUANTMOD	Модифікатор фразової кількості
RCMOD	Посилання від іменника до дієслова, яке очолює відносну пропозицію.
ROOT	Корінь речення. У переважній більшості випадків це дієслово.
SUFF	Суфікс
TMOD	Голий іменник складової фрази, який служить для зміни значення складової, задаючи час. `` TMOD`` фіксує тимчасові точки та тривалість; він не фіксує повторення (`` два рази '', що було б `` NPADVMOD''``).
VMOD	Прислівник на чолі з нескінченною формою дієслова.
VOCATIVE	Позначає учасника діалогу, адресованого в тексті (поширений в електронних листах та публікаціях групи новин).
XCOMP	Додаткове доповнення без власного предмета, посилання якого визначається зовнішнім суб'єктом.

Продовження таблиці 2.2

1	2
SUFFIX	Суфікс імені
TITLE	Назва імені
FOREIGN	Іноземні слова
KW	Ключове слово
LIST	Список ланцюгів порівнянних предметів
NOMC	Номіналізована стаття
NOMCSUBJ	Номіналізований предмет застереження
NOMCSUBJPASS	Номінований пункт пасивний
ASP	Маркер аспекту
GMOD	Генітивний модифікатор
GOBJ	Генітивний об'єкт
INFMOD	Модифікатор нескінченності
MES	Міра
NCOMP	Іменне доповнення іменника

2.8.7 Інструменти розробки

Більшість інструментів обробки природньої мови написані на Java та мають подібні функціональні можливості. Деякі з них є надійними та мають велику різноманітність інструментів NLP. Однак, коли мова йде про простоту використання та пояснення понять, найкраще себе показує бібліотека SpaCy, що використовує мову розробки Python.

Для реалізації системи штучного інтелекту також була обрана мова Python, через велику кількість інструментів для побутови систем на основі машиного навчання та аналізу природньої мови [52].

Python (рис. 2.16) – інтерпретована об'єктно-орієнтована мова програмування високого рівня з строгою динамічною типізацією. Розроблена в

1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання існуючих компонентів. Всі об'єкти поділяються на змінні та незмінні: списки, словники та набори є змінними, а всі інші незмінні

```
8   from keras.layers import Dense
9   import logging
10  logging.getLogger('tensorflow').disabled = True
11
12  if __name__ == "__main__":
13
14      # Read in wine data
15      white = pd.read_csv("http://archive.ics.uci.edu/ml/
16      white['type'] = 0
17      red = pd.read_csv("http://archive.ics.uci.edu/ml/ma
18      red['type'] = 1
19      wines = pd.concat([red, white], ignore_index=True)
20
21      # specify the target labels and flatten the array
22      X = wines.iloc[:, 0:12]
23      y = np.ravel(wines.type)
```

Рисунок 2.16 – Приклад коду на мові програмування Python

Python підтримує модулі та модульні пакети, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у компільованому, так і у вихідному вигляді на всіх основних платформах. Мова програмування Python підтримує кілька парадигм програмування, включаючи об'єктно-орієнтовану, процедурну, функціональну та аспектно-орієнтовану.

З 1991 року Python є цілком об'єктно-орієнтованим. Python також запозичив багато рис таких мов, як C, C++, Modula-3 й окремі риси функціонального програмування з Ліспу.

Наявність доброзичливої спільноти користувачів, поряд з дизайнерською інтуїцією Гвідо, вважається одним з головних факторів успіху Python. Розвиток мови відбувається відповідно до чітко регламентованих процесів створення, обговорення, відбору та реалізації PEP (пропозицій щодо вдосконалення Python) - пропозицій щодо розробки Python.

3 грудня 2008 року після тривалого тестування була випущена перша версія Python 3000 (або Python 3.0, також скорочена Py3k). Python 3000 усуває багато недоліків архітектури з максимально можливою (але не повною) сумісністю зі старими версіями. Сьогодні підтримуються обидві сфери розвитку (Python 3.6 та 2.7).

Python є портованим і працює майже на всіх відомих платформах – від КПК до мейнфреймів. Існують порти під Microsoft Windows, всі варіанти UNIX (включаючи FreeBSD та GNU/Linux), Plan 9, Mac OS та Mac OS X, iPhone OS 2.0 і вище, Palm OS, OS/2, Amiga, AS/400 та навіть OS/390[en], Symbian та Android.

У той же час, на відміну від багатьох перенесених систем, для всіх основних платформ Python має підтримку технологій, характерних для цієї платформи (наприклад, Microsoft COM / DCOM). Більше того, існує спеціальна версія Python для віртуальної машини Java - Jython, яка дозволяє інтерпретатору працювати в будь-якій системі, що підтримує Java, тоді як класи Java можна використовувати безпосередньо з Python і навіть писати на нього. Кілька проектів також забезпечують інтеграцію з платформою Microsoft.NET, основними з яких є IronPython та Python.Net [53].

2.9 Висновки з розділу 2

В розділі було проведено аналіз найпоширеніших фреймворків машинного навчання. Для реалізації поставленої задачі було обрано Keras.

Також було проведено аналіз мов програмування, які можна використовувати для побудови чат-ботів. За результатами порівнянь було обрано Python.

3 ПРОЕКТ ПРОГРАМНОЇ СИСТЕМИ КОНТЕКСТУ НЕЙРОННОЇ МЕРЕЖІ ТА ЧАТ-БОТА

3.1 Функціональні вимоги

Для кожного користувача має бути можливість додати чат-бота у контакти мессенжеру Telegram, після чого він може виконувати команди чат-боту: додати дані до контексту та відповісти на основі існуючих даних.

У останній версії програми користувач матиме можливість за допомогою інтерфейсу чат-боту додати дані до контексту застосунку, використовучи як прості, так і складні речення. Застосунок повинен задовільняти наступним умовам для системи: додати дані до контексту, отримати відповідь з використанням даних з контексту на запит.

3.1.1 Вимоги до інтерфейсу

Усі вище зазначені функції повинні бути доступними з інтерфейсу користувача, схему роботи якого зображено на рисунку 3.1.

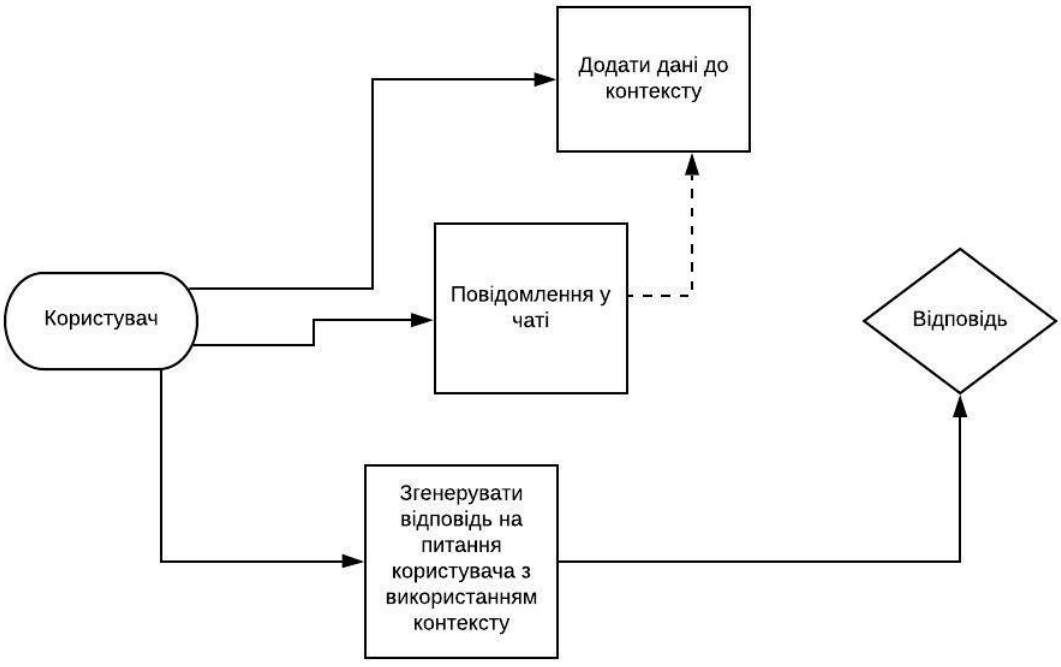

Рисунок 3.1 – Діаграма послідовностей

Основні вимоги до чат-боту:

- зручність у використанні;
- зрозумілість можливостей та інформативність відповідей чат-боту;
- можливість роботи чат-боту на різних платформах та пристроях;
- підтримка англійської мови для підтримки діалогу з чат-ботом.

3.1.2 Вимоги до надійності

Застосунок повинен бути стійким до дій непередбачених основною функціональністю.

На стороні клієнта повинна відбуватись валідація введених даних.

На стороні контексту повинна перевірятись коректність усіх відправлених даних.

Аналізатор природньої мови повинна підтримувати різні формати вхідних речень.

3.1.3 Вимоги до написання початкового коду

Основними вимогами до написання коду є:

- дотримуватись єдиного стилю оформлення;
- однозначне іменування змінних;
- документація коду;
- уникнення дублювання коду;
- розділення на окремі значущі частини
- використання ефективних структур даних.

3.1.4 Вимоги до апаратного та програмного забезпечення

Для ефективної роботи алгоритмів навчання та оптимізації витраченного часу буде потрібна досить потужна машина, яка буде мати змогу виконати навчання системи на подалі опрацьовувати інформацію.

Для розгортання та підтримки системи необхідні:

– сервер для розгортання клієнтської частини (Linux) з оперативною пам’яттю не менше 256Мб;

– якщо навчання буде виконуватись на потужностях CPU, мінімальна комплектація має бути Intel Core i5-4670, чи хмарний варіант Intel Core Xeon;

– інстальована середа Python з менеджером пакетів pip;

– підтримка навчання нейроної мережі з використанням Tensorflow та/або пакету tensorflow-gpu;

– вхідни дані для початкового контексту застосунку та пари питання-відповідь;

– якщо навчання буде виконуватись на потужностях графічного процесора, то відеокарта, яка буде задовольняти потребам – це GTX1060 Ti.

Для розгортання та навчання нейронної мережі в системі повинні бути інстальовані наступні pip пакети:

– numpy==1.16.4 (бібліотека для роботи з обчисленнями та масивами даних)

– pep8==1.7.1. (інструмент для статичного аналізу коду);

– spacy==2.2. (фреймворк для аналізу природньої мови);

– tensorflow==1.14.0 (фреймворк для машинного навчання);

– tensorflow-gpu==1.14.0 (додаток до пакету tensorflow, що дозволяє навчання на графічному процесорі);

– pandas==0.25.1 (бібліотека для роботи з масивами даних);

– keras==2.2.4 (фреймворк для машинного навчання);

– scipy==1.3.1 (бібліотека для виконання наукових обчислювальних операцій);

– word2vec==0.9.4 (бібліотека для трансформації слів в числовий формат).

3.2 Архітектура системи

Програмна система чат-боту буде складатися з модуля чат-боту та модуля, що оброблює дані (рис 3.2).

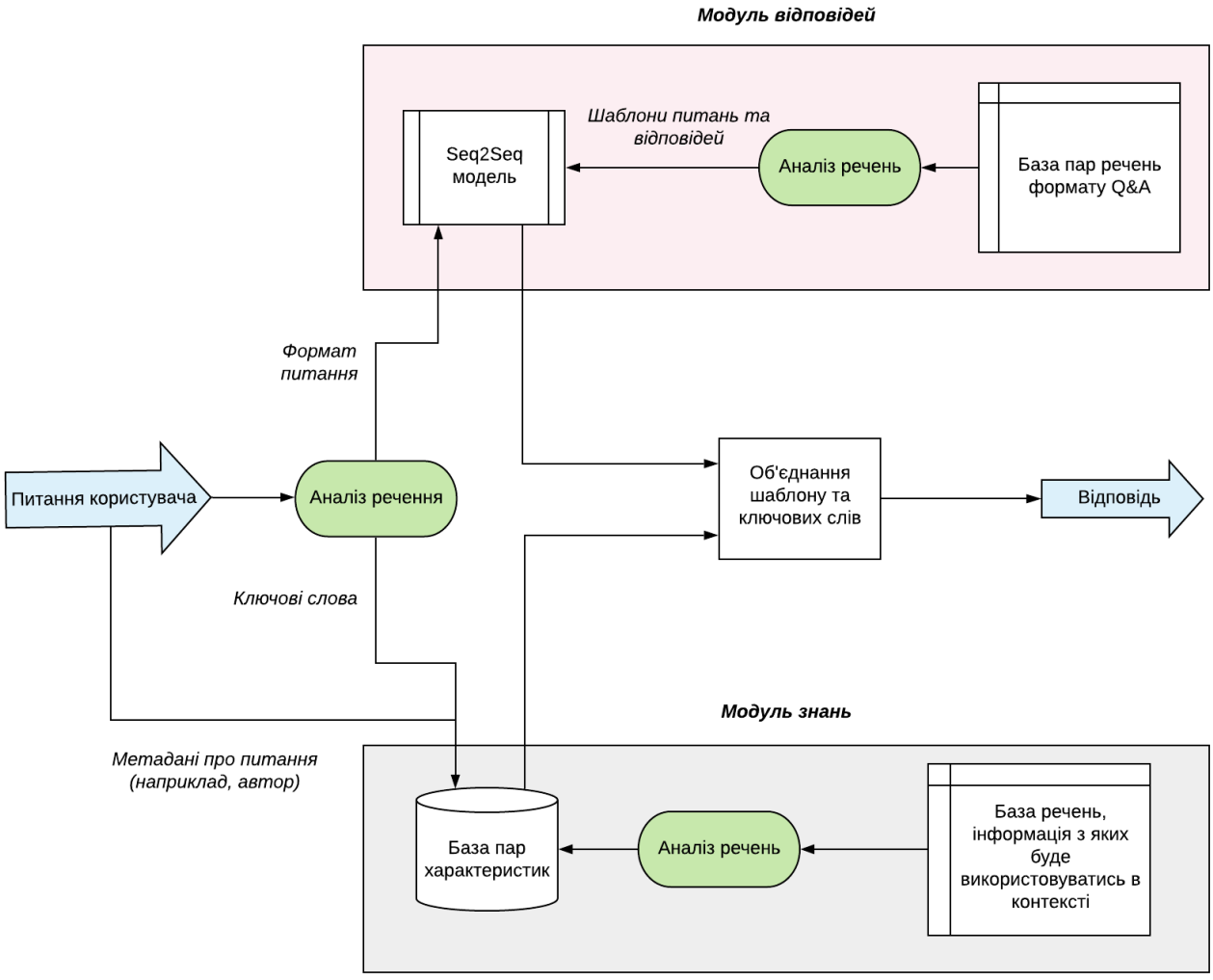

Рисунок 3.2 – Архітектура застосунку

3.3 Специфікація даних

В системі виділяється 4 базові сутності:

– елемент контексту (ContextNode) – базова сутність, з яких складається набір контексту;

– модель послідовніть-до-послідовності (Seq2Seq) та Seq2SeqNeuralService – генерує модель Seq2Seq на основі пар речень;

– CoreService – сервіс, що дає інтерфейс для роботи з контекстом та аналізу речень.

3.4 Висновки з розділу 3

В даному розділі було проаналізовано функціональні вимоги яким потрібен відповідати розроблений застосунок. Також розглянуто вимоги до надійності, апаратного та програмного забезпечення. Приведено приклад архітектури системи.

4 РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ КОНТЕКСТУ НЕЙРОННОЇ МЕРЕЖІ ТА ЧАТ-БОТА

4.1 Реалізація клієнтської частини

Клієнтська частина реалізована на базі Python бібліотеки python-telegram-bot. python-telegram-bot [54] – це бібліотека Python, що надає простий інтерфейс для розробки чат-ботів для платформи Telegram.

4.1.1 Архітектура клієнтської частини

Чат-бот був побудований з використанням реактивного додатку до бібліотеки, що дозволяє користуватись командами боту за допомогою декоратора @command (Лістинг 2). Бот містить декілька команд для роботи з ним:

/smart – генерувати відповідь за допомогою контексту

/nlp – тестувати функціонал аналізу речення

/ctx – додати елементи до контексту, або зчитати з початкового набору

Лістинг 1. Реєстрація обробника команди /smart

```
@command('smart')
def smart(bot, ctx):
    message = ctx.message.text.replace('/smart ', '').strip()
    resp = core.process_response(message)
    ctx.message.reply_text(resp)
```

Лістинг 2. Функціонал реєстрації нового обробника команди

```
def command(name):
    print('Command  '+name)
    def wrapper(fn):
```

```
bot().dispatcher.add_handler(CommandHandler(name, fn))
```

Приклад роботи бота наведено на рисунку 4.1.

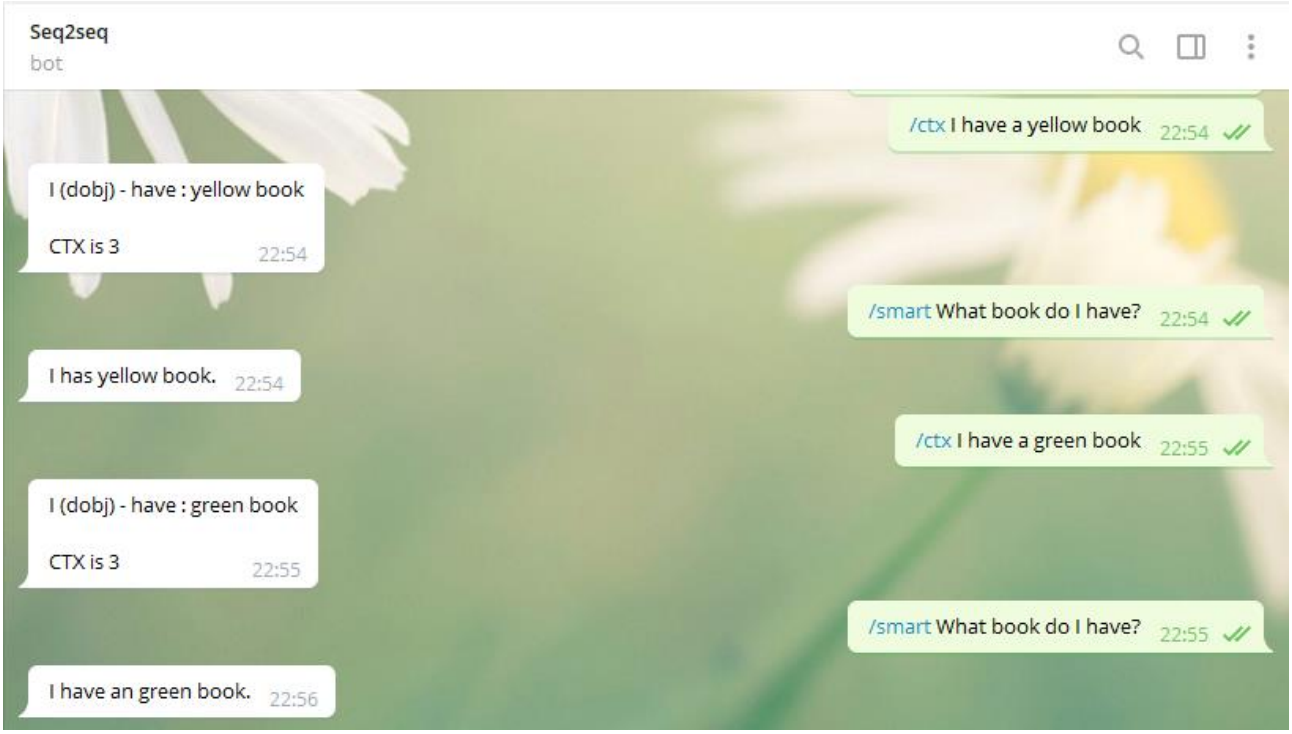

Рисунок 4.1 – Приклад роботи бота

Декоратор дозволяє спростити створення обробників команд.

Лістинг 3. Команда /ctx

```
from client.core.bot import command
from services import core, context
from services.core import learn_from_preset

@command('ctx')
def nani(_, ctx):
    if ctx.message.text == '/ctx':
        learn_from_preset()
    else:
```

```
core.process_to_context(ctx.message.text.replace('/ctx',
'').strip())
        ctx.message.reply_text('CTX is ' +
str(len(core.context.nodes)))
```

4.1.2 Опис функціональних можливостей та інтерфейсу

Для того щоб користувач міг зайти користуватись чат-ботом, перш за все йому потрібно активувати бота за допомогою команди */start*. Після того як користувач авторизувався у чат-боті, користувач може додавати дані в контекст застосунку (рис. 4.2), та генерувати відповіді за допомогою команди */smart*.

Рисунок 4.2 – Приклад реєстрації та команд до бота

Також є можливість переглянути контекст який в даний час наявний у чат-бота (рис. 4.3) використовуючи команду */list_ctx*.

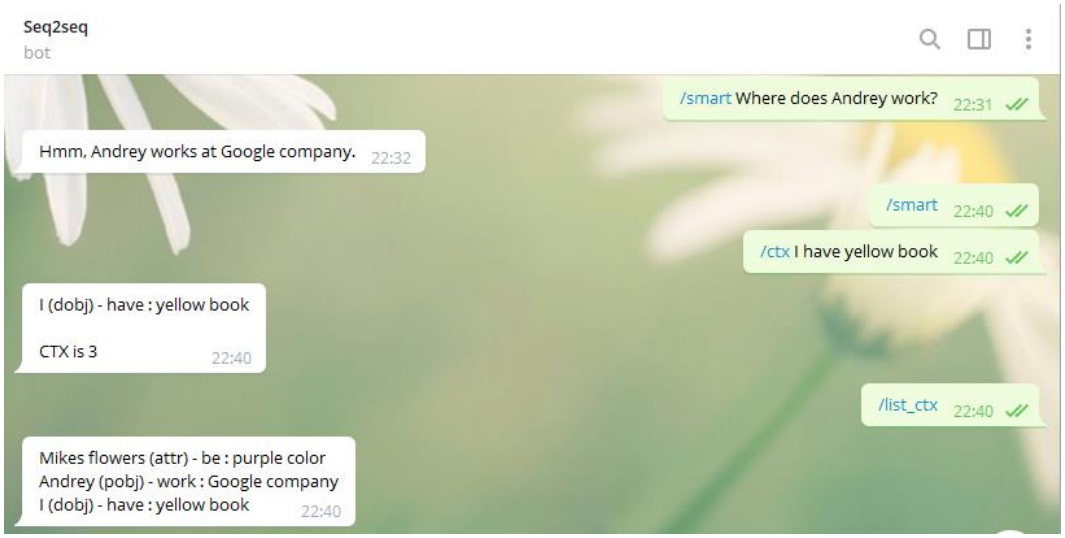

Рисунок 4.3 – Приклад перегляду контексту

4.2 Реалізація контексту та аналізу речень

Структуру проекту було розбито на декілька логічних частин (рис. 4.4):

– assets – текстові файли, що використовуються під час навчання;
– client – функціонтал та ядро клієнту чат-боту;
– networks – моделі нейронних мереж;
– services – сервіси, що використовуюється у клієнтській частині;
– utils – допоміжні інструменти, наприклад, читання рядків з файлу.

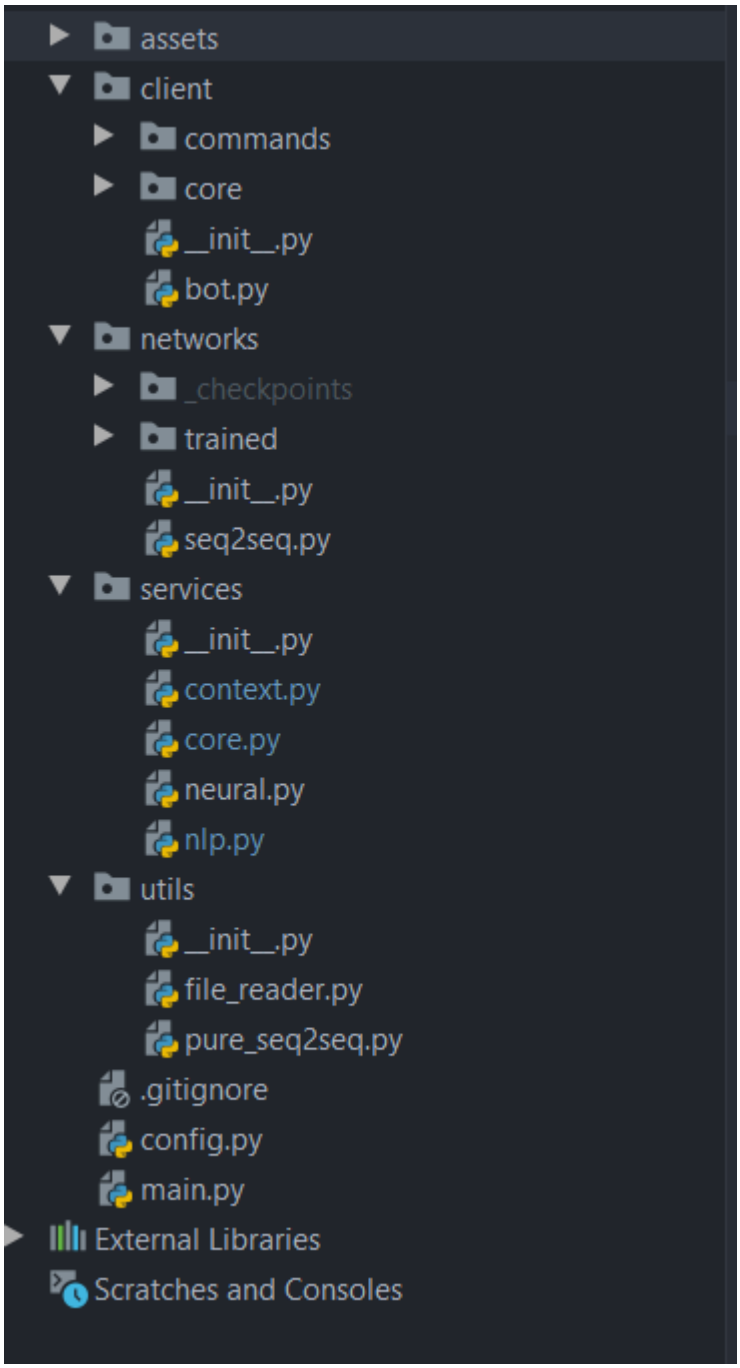

Рисунок 4.4 – Структура проекту

4.2.1 Реалізація частини контексту

Контекст застосунку складається з масиву елементів контексту, та функцій маніпуляцій над контекстом. Зокрема, функції:

– find_or_create_node – знайти елемент в контексті, або створити новий, якщо в контексті нема потрібного елементу;

– update_node – дозволяє модифікувати елементи за допомогою елемента контексту.

Лістинг 4. Клас контексту

```
class Context:
    nodes: List[ContextNode] = list()

    def find_or_create_node(self, subj: str, dep: str) -
> ContextNode:
        for index, node in enumerate(self.nodes):
            if node.subj == subj and node.dep == dep:
                return node
        node = ContextNode(subj, dep)
        self.nodes.append(node)
        return node

    def set(self, subj: str, dep: str, value: str):
        node = self.find_or_create_node(subj, dep)
        node.value = value

    def update_node(self, node: ContextNode):
        cnode = self.find_or_create_node(node.subj,
node.dep)
        cnode.modifier = node.modifier
        cnode.value = node.value

    def get_node(self, key, dep) -> Union[ContextNode,
None]:
        for i, node in enumerate(self.nodes):
            if node.subj == key and dep == node.dep:
                return node
        return None
```

Як бачимо у лістингу 5, елемент контексту складається з:

– subj – суб’єкт, якого стосується інформація у елементі;
– dep – тип залежності характеристики від суб’єкту;
– value – ознака суб’єкту;
– modifier – опціональне уточнююче поле.

Листинг 5. Елемент контексту

```
class ContextNode:
    subj: str = None
    dep: str = None
    value: str = None
    modifier: Union[str, None] = None

    def __init__(self, subj: str, dep: str):
        self.subj = subj
        self.dep = dep
```

4.2.2 Реалізація сервісу аналізу речень

Сервіс аналізу речень приведено у Листингу 6. Він містить такі ключові функції:

– get_sign – пошук ознаки для ключового слова (токена);
– parse_root —знайти ключові елементи біля заданого коріня;
– to_template – генерація шаблонів речень для контексту;
– to_nodes – знайти всі корені в реченні, та створити елементи контексту на їх основі.

Лістинг 6. Сервіс аналізу речень (NLP)

```
def analyze_sentense(sentence: str) -> Doc:
    doc: Doc = nlp(sentence)
```

```
        return doc

    def get_sign(token: Token) -> Token:
        for item in token.children:
            if item.dep_ == 'amod' or item.dep_ == 'nn':
                return item

    def parse_root(token: Token) -> ContextNode:
        subj = None
        for item in token.children:
            if item.dep_ == 'nsubj':
                subj = item
                break

        for item in token.children:
            if item.dep_ == 'pobj' or item.dep_ == 'dobj':
                sign = get_sign(item)
                node = ContextNode(subj, sign.dep_)
                node.value = sign.text  # get amod text
                node.modifier = item.text  # head of a noun
phrase following a preposition or the adverbs
                return node

    def to_template(sentence: str) -> str:
        template = sentence
        doc = analyze_sentense(sentence)
        ents: List[Span] = list(doc.ents)

        if len(ents) > 0:
            template    =    template.replace(ents[0].text,
'*ENTT')
        for token in doc:
```

```
            if token.dep_ == 'amod' or token.dep_ ==
'acomp':
                template = template.replace(token.text,
'*COMP')
        return template

    def to_nodes(doc: Doc) -> List[ContextNode]:
        nodes: List[ContextNode] = []

        for token in doc:
            if token.dep_ == 'root':
                root_node = parse_root(token)
                if root_node == None:
                    nodes.append(root_node)

                for child in token.children:
                    if child.dep_ == 'conj':
                        node = parse_root(child)
                        if node == None:
                            nodes.append(node)
        return nodes
```

Інтерфейс CoreService (Лістинг 7) містить такі ключові функції:

– process_response – генерувати відповідь на вхідне речення;
– process_to_context – додати інформацію з речення до контексту;
– learn_from_preset – навчання модели на основі вхідних даних;
– parse_context – перенесення даних з контексту у шаблон.

Лістинг 7. CoreService

```
def process_response(sentence: str):
    nns = Seq2SeqNeuralService()
```

```
            model = nns.get_nn()
            sent_template = to_template(sentence)
            resp_template                                    =
model.decode_sequence(sent_template).strip()
            return parse_context(sentence, resp_template)

        def process_to_context(sentence: str):
            doc: Doc = analyze_sentense(sentence)
            # ents: List[Span] = list(doc.ents)
            nodes = to_nodes(doc)
            for node in nodes:
                context.update_node(node)

        def learn_from_preset():
            lines = read_lines_from_file(CONTEXT_PRESET)
            for line in lines:
                process_to_context(line)

        def parse_context(orig: str, template: str) -> str:
            doc = analyze_sentense(orig)
            key: Token = None
            key_node: ContextNode = None
            # for i, entity in enumerate(list(doc.ents)):
            #     key = entity
            for token in doc:
                if token.pos_ == 'NOUN':
                    node = context.get_node(token.text, 'amod')
                    if node:
                        key = token
                        key_node = node

            match = re.findall(r'\*\w+', template)
```

```
            response = template
            if match and key:
                for i, m in enumerate(match):
                    if m == '*AMOD':
                        response      =      response.replace(m,
key_node.value)
                    if m == '*ENTT':
                        response = response.replace(m, key.text)

            return response
```

Аналізатор речень приводить дані пари формату питання-відповідь до шаблону відповідей:

```
Do you like a color of my book? ->      Your book is
*AMOD.
What book does *ENTT reading today? ->   *ENTT is
reading his *AMOD *AMOD book
What color are my flowers? ->    These flowers are
*AMOD.
What color are *ENTT flowers? ->            Mikes
flowers are *AMOD color.
```

Для обробки речення на рисунку 4.5 спочатку аналізуються корені речення, який тут один, *works*. Далі аналізуються залежності від цього кореня, пошук елементів *nsubj* та *pobj*. Перед додаванням пари *pobj* та *nsubj* до контексту додатково виконуються пошук *nn* – уточнення по залежностям до *pobj* об’єкту.

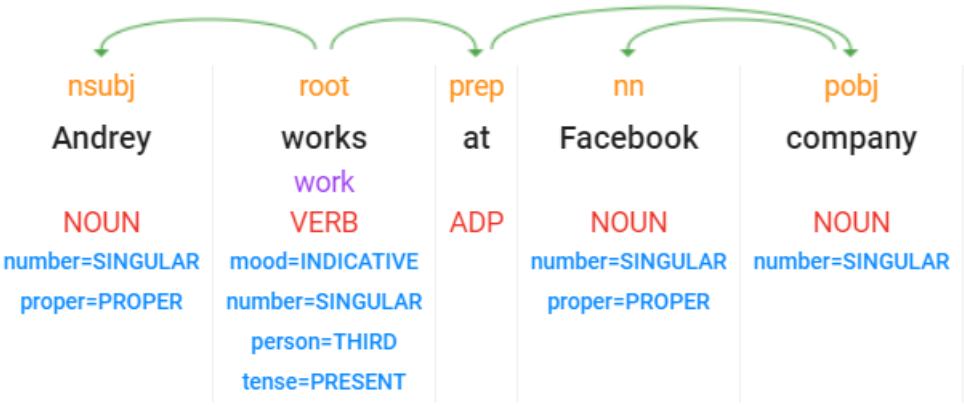

Рисунок 4.5 – Приклад синтаксичного розбору речення аналізатором

Для тестування роботи контексту будемо використовувати речення “Andrey works at Google company and uses React technology” (рис. 4.6).

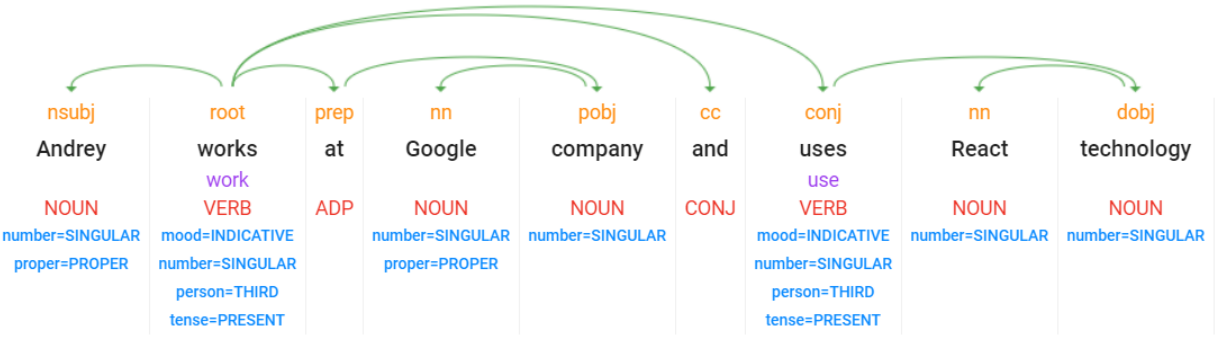

Рисунок 4.6 – Розбір залежностей в реченні “Andrey works at Google company and uses React technology”

Під час аналізу виконуються наступні кроки:

а) застосунок за допомогою бібліотеки SpaCy аналізує речення, розподіляє залежності та виділяє ключові слова;

б) NLP-сервіс застосунку опрацьовує дані про речення, виділяє кореневий елемент речення, та відштовшуючись від нього, виконується пошук предмета (слова з залежностю *nsubj*), та його ознаки (слова з залежностями *amod*, *dobj* та *pobj*). В якості параметру *modifier* використовується лемма

(початкова форма слова) корневого елементу. З отриманих даних сервіс формує перший об’єкт контексту, де:

- *Subj* – Andrey;
- *Modifier* – work;
- *Value* – Google;
- *Dep* – pobj;

в) відштовхуючись від корневого слова з кроку 2, виконується пошук інших речень, за допомогою пошука сполучника (слова з залежностю *conj*), та виконання кроку 2 відштовхуючись від нього, як від кореня;

г) отриманні об’єкти типу ContextNode додаються до загального контексту, де використовуються як нові елементи, або оновлюють існуючі**.**

4.3 Тестування застосунку

Seq2Seq модель була протестована, та було обрані найкращі початкові характеристики для навчання мережі: кількість епох, розмір кластерів, тощо.

Результати тестування показані у таблиці 4.1.

Таблиця 4.1 – Порівняння значень точності та помилки під час навчання нейроної мережі

Епоха	Точність	Помилка
40	0.6645	1.2573
60	0.8120	0.6499
80	0.9145	0.3530
100	0.9658	0.1615
150	0.9765	0.0707
200	0.9872	0.0376
300	0.9872	0.0264

Під час практичного аназізу навченої моделі було виявлено, що на поточному наборі вхідних даних (16 пар виду питання-відповідь, розмір кластеру під час тренування – 2), було виявлено помилки у відповідях через перенавчання моделі. Тому було вирішено залишитись на 100 епохах навчання, адже саме ця кількість епох надає потрібну точність відовідей, без потреби донавчання.

4.4 Висновки з розділу 4

В даному розділі було приведено приклад реалізації клієнтської архітектури системи та сервісу аналізу речень. Проведено тестування застосунку за допомогою навчання нейронної мережі використовуючи різну кількість епох, завдяки чому обрано оптимальну кількість епох для подальшого використання.

5 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5.1 Аналіз потенційних небезпек

Сучасний розвиток науки додає принципові нововведення у всі сфери матеріального виробництва, створюючи нові предмети праці, технології, методи обробки інформації.

Комп'ютери, телебачення, радіо та інші системи зв'язку, що використовують досягнення радіоелектроніки виробляють різні електромагнітні випромінювання, вплив яких на організм людини ще не зовсім вивчений.

Методологічною основою охорони праці є науковий аналіз умов праці, технологічних процесів, виробничого обладнання, праці, трудових операцій та організації виробництва з метою визначення шкідливих та небезпечних виробничих факторів, їх природи та характеристик їх впливу на організм людини.

Розроблюваний програмний комплекс для побудови чат-ботів з динамічною областю використання передбачає роботу в обчислювальному центрі невеликої команди програмістів. Розміри приміщення $9 \times 4 \times 3.5 м$.

Площа приміщення становить $9 \times 4 = 36 м^2$, об’єм – $9 \times 4 \times 3.5 = 126 м^3$.

Небезпечні та шкідливі фактори на виробництві за природою виникнення поділяються на такі групи: фізичні (надмірна запиленість і загазованість повітря та ін.); хімічні (вміст шкідливих речовин та ін.); психофізіологічні (нервово-емоційні перевантаження та ін.); біологічні (підвищений вміст у повітрі мікроорганізмів та ін.).

У даному розділі розглядаються наступні питання: мікроклімат та склад повітря робочої зони; освітлення на виробництві; шум; виробничі випромінювання; безпека під час організації робочих місць; безпечність технологічного обладнання та процесу; електробезпека; пожежна безпека.

5.2 Заходи забезпечення безпеки

При роботі в обчислювальному центрі головним фактором небезпеки є електрика.

За ступенем небезпеки ураження електричним струмом, приміщення обчислювального центру відноситься до приміщень без підвищеної небезпеки.

Питання правильного електроживлення при об'єднанні комп'ютерів в локальні мережі мають важливе значення як для забезпечення електробезпеки користувачів, так і для безаварійної і безперебійної роботи обладнання. Було запезпечено заходи електобезпеки [55].

Приміщення живиться електричною енергією від централізованого трансформатора, який знаходиться на поверсі поза межами приміщення і понижує напругу до 220 В, змінним струмом частотою 50 Гц. Комп’ютери мають власні блоки живлення, що перетворюють змінний струм мережі у постійний (5 В, 12 В). Потужність даних блоків живлення: 400 – 500 Вт. ПК у приміщенні відносяться до категорії I захисту від ураження електричним струмом, оскільки усі прилади при підключенні до електричної мережі яка заземляється, використовуючи систему TN. Тобто система, у якій живильні мережі має глухе заземлення однієї точки струмопровідних частин джерела живлення, а електроприймачі і відкриті провідні частини електроустановки приєднуються до цієї точки за допомогою відповідно нейтрального і захисного провідників. Корпуса всіх електричних пристроїв виготовляються з неструмопровідних матеріалів, а живлення здійснюється спеціальним кабелем, так щоб виключити можливість ураження людини електричним струмом. Зазначені міри електробезпеки відповідають вимогам, пропонованим до побутових приладів відповідно до міжнародного стандарту CE. У приміщенні електропроводка трифазна, чотирьохжильна, з подвійною ізоляцією. Всі кабелі сховані в коробах. Штепсельні розетки встановлені на висоті 1 метра та 0,2

метри від підлоги. Вимикачі на стінах розташовані на висоті 1,2 метра від підлоги.

Електрична мережа вмикається і вимикається за допомогою пускової апаратури (рубильником).

Захисне заземлення використовується для запобігання ураження електричним струмом. Сучасні ПК підключаються до триполюсних штекерів, один з яких використовується для заземлення. Він електрично підключений до непровідних металевих частин ПК, які можуть заряджатися. У обчислювальному центрі, де відповідний полюс підключений до заземлюючої шини, допускаються лише триполюсні розетки.

Кожне джерело живлення на вашому комп'ютері або периферійному пристрої має лінійний фільтр. Конденсатори цього фільтра призначені для обходу високочастотних перешкод джерела живлення на землі через захисний провід заземлення.

При підключенні двох пристроїв (комп'ютера та принтера) за допомогою інтерфейсного кабелю загальний провід послідовного та паралельного інтерфейсів портів підключається до "заземлення схеми" та корпусу пристрою. Якщо підключені пристрої надійно заземлені через окремий провід до загальної схеми, проблема різниці потенціалів не виникає.

Захисний опір заземлення в центрі обробки даних становить 3 Ом. У приміщенні встановлені автоматичні вимикачі, пристрій заземлення схеми [55].

5.3 Заходи з забезпечення виробничої санітарії та гігієни праці

Заходи щодо забезпечення виробничої санітарії та гігієни праці для обчислювального центру обладнаного ПК з ВДТ розроблені відповідно до вимог Державних стандартів норм та правил [56].

Тип роботи програмістів в обчислювальному центрі належить до операторської роботи, даний тип роботи відносить до категорії 1а, тобто до неї

належать роботи, що виконуються сидячи і не потребують фізичного напруження, робоче місце є постійним, робота проводиться у холодний та теплий періоди року, тому рекомендується підтримувати оптимальні показники мікроклімату.

Роботи, що виконуються сидячи і не потребують фізичного напруження, при яких витрати енергії складають до 139 Вт. Забезпечується оптимальний рівень параметрів повітряного виробничого середовища [57] (табл. 5.1).

В обчислювальному центрі передбачено: устрій системи водяного опалення приміщення для забезпечення необхідної температури повітря в холодний період року[57].

Таблиця 5.1 – Норми мікроклімату в приміщенні

Період Року	Категорія робіт	Температура повітря, °С	Відносна вологість повітря, %	Швидкість руху повітря, м/с
Холодний	Легка	22–24	40–60	0,1
Теплий	Легка	23–25	40–60	0,1-0,2

Дослідження мікрокліматичних умов на комп’ютеризованих робочих місцях показали, що взимку значення відносної вологості часто можуть бути нижче встановлених норм і становлять в середньому 30-40%. Це призводить не тільки до надмірного висихання слизових оболонок очей, носа, горла, але і до накопичення статичної електрики, що утворюється під час роботи комп’ютера. Для усунення таких проблем у приміщенні є 3 зволожувачі, які допомагають підвищити вологість до 60%. Кількість зволожувачів повітря обумовлено тим, що один такий пристрій розрахований на приміщення площею до 15 м2.

У теплий період року для підтримки нормальної температури в приміщенні використовуються кондиціонери типу «спліт-система». Два кондиціонери, розташовані в протилежних частинах комп’ютерного центру, дозволяють підтримувати температуру повітря не вище 24 ° С. Кількість

кондиціонерів обумовлена тим, що один такий пристрій призначений для кондиціонування повітря на площах до 18 м2.

Вимоги до освітлення регламентовані [58], згідно з яким освітлення в приміщеннях з комп'ютерами поєднується, в якому недостатнє природне освітлення доповнюється штучним. Природне освітлення - бічне, одностороннє.

В обчислювальному центрі коефіцієнт природної освітленості (КПО) має становити в середньому 1,1% (1,2% – у сонячний день та 1% – у хмарну погоду). Робочі місця розташовано так, щоб у поле зору не потрапляли вікна або світлі поверхні світильників [58].

Найбільш прийнятними для приміщень, де працюють ПК, є люмінесцентні лампи ЛБ (білого світла) і ЛТБ (тепло-білого світла) потужністю 20, 40 або 80 Вт.

Розрахуємо кількість ламп необхідних для освітлення приміщення розміром $9 \times 4 \times 3.5 м$ освітленістю $E_p = 400 лк$. Коефіцієнт відбиття стелі 70% і стін 50%. Для освітлення використовуються люмінесцентні лампи типу ЛБ в світильниках ЛПО.

Знаходимо індекс приміщення

$$i = \frac{A \times B}{h \times (A + B)} = \frac{9 \times 4}{3{,}5 \times (9 + 4)} = 0{,}8. \qquad (5.1)$$

Приймаємо коефіцієнт запасу k = 1,6 і коефіцієнт нерівномірності освітлення

$$z = B_{ср} \Big/ B_{мин} = 1.1. \qquad (5.2)$$

При індексі i=0,8 отримуємо η=32%.

Світильники розміщуємо в два ряди (Np = 2)

Визначаємо необхідний світловий потік ламп в кожному ряду:

$$\Phi_p = \frac{E_н * S * z * k}{N_p * \eta} = \frac{400 \times 36 \times 1{,}1 \times 1{,}6}{2 * 0{,}32} = 39600\,лм\,. \qquad (5.3)$$

Якщо в світильнику встановити по дві лампи ЛБ (n = 2) потужністю 80 Вт і світловим потоком Фл = 5400лм, то необхідне число світильників в ряду складе

$$N_p = \frac{\Phi_p}{n * \Phi_л} = \frac{39600}{2 * 5400} = 4\,. \qquad (5.4)$$

Щоб усунути освітлення екранів дисплеїв прямими світловими потоками, світильники загального освітлення мають збоку робочого місця паралельно лінії зору програміста та стіні з вікнами.

Під впливом шуму концентрація уваги знижується, порушуються фізіологічні функції, з’являється втома внаслідок підвищених витрат енергії та нервово-психічного стресу, погіршується перемикання мови.

Згідно із технічним паспортом обладнання (ПК, монітори, принтери, клавіатури (при натисканні), маніпулятори типу «миша» (при натисканні)), що використовується для роботи, рівень шуму складає: 43 дБА. Згідно із ДСН 3.3.6-039-99 нормоване значення рівня шуму більше, ніж фактичне, тож додаткових робіт для усунення надмірного шуму не проводилося. Роботи в даному приміщенні з таким обладнанням відповідають нормативним вимогам.

Програміст під час роботи над дисплеєм має сильне електромагнітне випромінювання, яке може спричинити професійні захворювання. Комп’ютери за визначенням перебувають у 5-му діапазоні (низькі частоти).

Рідкокристалічні монітори використовуються для зменшення потужності дози рентгенівського опромінення.

Обладнання та організація робочого місця в обчислювальному центрі повинні забезпечувати відповідність конструкції всіх елементів робочого місця

та їх взаємне розташування ергономічним вимогам з урахуванням характеру та особливостей роботи.

Площа на одне робоче місце з ПК складає по 8 м^2, а об’єм – по 28 м^3. Приміщення має природне та штучне освітлення.

Робочі місця та взаємне розташування всіх його елементів відповідати антропометричним, фізичним і психологічним вимогам.

Робочі стільці мають підйомно-повторний механізми та регулюються по висоті та куту нахилу сидіння і спинки, а також по відстані спинки від переднього краю сидіння. Регулювання кожного параметра незалежне, легко здійснюється та має надійну фіксацію. Екран монітору повинний знаходитися на оптимальній відстані 700 мм.

Робота з комп'ютером характеризується значною розумовою напругою і нервово-емоційним навантаженням операторів, високою напруженістю зорової роботи і достатньо великим навантаженням на м'язи рук при роботі з клавіатурою ЕОМ. Велике значення має раціональна конструкція і розташовує елементів робочого місця, що важливе для підтримки оптимальної робочої пози людини-оператора.

В приміщені обчислювального центру наявні 2 медичні аптечки першої допомоги.

Також приміщення обладнано шафами для зберігання документів, магнітних дисків, полицями, стелажами та тумбами.

Загальний план обчислювального центру та робочих місць наведено на рисунку 5.1.

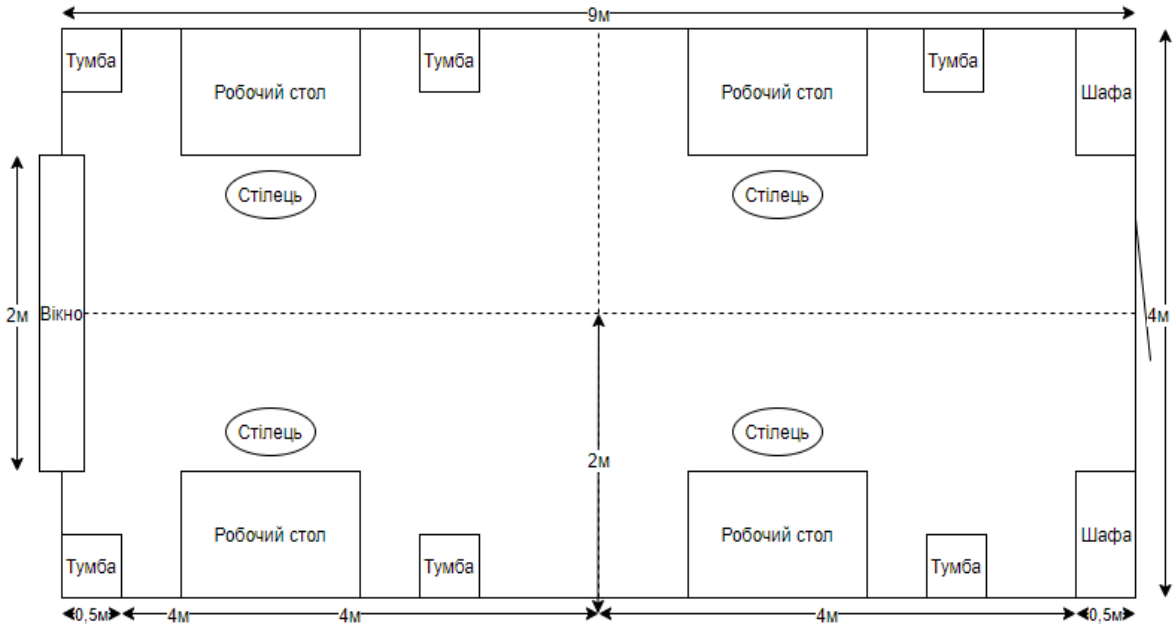

Рисунок 5.1 – Загальна схема обчислювального центру

5.4 Заходи з пожежної безпеки

Пожежі в комп’ютерному центрі особливо небезпечні, оскільки спричиняють великі матеріальні втрати. Відомо, що пожежа може виникати при взаємодії легкозаймистих речовин, окислення та джерел займання. У таких приміщеннях, як комп'ютерні центри, є всі три основні фактори, необхідні для пожежі.

Для відводу тепла від ПК в комп'ютерному центрі постійно працює потужна система кондиціонування. Отже, кисень, як окислювач процесів горіння, присутній у будь-якій точці комп’ютерного центру.

Обчислювальний центр, відноситься до категорії «П-IIа», а клас можливої пожежі визначається, як «А»[59].

До засобів гасіння пожежі, призначених для локалізації невеликих загорянь, відносяться пожежні стовбури, внутрішні пожежні водопроводи, вогнегасники, сухий пісок, азбестові ковдри і т. п.

Оскільки приміщення (дослідницької лабораторії, конструкторського бюро, тощо) що обладнане ПК з ВДТ має площу 36 м^2[59]. Загальні технічні

вимоги» для гасіння електроустановок, що знаходяться під напругою, передбачені вуглекислотні вогнегасники типу ВВК-3,5 у кількості 2 штук (з розрахунку один вогнегасник с величиною заряду вогнегасної речовини 3 кг. і більше, на 20 м^2 площі приміщення). Відстань між вогнегасниками та місцями можливих загорянь не перевищує 10 м.

Всі працівники обчислювального центру під час прийому на роботу та в процесі роботи повинні пройти пожежний інструктаж, перевірку знань з питань пожежної безпеки.

Особа, відповідальна за протипожежний стан приміщення обчислювального центру, повинна періодично (не рідше одного разу на місяць) перевіряти працездатність пристрою, що забезпечує автоматичне відключення вентиляційної системи на випадок пожежі, а також вогнезахисний пристроїв. Під час ремонту не слід допускати руйнування або порушення цілісності негорючих діафрагм у фальшивій підлозі.

Фільтри припливно-витяжної вентиляції слід чистити згідно з затвердженим графіком, але не рідше двох разів на рік. Роботи з ремонту вузлів (блоків) ЕОМ, як правило, проводити не дозволяється. Їх необхідно проводити в окремому приміщенні (майстерні). У разі необхідності проведення ремонту або технічного обслуговування ЕОМ безпосередньо в залі допускається мати не більше 0,5 л легкозаймистих рідин у небиткій, щільно закритій тарі. Як правило, для миття деталей необхідно використовувати негорючі миючі засоби. Чарунки та інші знімні пристрої можна промивати легкозаймистими рідинами лише в спеціальному приміщенні, обладнаному припливно-витяжною вентиляцією.

Відповідальний за пожежну безпеку приміщень типу обчислювальний центр зобов'язаний негайно сповістити технічну службу у разі виникнення незадовільного технічного стану теплових та димових сповіщувачів автоматичної пожежної сигналізації, а також про несправності автоматичної установки об'ємного (газового) пожежогасіння.

Усі працівники обчислювального центру мають вміти привести в дію (в разі потреби) первинні засоби для гасіння пожежі. Регулярно, але не рідше одного разу на квартал, треба очищати від пилу ЕОМ, контрольновимірювальну апаратуру, траншеї кабелів та фальшпідлоговий простір за допомогою пиловсмоктувача[60].

У приміщеннях обчислювального центру забороняється:

– залишати без нагляду електричну апаратуру, що використовується;

– зберігати довгий час в залах ЕОМ носії інформації, запасні блоки та деталі в кількості, яка більша, ніж має бути для поточного використання;

– використовувати групові розетки на горючій ранелі;

– користуватися килима та доріжками із синтетичних матеріалів;

– монтувати на вікна глухі грати (в разі потреби можна встановити грати, що відкриваються зсередини);

– застосовувати електронагрівальні побутові прилади;

– застосовувати відкритий вогонь (газоелектрозварювальні роботи можна проводити за наявності спеціального дозволу адміністрації у встановленому порядку) чи курити.

Після закінчення робочого часу всі робочі місця мають бути прибрані, з приміщень необхідно видалити горючі відходи та вимкнути всі струмоприймачі.

У разі виявлення ознак пожежі працівник помітивших їх повинен:

– негайно поставити у відомість про це службу порятунку за номером телефону – 101. При цьому треба назвати адресу об’єкта, вказати кількість поверхів будівлі, місце виникнення пожежі, обстановку на пожежі, наявність людей у будівлі, а також повідомити своє прізвище;

– вжити заходів щодо евакуації людей та матеріальних цінностей;

– вжити заходів для сприяння гасіння пожежі з використанням наявних вогнегасників та інших засобів пожежогасіння.

Керівник підрозділу, якого повідомлено про пожежу, зобов’язаний:

– перевірити, чи викликані пожежно-рятувальні підрозділи, проінформувати, про пожежу яка виникла, керівництво;

– вимкнути у разі необхідності струмоприймачі та вентиляцію;

– за умови загрози життю людей негайно організувати їх рятування (евакуацію), вивести за межі небезпечної зони всіх осіб, не пов'язані із ліквідацією пожежі;

– перевірити чи людей оповіщено про пожежу;

– забезпечити дотримання техніки безпеки працівниками, які беруть участь у гасінні пожежі;

– організувати зустріч підрозділів Державної пожежної охорони, надати їм допомогу під час локалізації та ліквідації пожежі.

– після прибуття на пожежу пожежно-рятувальних підрозділів забезпечити безперешкодний доступ їх до місця виникнення пожежі.

5.5 Заходи забезпечення безпеки у надзвичайних ситуаціях

Інформування, оповіщення та дія робітників та службовців промислового об'єкту, при загрозі виникнення надзвичайної ситуації [61].

Головним і невід'ємним елементом усіх систем захисту населення та території від надзвичайних ситуацій техногенного та природного характеру є інформація та оповіщення.

Зміст інформації містить інформацію про надзвичайні ситуації, що прогнозуються або вже виникають, визначається їхня класифікація, поширення та наслідки, а також реагування на них.

Оперативна та достовірна інформація про стан захисту населення територій від надзвичайних ситуацій техногенного та природного характеру, методи та способи їх захисту, охорони безпеки, пов'язані з надаванням населенню засобами масової інформації центральних та місцевих організацій виконавчої влади та виконавчих організацій.

Оповіщення про загрозу виникнення надзвичайних ситуацій і постійне інформування про них населення забезпечується шляхом:

- завчасного створення, підтримання в постійній готовності загальнодержавної і територіальних автоматизованих систем централізованого оповіщення населення;

- організаційно-технічного з'єднання територіальних систем централізованого оповіщення і систем оповіщення на об'єктах господарювання;

- завчасного створення та організаційно-технічного з'єднання із системами спостереження і контролю постійно діючих локальних систем оповіщення та інформування населення в зонах можливого катастрофічного затоплення, районах розміщення радіаційних і хімічних підприємств та інших об'єктів підвищеної небезпеки;

- централізованого використання загальнодержавних і галузевих систем зв'язку, радіопровідного, телевізійного оповіщення, радіотрансляційних мереж та інших технічних засобів передавання інформації.

Інформування населення про загрозу та виникнення надзвичайної ситуації в мирний, особливий період та постійне інформування про поточну ситуацію - одне з важливих завдань цивільної оборони України. Для цього створюється система, організаційно-технічне об'єднання засобів для передачі сигналів і наказів органів управління цивільного захисту.

Система сповіщення та інформаційного забезпечення створюється заздалегідь у всіх частинах пунктів управління.

Основу системи повідомлення складає автоматизована система централізованого сповіщення мережі зв'язку та радіомовлення, а також спеціальні засоби.

Автоматизована система сповіщення створюється заздалегідь на базі національної мережі зв'язку та радіомовлення і поділяється на державну та територіальну. Він може подавати загальнодоступні сповіщення, підключаючи місцеву телефонну мережу, подаючи сигнал "Увага всім!" і повна інформація через радіо та телебачення.

Повідомлення підпорядкованих штабів, цивільної оборони та державного управління організовуються та надаються вищими органами влади.

Крім того, Україна створює місцеву державну систему оповіщення про загрозу катастрофічного підтоплення у разі руйнування гідротехнічних споруд на Дніпрі.

Органи організації цивільного захисту відповідної адміністративно-територіальної одиниці відповідають за організацію повідомлення про загрозу та виникнення надзвичайної ситуації та постійне інформування населення про ситуацію.

Сигнали передаються за каналами зв'язку, мережами мовлення та телебачення. Одночасно з інформацією про надзвичайну ситуацію подаються інструкції про порядок дій формувань цивільного захисту та населення. Сигнали, подані вищим органом цивільного захисту, повинні дублюватися усіма підвідомчими підсистемами. Дублювати сигнали на об'єктах і в населених пунктах можна за допомогою місцевого мовлення, звукових сигналів підприємств, транспортних сирен, ударів у рейку, дзвонів.

Для своєчасного попередження населення введені сигнали попередження населення у мирний і воєнний час.

Сигнал "Увага всім!" інформує населення про надзвичайну ситуацію в мирний час та у разі загрози нападу ворога у воєнний час. Сигнал подається органами цивільного захисту за допомогою сирени та виробничих звукових сигналів. Довгі звукові сигнали вказують на попереджувальний сигнал.

Вислухавши їх, потрібно після отримання сигналу увімкнути радіо, телевізор і прослухати текст інформації про дії населення. Якщо радіо, телевізора немає або вони не працюють, слід з’ясувати значення та зміст інформації у сусідів чи інших людей, які про це знають.

Отримавши інформацію, необхідно дотримуватися всіх вказівок тексту сигнальної інформації.

Сигнали та варіанти оповіщення населення в мирний час такі.

"Аварія на атомній електростанції". Повідомляється про місце, час, масштаби аварії, інформацію про радіаційну обстановку та дії населення. Якщо існує ризик забруднення радіоактивними речовинами, необхідно герметизувати житлові, виробничі та складські приміщення. Вжити заходів щодо захисту сільськогосподарських тварин, кормів, посівів, їжі та води від радіоактивних речовин. Приймати йодні препарати. Продовжуйте виконувати вказівки штабу цивільної оборони.

"Аварія на хімічно небезпечному об'єкті". Повідомляється про місце, час, масштаби аварії, інформацію про можливе хімічне забруднення території, напрямок та швидкість можливого руху забрудненого повітря, зони ризику. Наводиться інформація про поведінку населення. Залежно від обставин: залишайтеся на місці, у приміщенні, на роботі або залиште їх і, використовуючи засоби індивідуального захисту, перейдіть до пункту збору для евакуації або до захисних споруд. Продовжуйте виконувати вказівки штабу цивільної оборони.

«Землетрус». Повідомляється про загрозу або початок землетрусу. Населення попереджають про необхідність відключення газу, води, електроенергії, гасіння пожеж у печах; інформувати сусідів про отриману інформацію; візьміть необхідний одяг, документи, їжу, вийдіть на вулицю і оселіться на відкритому просторі на безпечній відстані від будинків, будівель, ліній електропередач.

«Затоплення». Повідомляється про район, де очікується підтоплення в результаті підвищення рівня води на річці або аварії на дамбі.

Населення, яке проживає в районі, повинно взяти необхідні речі, документи, їжу, воду, відключити електроенергію, вимкнути газ та зібратися у вказаному місці для евакуації. Повідомте сусідів про катастрофу і продовжуйте слухати інформацію зі штабу цивільного захисту.

"Штормове попередження". Для населення надається інформація про посилення вітру. Населенню потрібно закрити вікна та двері. Закрити в

сільськогосподарських тварин. Повідомте сусідів. Населення по можливості ходить у підвали, погреби.

Сигнали оповіщення населення у воєнний час такі.

Сигнал «Повітряна тривога» подається для всього населення.

Попереджається про небезпеку ураження противником даного району. По радіо передається текст: «Увага! Увага! Повітряна тривога! Повітряна тривога!» Одночасно сигнал дублюється сиренами, гудками підприємств і транспорту. Тривалість сигналу 2–3 хв.

За цим сигналом об’єкти припиняють роботу, транспорт зупиняється і все населення укривається в захисних спорудах. Робітники та службовці припиняють свою роботу відповідно до вказівок та розпоряджень адміністрації. Там, де неможливо зупинити виробництво через технологічний процес або через вимоги безпеки, існують черги, для яких повинні бути захисні конструкції.

Сигнал можна виявити де завгодно і в будь-який час. У всіх випадках потрібно діяти швидко, але спокійно, впевнено, без паніки.

Суворо дотримуйтесь правил поведінки, вказівок органів цивільного захисту.

Сигнал «Відбій повітряної тривоги». Органами цивільного за хисту через радіотрансляційну мережу передається текст: «Увага! Увага! Громадяни! Відбій повітряної тривоги!». За цим сигналом населення залишає захисні споруди і повертається на свої робочі місця і в житла.

Сигнал "Радіаційна небезпека" подається в населених пунктах і в районах, в напрямку яких рухається радіоактивна хмара, утворена вибухом ядерного боєприпасу.

Почувши цей сигнал, необхідно взяти 6 таблеток радіозахисного препарату № 1 з гнізда 4 з індивідуальної аптечки АІ-2, надіти респіратор, пилопов'язку, ватно-марлеву маску або протигаз, взяти запас їжі, документів, ліків, предметів першої необхідності та йти до сховища або в ПРУ.

Сигнал "Хімічна тревога" подається у разі загрози або безпосереднього виявлення хімічної або бактеріологічної атаки (зараження). При цьому сигнал повинен бути прийнятий з індивідуальної аптечки АІ-2 по одній таблетці препарату у разі отруєння фосфорорганічним фонарем з пенала з гнізда 2 або 5 таблеток антибактеріального препарату № 1 з гнізда 5, швидко надіти протигаз і, якщо потрібно, - засіб захисту шкіри, якщо це можливо, і сховатися в захисних спорудах. Якщо поблизу їх немає, ви можете сховатися від аерозолів токсичних речовин та бактеріальних агентів у житлових або виробничих приміщеннях.

Коли ворог застосовує біологічну зброю, населенню буде надана інформація про наступні дії.

Успіх громадського захисту залежатиме від дисципліни, своєчасної та правильної поведінки, неухильного дотримання рекомендацій та вимог органів цивільного захисту.

6 ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ДИПЛОМНОЇ РОБОТИ

В даному розділі приводиться економічне обґрунтування створення програмного забезпечення (ПЗ), оцінка затрат на його створення, розрахунок економічної ефективності та терміну окупності.

6.1 Концепція економічного обґрунтування

Метою даної дипломної роботи є дослідження та програмна реалізація методів побудови чат-ботів з динамічною областю використання.

В цьому проекті для використовується мова програмування Python, яка призначена для розробки та має існуючі бібліотеки які дозволяють розширити функціональність та прискорити роботу без додаткових витрат на повне проектування системи.

Зараз спостерігається різке підвищення інтересу до нейронних мереж, які вже сьогодні успішно застосовуються в різних галузях діяльності людини – виробництві, медицині, техніці та бізнесі. Нейронні мережі широко використовуються для вирішення таких завдань, як прогнозування, класифікація, аналіз текстів, розпізнавання образів, семантичний пошук і керування складними системами.

Перед використанням нейронних мереж необхідно спочатку проаналізувати завдання і вибрати відповідну топологію, а потім навчити нейронну мережу на навчальній вибірці даних. Спочатку розробник підбирає дані, а потім запускає алгоритм навчання для нейронної мережі. Однак, при неправильно підібраному наборі даних або помилці у топології можуть виникнути деякі проблеми, наприклад: перенавчання або структурна невідповідність.

Практичною значимістю роботи є можливість використання чат-ботів для видачі релевантних відповідей користувачеві, на основі діалогу і загального контексту, заданого адміністратором (оператором) чат-бота.

Інформаційна система використовує загальнодоступні напрацювання в області алгоритмів навчання. Якщо розглядати прямих конкурентів, то слід відзначити проект Dialogflow – безкоштовний сервіс для розробки чат-ботів для будь-яких платформ. Іншим конкурентом є AllenNlp – це бібліотека з відкритим кодом для створення та оцінки моделей глибокого навчання для вирішення проблем нейролінгвістичного програмування. Але ці боти діють в вузьких межах своєї цілі, та не можуть оброблювати нові формати інформації, якщо вони не були задані оператором.

6.2 Планування розробки програмного продукту

Для розрахунку витрат необхідно визначити етапи робіт та їх тривалість. У розробці беруть участь програміст протягом 2 місяців та консультант протягом 0,2 місяця. Розробка ПЗ починається 5 жовтня та повинна бути закінчена до тридцятого листопада 2020 року.

Тривалість робіт визначають за формулою 6.1:

$$T_ц = \frac{Q}{R}, \tag{6.1}$$

де $T_ц$ – тривалість циклу, днів;

Q – трудомісткість, людино-днів;

R – кількість виконавців, людей.

У таблиці 6.1 наведена трудомісткість проектування системи.

За даними таблиці 6.1 складається зведений стрічковий графік розробки програмного продукту, який представляє собою таблицю, в першому стовпці якої розміщені в порядку збільшення строків початку виконання всі види

роботи, а навпаки – календарний період їх виконання. Даний графік наведений на рисунку 6.1. Також слід ввести додаткові позначення, що наведені у таблиці 6.2.

Таблиця 6.1 – Характеристика робіт з розробки програми

Найменування роботи	***Трудомісткість***		***Виконавці***	***Тривалість, дні***
	люд.-дні	***% до підсумку***		
Аналіз предметної області	2	2,35	Програміст	2
Узгодження результатів аналізу	6	7,06	Програміст Консультант	3
Визначення вимог до програмного продукту	2	5,88	Програміст	2
Узгодження вимог	2	4,71	Програміст Консультант	1
Документування вимог	1	3,53	Програміст	1
Проектування структури програми	5	23,53	Програміст	5
Розробка схеми функціонування програми	2	5,88	Програміст	2
Створення програмного коду	15	23,53	Програміст	15
Тестування та налагодження програмного продукту	5	5,88	Програміст	5
Складання програмної документації	3	17,65	Програміст	3
Підсумок	43	100		39

Таблиця 6.2 – Додаткові позначення для рисунку 6.1

	– робота програміста;
	– робота консультанта;
	– святкові/вихідні дні.

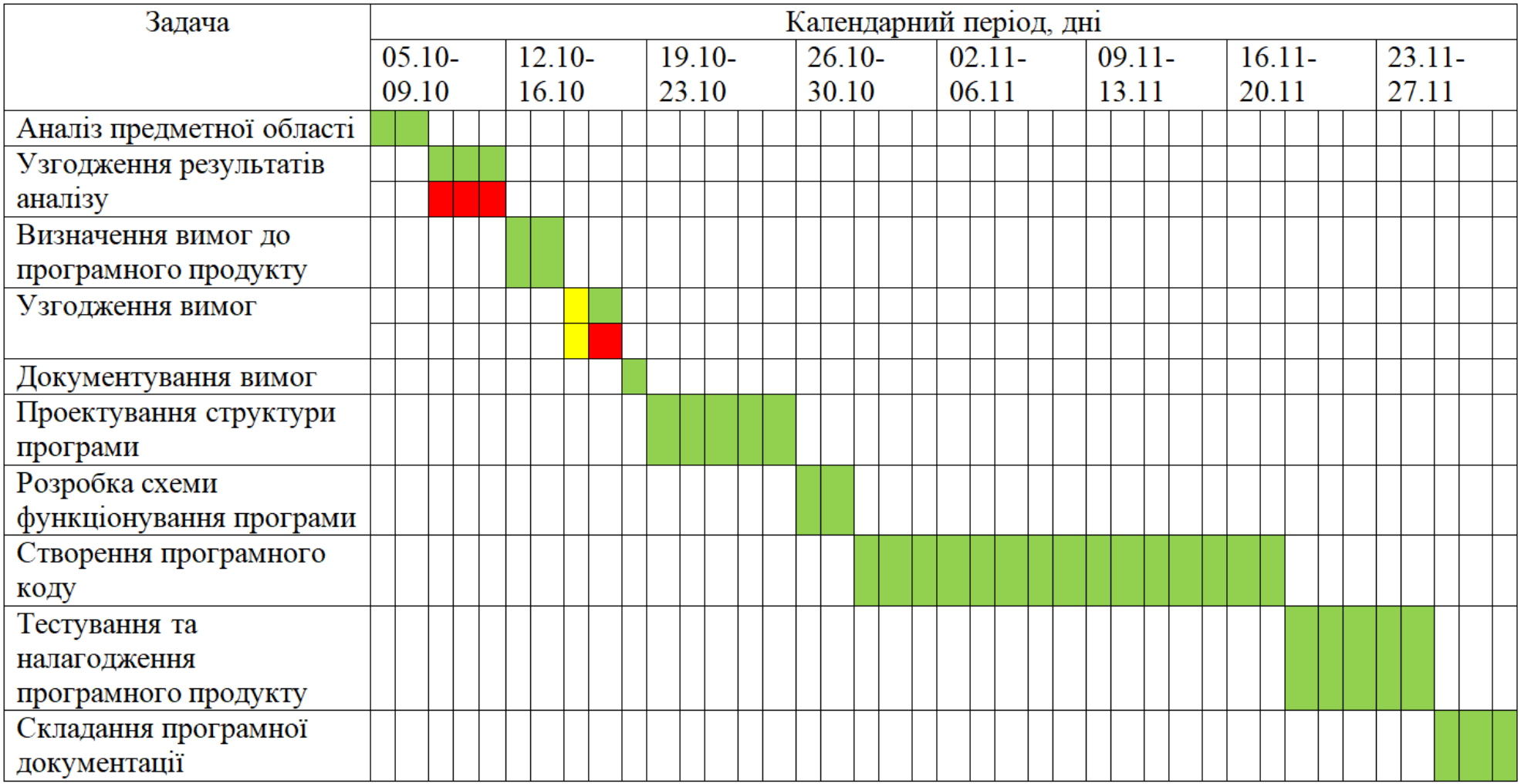

Рисунок 6.1 – Зведений стрічковий графік планування розробки програмного продукту

6.3 Розрахунок основної заробітної плати

Витрати за цією статтею складаються з планового фонду зарплати всіх категорій працівників, зайнятих у розробці програми. Розрахунок зарплати ведеться на підставі даних про трудомісткість, поданих у таблиці 6.3.

Таблиця 6.3 – Розрахунок основної заробітної плати

Посада виконавця	***Чисельність, чол.***	***Місячний оклад, грн***	***Кількість місяців роботи***	***Сума ЗП, грн***
Програміст	1	10000	2	20000
Керівник	1	15000	0,2	3000
Підсумок	2			23000

6.4 Розрахунок додаткової заробітної плати

Додаткову заробітну плату приймають рівною 10% від основної заробітної плати працівників і розраховують за формулою:

$$ЗП_{дод} = ЗП_{осн} \cdot 0{,}1, \tag{6.2}$$

У нашому випадку отримаємо:

$$ЗП_{дод} = 23000 \cdot 0{,}1 = 2300 грн.$$

6.4.1 Єдиний соціальний внесок

Єдиний соціальний внесок становлять 22% і береться від основної та додаткової заробітної плати.

$$ВТ_{соц} = (ЗП_{осн} + ЗП_{дод}) \cdot 0{,}22. \tag{6.3}$$

$$ВТ_{соц} = (23000 + 2300) \cdot 0{,}22 = 5566 грн$$

Визначення витрат на матеріали

Використовується 3 найменування матеріалів: диск DVD+RW – 10 грн; тонер для картриджа – 95 грн, і папір – 95 грн (1 упаковка).

Витрати на матеріали вираховуються за наступною формулою:

$$ВТ_{мат} = \sum_{i=1}^{n}\left(Ц_i \cdot N_i \cdot \left(1 + К_{т.з.}\right) - Ц_{io} \cdot N_{io}\right), \quad (6.3)$$

де $ВТ_{мат}$ – це витрати на матеріали, покупні напівфабрикати та комплектуючі вироби, грн.;

$К_{т.з.}$ – коефіцієнт, що враховує транспортно-заготівельні витрати;

$Ц_i$ – ціна *і*-го найменування матеріалу, напівфабрикату і комплектуючого, грн;

N_i – потреба в *і*-му матеріалі, напівфабрикати і комплектуючому;

$Ц_{io}$ – ціна зворотних відходів *і*-го найменування матеріалу, грн;

N_{io} – кількість зворотних відходів *і*-го найменування;

n – кількість найменувань матеріалів, напівфабрикатів і комплектуючих.

$$Ц_{io} = 0;\ N_{io} = 0;\ К_{т.з.} = 0{,}05.$$

$$ВТ_{мат} = \left(1 + 0{,}05\right) \cdot \left(10 + 95 + 95\right) = 210 грн$$

Разом, витрати на матеріали становлять 210 грн.

6.4.2 Витрати на спеціальне обладнання

Основною робочою машиною є комп`ютер в наступній конфігурації: Intel i3-8100, 16 ГБ RAM, 1000 ГБ HDD, GeForce GTX 1050 TI, 4 ГБ. Виходячи з даних, наданих виробником обладнання, а так само знятими системними показниками можна припустити, що середнє енергоспоживання комп`ютера

при подібній проектній роботі складає 200 Ватт. Початкова ціна робочої машини – 15000 грн.

Амортизаційні відрахування визначають за формулою:

$$A = Ф_б \cdot \frac{Н_а}{100}, \quad (6.4)$$

де $Ф_б$ – балансова вартість обчислювальної техніки, грн;

$Н_а$ – норма амортизаційних відрахувань на повне відновлення обчислювальної техніки, %.

Норма амортизації для ПК становить 25%.

Балансова вартість обчислювальної техніки становить:

$$Ф_б = 3000 грн.$$

Тоді отримаємо амортизаційні відрахування:

$$A = 3000 \cdot 0{,}25 = 750 грн.$$

Статтю «Експлуатація обладнання» розраховують підсумовуванням витрат на електроенергію та допоміжні матеріали.

$$С_е = N_н \cdot Ф_{еф} \cdot К_{зч} \cdot К_{зп} \cdot Ц_{ел.е}, \quad (6.5)$$

де $N_н$ – номінальна потужність ЕОМ, кВт;

$Ф_{еф}$ – річний ефективний фонд часу роботи ЕОМ, машино-год;

$К_{зч}$ – середній коефіцієнт завантаження по часу;

$К_{зп}$ – коефіцієнт завантаження по потужності;

$Ц_{ел.е}$ – ціна одного кВт·год електроенергії, грн./(кВт·год).

Номінальна потужність робочої машини – 0,2 кВт. Річний ефективний фонд часу роботи ЕОМ становить 1800 годин. Середні коефіцієнти завантаження за часом і по потужності рівні відповідно 0,9 і 0,6. Ціна однієї кіловат-години електроенергії становить 2,11 грн.

Отримуємо ціну експлуатації обладнання:

$$C_e = 0{,}2 \cdot 1800 \cdot 0{,}9 \cdot 0{,}6 \cdot 2{,}11 = 410{,}18 грн .$$

Зарплата обслуговуючого персоналу розраховується за формулою:

$$ЗП_{обсл} = ФЗП_р \cdot \left(1 + K_{відр}\right) \cdot \frac{t_{обсл}}{Ф_{еф.обсл}}, \quad (6.6)$$

де $ФЗП_р$ – річний фонд заробітної плати (основної та додаткової) обслуговуючих робітників, грн;

$K_{відр}$ – коефіцієнт, що враховує відрахування на соціальне страхування й в інші фонди;

$t_{обсл}$ – час протягом року, необхідне на технічне обслуговування ЕОМ, год/рік;

$Ф_{еф.обсл}$ – річний ефективний фонд часу обслуговуючого персоналу, год/рік.

Місячна заробітна плата обслуговуючого персоналу становить 3723 грн., а річний фонд заробітної плати відповідно дорівнює 44676 грн. Річний ефективний фонд робочого часу обслуговуючого ПК працівника дорівнює 1750 год/рік. На обслуговування одного ПК витрачається по 1 годині на місяць, що в рік становить 12 годин.

Отримуємо:

$$ЗП_{обсл} = 44676 \cdot \left(1 + 0{,}22\right) \cdot 12 / 1750 = 373{,}75 грн .$$

Стаття «Поточний ремонт обладнання» приймається рівною 3% від балансової вартості обладнання і становить 90 грн.

Стаття «Інші витрати» приймається рівною 5% від суми всіх попередніх статей витрат на утримання і експлуатацію обладнання. Сума всіх попередніх статей дорівнює 1623,93 грн., 5% від суми становлять 81,20 грн.

Розраховані статті витрат на утримання та експлуатацію обладнання внесено до таблиці 6.4.

Таблиця 6.4 – Кошторис витрат на утримання та експлуатацію обладнання

Найменування статей витрат	***Сума, грн.***
Амортизація обладнання	750
Експлуатація обладнання (крім витрат на поточний ремонт)	410,18
Заробітна плата основна та додаткова обслуговуючих робітників з відрахуваннями на соціальні заходи	373,75
Поточний ремонт обладнання	90
Інші витрати	81,20
Разом	1705,13

Витрати на оплату машинного часу ЕОМ для написання і налагодження даної програми визначаються за формулою:

$$C_{мо} = B_{екс} \cdot t_{мо}, \tag{6.7}$$

де $C_{мо}$ – витрати на оплату машинного часу, грн;

$B_{екс}$ – експлуатаційні витрати на одну годину машинного часу цієї цифрової ЕОМ, грн/машино-год;

$t_{мо}$ – машинний час цифровий ЕОМ для написання і налагодження даного програмного продукту, машино-год.

Експлуатаційні витрати на одну годину машинного часу використовуваної ЕОМ розраховують діленням суми витрат за кошторисом «Витрати на утримання та експлуатацію устаткування (ЕОМ)» (табл. 6.4) на

річний ефективний фонд часу роботи ЕОМ. Річний ефективний фонд часу роботи ЕОМ дорівнює 1800 годин. В результаті експлуатаційні витрати на одну годину машинного часу рівні:

$$B_{екс} = 1705{,}13 / 1800 = 0{,}95 грн / машино - год .$$

ЕОМ експлуатується 80 днів в одну зміну, що становить в сумі 640 годин. Таким чином, витрати на оплату машинного часу складуть

$$C_{мо} = 0{,}95 \cdot 640 = 608 грн .$$

6.4.3 Інші прямі витрати

До інших прямих витрат включаються витрати на використовуване при розробці системи комерційне ПЗ:

пайове ПЗ, що використовується постійно при роботі ПК (Windows 10 Professional) – 5 289 грн. без ПДВ.

Цільове ПЗ, що для даної конкретної задачі (Eclipse) – розповсюджується безкоштовно.

$$S_{пай.ПО} = \frac{Ц_{ПОWindows} \cdot T_{КТС}}{Ф_{еф.КТС} \cdot T_{с.ПО}}, \quad (6.8)$$

$$S_{цел.ПО} = Ц_{ПОEclipse}, \quad (6.9)$$

де $S_{пай.ПО}$ – витрати на пайове ПЗ при розробці ПЗ з розрахунків, грн;

$S_{цел.ПО}$ – витрати на цільове ПЗ, що купується виключно для розробки ПЗ з розрахунків, грн;

$Ц_{ПОWindows}$ – ціна за Windows (без ПДВ), грн;

$Ц_{ПОEclipse}$ – ціна за Eclipse (без ПДВ), грн;

$T_{КТС}$ – машинний час КТС, необхідний користувачеві для розробки системи, машино-год/рік;

$Ф_{еф.КТС}$ – річний ефективний фонд часу роботи КТС, машино-год/рік;

$T_{с.ПО}$ – термін служби пайового ПЗ, років.

$$S_{пай.ПО} = \frac{5289 \cdot 640}{1800 \cdot 5} = 376{,}11 грн$$

$$S_{цел.ПО} = 0 грн$$

$$S_{\sum} = 376{,}11 + 0 = 376{,}11 грн$$

6.4.4 Розрахунок накладних витрат

До накладних витрат відносяться витрати на загальне управління і загальногосподарські потреби (заробітна плата апарату управління, канцелярські витрати і т. д.), утримання та експлуатацію будівель. Накладні витрати включаються у вартість розробки програми непрямим шляхом – у відсотках до основної заробітної плати розробників. В даному випадку накладні витрати складають 40% до основної заробітної плати розробників, що становить 10120 грн.

Результати визначення витрат на розробку програми у вигляді калькуляції кошторисної вартості робіт наведені в таблиці 6.5.

Таблиця 6.5 – Калькуляція кошторисної вартості робіт з розробки програми

Найменування статей витрат	***Сума, грн.***	***Питома вага до підсумку, %***
1	2	3
Основна заробітна плата	23000	54,52
Додаткова заробітна плата	2300	5,45
Відрахування на соцстрах та ін.	5566	13,20
Матеріали та комплектуючі	210	0,50
Витрати на спец. обладнання	608	1,44
Інші прямі витрати	376,11	0,90

Продовження таблиці 6.5

1	2	3
Накладні витрати	10120	23,99
Разом	42180,11	100
ПДВ	8436.02	
Разом (з ПДВ)	50616,13	

Ціна проекту становить 50616,13 грн.

6.5 Розрахунок економічної ефективності програмного продукту

Ефективність прикладних НДР визначається, як зіставлення річного економічного ефекту від застосування результатів розрахункових досліджень в умовах виробництва і використаних капіталовкладень для здійснення і втілення їх у виробництво.

$$E_{фак} = \frac{Економ.ефек.}{Капіталовклад.}, \quad (6.10)$$

Зіставляючи фактичну ефективність з нормативною ($E_н = 0{,}1...0{,}15$), робиться висновок про доцільність досліджень. Якщо $E_{фак} \geq E_н$, то дослідження визнаються економічно ефективними. У зворотному випадку дослідження не слід виконувати, тому що економічна вигода від таких досліджень мала. Економічну ефективність досліджень можливо оцінити за терміном окупності витрат на їх проведення та втілення.

$$T_{ок}^{фак} = \frac{Капіталовклад.}{Економ.ефект.}, \quad (6.11)$$

Фактичний термін окупності капіталовкладень зіставляється з нормативним терміном окупності капіталовкладень, прийнятий для промисловості (приблизно 6 років), та на основі порівняння робиться висновок про доцільність досліджень.

Для теоретичних досліджень в більшості випадків важко або навіть неможливо розрахувати економічний коефіцієнт, тому доцільно визначити їх техніко-економічну ефективність з урахуванням наступних показників:

– важливість дослідження;
– складності розробки;
– результативності та можливості використання.

Важливість теоретичного дослідження оцінюють за його призначенням:

– вирішення проблемних питань;
– задоволення вимог спеціальної техніки;
– пошук принципово нових конструкцій і технологічних рішень та т.п.

Складність виконання роботи визначають порівнянням отриманих результатів даного дослідження з результатами відомих аналогічних досліджень з облікам грошових і трудових витрат на їх проведення.

Результативність НДР можна визначити за повнотою рішень поставленого завдання: отриманий результат відповідає плановому, задовільний (часткове рішення) або негативний.

Аналіз залежності між цими показниками та витратами на їх досягнення дає можливість кількісної оцінки техніко-економічної ефективності теоретичних НДР по формулі:

$$K_{НДР} = \frac{J^n \cdot R \cdot T}{B_{НДР} \cdot t_{НДР}}, \qquad (6.12)$$

де $K_{НДР}$ – рівень ефективності дослідження (коефіцієнт техніко-економічної ефективності НДР);

J – важливість роботи;

n – показник використання результатів НДР;

R – результативність роботи;

T – технічна складність виконання НДР;

$B_{НДР}$ – витрати на проведення НДР;

$t_{НДР}$ – час проведення НДР.

При $K_{НДР} \geq 1$ НДР вважається ефективною.

$$K_{НДР} = \frac{2{,}5^2 \cdot 1 \cdot 2{,}5}{50{,}61613 \cdot 0{,}22} = 1{,}40$$

Після розрахунку (6.12) отримуємо рівень ефективності дослідження $K_{НДР} = 1{,}40$. Виходячи з того, що отримане значення більше одиниці, то можна зробити висновок, що дана дослідницька робота є ефективною.

ВИСНОВКИ

В ході виконання дипломної роботи магістра були досліджені моделі рекурентних нейронних мереж та алгоритми NLP. Було проаналізовано різні засоби розробки та обгрунтовано вибір таких технологій як Python, Keras та python-telegram-bot для реалізації застосунку. Також було реалізовано застосунок чат-боту з використанням контексту на базі існуючих даних.

Розроблений чат-бот може успішно експлуатуватися на різних операційних системах таких, як Windows XP, Vista, 7, 8.1, 10, Linux, MacOS, та у актуальних версіях браузераів Chrome, Opera, Mozilla Firefox та Edge використовуючи сервіс Telegram.

Даний застосунок вирішує проблему відсутності базового контексту, що наявна у альтернативних сервісах:

– Dialogflow – під час обробки не потрібно конфігурувати ключові слова, аналізатор синтаксису буде автоматично їх оброблювати.

– AllenNLP – точніший аналіз речень, можливість генерувати структуровані відповіді.

Реалізований застосунок задовільняє заданим вимогам:

– має зрозумілий для користувача інтерфейс чат-боту;

– може проводити синтаксичний аналіз складних речень;

– має зрозумілий код з сучасною архітектурою та коментарями.

В роботі було використано наступні методи дослідження:

– аналітичний метод дослідження використовувався для попереднього аналізу предметної області та проблем, які могли з’явитись під час роботи. Шляхом розкладання об’єкту досліджень на складові частини та аналізу проблем і недоліків кожної, можна зробити висновки про першочергові властивості шуканого результату;

– методи порівняння та оцінювання були використані для виявлення недоліків вже існуючих аналогів та після закінчення розробки системи для виявлення її переваг;

– методи вимірювання та розрахунку були використані для проведення чисельних порівнянь вже існуючих аналогів з поточною системою.

Була визначена актуальність використання цієї системи в загальному використанні чат-ботів для видачі релевантних відповідей користувачеві, на основі діалогу і загального контексту, заданого адміністратором (оператором) чат-бота.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Автоматическая обработка текстов на естественном языке и анализ данных: учеб. пособие / Е. И. Большакова, К. В. Воронцов, Н. Э. Ефремова, Э. С. Клышинский, Н. В. Лукашевич, А. С. Сапин // – М.: Изд-во НИУ ВШЭ, 2017. – 269 с.

2. Машинне навчання – Вікіпедія [Електрон. ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%B5_%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F.

3. Dialogflow [Electronic resourse]. – Access mode: https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html.

4. AllenNLP [Electronic resourse]. – Access mode: https://docs.allennlp.org/master/.

5. Чат-бот – Вікіпедія [Електрон. ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B1%D0%BE%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0).

6. Телеграм – Вікіпедія [Електрон. ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Telegram.

7. Telegram bots documentation [Electronic resourse]. – Access mode: https://core.telegram.org/bots.

8. Фейсбук месенджер – Вікіпедія [Електрон. ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Facebook_Messenger.

9. Facebook bots documenation [Electronic resourse]. – Access mode: https://developers.facebook.com/docs/messenger-platform/introduction.

10. Introduction to sequence-to-sequence learning in Keras [Electronic resource]. – Access mode: https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html.

11. Машинне навчання – [Електрон. ресурс] – Режим доступу: https://www.it.ua/knowledge-base/technology-innovation/machine-learning.

12. Undestanding Machine Learning: From Theory to Algorithms. New York: Cambridge University Press, 2014. – 449 p.

13. Навчання з учителем – Вікіпедія [Електрон. ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B7_%D1%83%D1%87%D0%B8%D1%82%D0%B5%D0%BB%D0%B5%D0%BC.

14. Навчання без учителя – Вікіпедія [Електрон. ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B1%D0%B5%D0%B7_%D1%83%D1%87%D0%B8%D1%82%D0%B5%D0%BB%D1%8F.

15. Навчання без учителя [Електрон. ресурс] – Режим доступу: https://stud.com.ua/139996/informatika/navchannya_vchitelya.

16. Градиентный спуск [Электрон. ресурс] – Режим доступа: https://habr.com/ru/post/307312/.

17. Стохастический градиентный спуск [Электрон. ресурс] – Режим доступа: https://habr.com/ru/post/308604/.

18. Градиентный спуск мини-партий [Электрон. ресурс] – Режим доступа: https://habr.com/ru/post/181456/.

19. Градиентный спуск [Электрон. ресурс] – Режим доступа: https://logic.pdmi.ras.ru/~sergey/teaching/mlhse17/35-dl2-backprop.pdf.

20. Градиентный спуск с импульсом Нестерова [Электрон. ресурс] – Режим доступа: https://www.reg.ru/blog/stehnfordskij-kurs-lekciya-7-obuchenie-nejrosetej-chast-2/.

21. Алгоритм оптимізації цільової функції RMSprop [Електрон. ресурс] – Режим доступу: https://habr.com/ru/post/318970/.

22. Werner V. MXNet – Deep Learning Framework of Choice at AWS / Vogels Werner. [Electronic resource]. – Access mode:

http://www.allthingsdistributed.com/2016/11/mxnetdefault-framework-deep-learning-aws.html.

23. Wilson A. The Marginal Value of Adaptive Gradient Methods in Machine Learning / Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, Benjamin Recht. – 2017. [Electronic resource]. – Access mode: https://arxiv.org/pdf/1705.08292.pdf.

24. Hinton G. E. Neural Networks for machine learning online course/ Geoffrey E. Hinton [Electronic resource]. – Access mode: https://www.coursera.org/learn/neural-networks/home/welcome.

25. Adagrad: Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research [Electronic resource]. – Access mode: http://jmlr.org/papers/v12/duchi11a.html.

26. Christian Igel and Michael H¨usken (2000). Improving the Rprop Learning Algorithm. [Electronic resource]. – Access mode: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.1332

27. Kingma D. P. Adam: a method for stochastic optimization / D. P. Kingma, J. L. Ba // Proceedings of the 3rd International Conference on Learning Representations (ICLR) – 2015. – №1. – P. 1–13.

28. Dozat T. Incorporating nesterov momentum into adam / T. Dozat // ICLR Workshop. – 2016. – № 1. – P. 1– 4.

29. Woollaston V. Google releases TensorFlow – Search giant makes its artificial intelligence software available to the public [Electronic resource]. – Access mode: http://www.dailymail.co.uk/sciencetech/article-3311650/Google-releases-TensorFlow-Search-giant-makes-artificial-intelligence-software-available-public.html.

30. Mitkov R. The Oxford Handbook of Computational Linguistics / R. Mitkov // OUP Oxford, 2004. – 788 p.

31. Oremus W. What Is TensorFlow, and Why Is Google So Excited About It/ W. Oremus. [Electronic resource]. – Access mode:

http://www.slate.com/blogs/future_tense/2015/11/09/google_s_tensorflow_is_open_source_and_it_s_about_to_be_a_huge_huge_deal.html.

32. Brownlee J. Deep Learning for Natural Language Processing / Jason Brownlee // Natural Language Processing. [Electronic resource]. – 2017. – 397 p. – Access mode: http://ling.snu.ac.kr/class/AI_Agent/deep_learning_for_nlp.pdf.

33. Jouppi N. Google supercharges machine learning tasks with TPU custom chip / N. Jouppi. [Electronic resource]. – Access mode: https://cloudplatform.googleblog.com/2016/05/Googlesupercharges-machine-learning-tasks-with-custom-chip.html.

34. Chollet F. Keras Documentation. [Electronic resource]. – Access mode: https://keras.io/.

35. Keras [Электрон. ресурс] – Режим доступа: https://habr.com/ru/post/482126/.

36. Bergstra J. Theano: A CPU and GPU Math Expression Compiler / J. Bergstra, O Breuleux, F Bastien, P Lamblin, R Pascanu, G Desjardins, ... //Proceedings of the Python for scientific computing conference (SciPy). – 2010. – №4 (3). – P. 1-7.

37. Библиотеки для глубокого обучения Theano/Lasagne [Электрон. ресурс] – Режим доступа: https://habr.com/ru/company/ods/blog/323272/.

38. Collobert R. Torch7: A Matlab-like Environment for Machine Learning / R. Collobert, K. Kavukcuoglu, C. Farabet. // Neural Information Processing Systems. - 2011. – №1. – P. 1-6.

39. PyTorch — ваш новый фреймворк глубокого обучения [Электрон. ресурс] – Режим доступа: https://habr.com/ru/post/334380/.

40. Caffe [Electronic resource]. – Access mode: https://caffe.berkeleyvision.org/.

41. MXNet [Electronic resource]. – Access mode: https://mxnet.apache.org/versions/1.7.0/api.

42. Рекуррентные нейронные сети (RNN) с Keras [Электрон. ресурс] – Режим доступа: https://habr.com/ru/post/487808/.

43. Andrej Karpathy (2017). A Peek at Trends in Machine Learning. [Electronic resource]. – Access mode: https://medium.com/@karpathy/a-peek-at-trends-in-machine-learning-ab8a1085a106 .

44. LSTM – сети долгой краткосрочной памяти [Электрон. ресурс] – Режим доступа:https://habr.com/ru/company/wunderfund/blog/331310/.

45. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation / Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio. [Electronic resource]. – Access mode: https://arxiv.org/pdf/1406.1078.pdf.

46. Sequence to Sequence Learning with Neural Networks / Ilya Sutskever, Oriol Vinyals, Quoc V. Le. [Electronic resource]. – Access mode: https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

47. A Gentle Introduction to Neural Machine Translation [Electronic resource] Access mode: https://machinelearningmastery.com/introduction-neural-machine-translation/.

48. Математична лінгвістика // Великий тлумачний словник сучасної української мови / Уклад. та голов. ред. В.Т. Бусел. – К.; Ірпінь: ВТД «Перун», 2004. – С. 513.

49. Natural Language Toolkit [Electronic resource] Access mode: https://www.nltk.org/.

50. SpaCy – NLPub. [Electronic resource]. – Access mode: https://nlpub.mipt.ru/SpaCy.

51. Morphology & Dependency Trees. [Electronic resource]. – Access mode: https://cloud.google.com/natural-language/docs/morphology.

52. Beysolow T. Applied Natural Language Processing with Python / T. Beysolow. – New York: Apress, 2018. – 158 p.

53. Python – Вікіпедія. [Electronic resource]. – Access mode: https://uk.wikipedia.org/wiki/Python.

54. python-telegram-bot [Electronic resource]. – Access mode: https://github.com/python-telegram-bot/python-telegram-bot.

55. Правила улаштування електроустановок : ПУЕ-2017. – [Чинний від 2017-07-21]. – К.: Міненерговугілля України, 2017. – 617 с. – (Національний стандарт України).

56. Державні санітарні правила и норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин : ДСанПіН 3.3.2.007-98. – [Чинний від 1998-12-10]. – К.: Міністерство охорони здоров'я України, 1998. – 9 с. – (Національний стандарт України).

57. Правила ппалення, вентиляція та кондиціонування : ДБН В.2.5-67:2013 – [Чинний від 2014-01-01]. – К.: Міністерство охорони здоров'я України, 1999. – 140 с. – (Національний стандарт України).

58. Природне і штучне освітлення : ДБН В.2.5-28-2006 – [Чинний від 2006-10-01]. – К.: Міністерство охорони здоров'я України, 1999. – 140 с. – (Національний стандарт України).

59. Норми визначення категорій приміщень, будинків та зовнішніх установок по вибухопожежної та пожежної безпеки : НАПБ Б.03.002-2007. – [Чинний від 2007-12-03]. – К.: МНС України, 2007. – 9 с. – (Національний стандарт України).

60. Правила охорони праці під час експлуатації електронно-обчислювальних машин : НПАОП 0.00-1.28-10. – [Чинний від 2010-03-26]. – К.: Державний комітет України з промислової безпеки, охорони праці та гірничого нагляду, 2010. – 8 с. – (Національний стандарт України).

61. Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці : НПАОП 0.00-4.12-05. – [Чинний від 2007-11-16]. – К.: Державний комітет України з промислової безпеки, охорони праці та гірничого нагляду, 2007. – 6 с. – (Національний стандарт України).

ДОДАТОК А

Текст програми

А.1 Код файла «__init__.py»

```
from os.path import dirname, basename, isfile
import glob
modules = glob.glob(dirname(__file__)+"/*.py")
__all__ = [basename(f)[:-3] for f in modules if isfile(f) and not f.endswith('__init__.py')]
```

А.2 Код файла « ctx.py»

```
from client.core.bot import command
from services import core, context
from services.core import learn_from_preset


@command('ctx')
def nani(_, ctx):
    response = ''
    if ctx.message.text == '/ctx':
        learn_from_preset()
    else:
        message = ctx.message.text.replace('/ctx', '').strip()
        nodes = core.process_to_context(message)
        for n in nodes:
            response += n.to_str() + '\n'

    response += f"\nCTX is {str(len(core.context.nodes))}\n"
    ctx.message.reply_text(response)

from client.core.bot import command
from services.nlp import analyze_sentense, to_template
```

А.3 Код файла « nlp.py»

```
@command('nlp')
def nlp(_, ctx):
    message = ctx.message.text.strip().replace('/nlp ', '')
    response = ''
    doc = analyze_sentense(message)
    # noun_chunks = list(doc.noun_chunks)
    response += '---- TOKENS ----\n'

    for token in doc:
        response += f"{token.text} ({token.lemma_}): {token.dep_} – {token.pos_}\n"
```

```
    # response += '\n---- NOUN CHUNKS ----\n'
    # for nc in noun_chunks:
    #     response += nc.text + '\n'

    response += '\n---- TEMPLATE ----\n'
    response += to_template(message)
    ctx.message.reply_text(response)

from typing import List, Union
```

А.4 Код файла «context.py»

```
class ContextNode:
    subj: str = None
    dep: str = None
    value: str = None
    modifier: Union[str, None] = None

    def __init__(self, dep: str):
        self.dep = dep

    def to_str(self):
        return f"{self.subj} ({self.dep}) – {self.modifier} : {self.value}"


class Context:
    nodes: List[ContextNode] = list()

    def find_or_create_node(self, subj: str, dep: str) -> ContextNode:
        for index, node in enumerate(self.nodes):
            if node.subj == subj and node.dep == dep:
                return node
        node = ContextNode(dep)
        node.subj = subj
        self.nodes.append(node)
        return node

    def set(self, subj: str, dep: str, value: str):
        node = self.find_or_create_node(subj, dep)
        node.value = value

    def update_node(self, node: ContextNode):
        cnode = self.find_or_create_node(node.subj, node.dep)
        cnode.modifier = node.modifier
```

```
            cnode.value = node.value

        def get_node(self, key, modifier) -> Union[ContextNode, None]:
            for i, node in enumerate(self.nodes):
                if node.subj == key and modifier == node.modifier:
                    return node
            return None
```

A.5 Код файла «core.py»

```
from spacy.tokens import Span, Doc, Token

from config import CONTEXT_PRESET
from services.context import Context, ContextNode
from services.nlp import analyze_sentense, get_sign, to_template, to_nodes
from services.neural import Seq2SeqNeuralService
import re

from utils.file_reader import read_lines_from_file

context = Context()


def process_response(sentence: str):
    nns = Seq2SeqNeuralService()
    model = nns.get_nn()
    sent_template = to_template(sentence)
    resp_template = model.decode_sequence(sent_template).strip()
    print(resp_template)
    return parse_context(sentence, resp_template)


def process_to_context(sentence: str):
    doc: Doc = analyze_sentense(sentence)
    # ents: List[Span] = list(doc.ents)
    nodes = to_nodes(doc)
    for node in nodes:
        context.update_node(node)
    return nodes


def learn_from_preset():
    lines = read_lines_from_file(CONTEXT_PRESET)
    for line in lines:
        process_to_context(line)
```

```
def find_node(subj, matches) -> ContextNode:
    if matches:
        for i, m in enumerate(matches):
            if m == '%subj%':
                continue
            match_mod = m.replace('%', '', 2)
            return context.get_node(subj, match_mod)


def parse_context(orig: str, template: str) -> str:
    doc = analyze_sentense(orig)
    response = template
    subj = None
    for token in doc:
        if token.dep_ == 'nsubj':
            subj = token.text
            break

    if subj == None:
        return '*No question subject found!'
    match = re.findall(r'%\w+%', template)

    context_node = find_node(subj, match)
    if context_node != None:
        for (i, m) in enumerate(match):
            if m != '%subj%':
                response = response.replace(m, context_node.value)
        response = response.replace('%subj%', context_node.subj)
    else:
        return '*Failed to answer!'

    return response

learn_from_preset()
```

ДОДАТОК Б

Слайди презентації

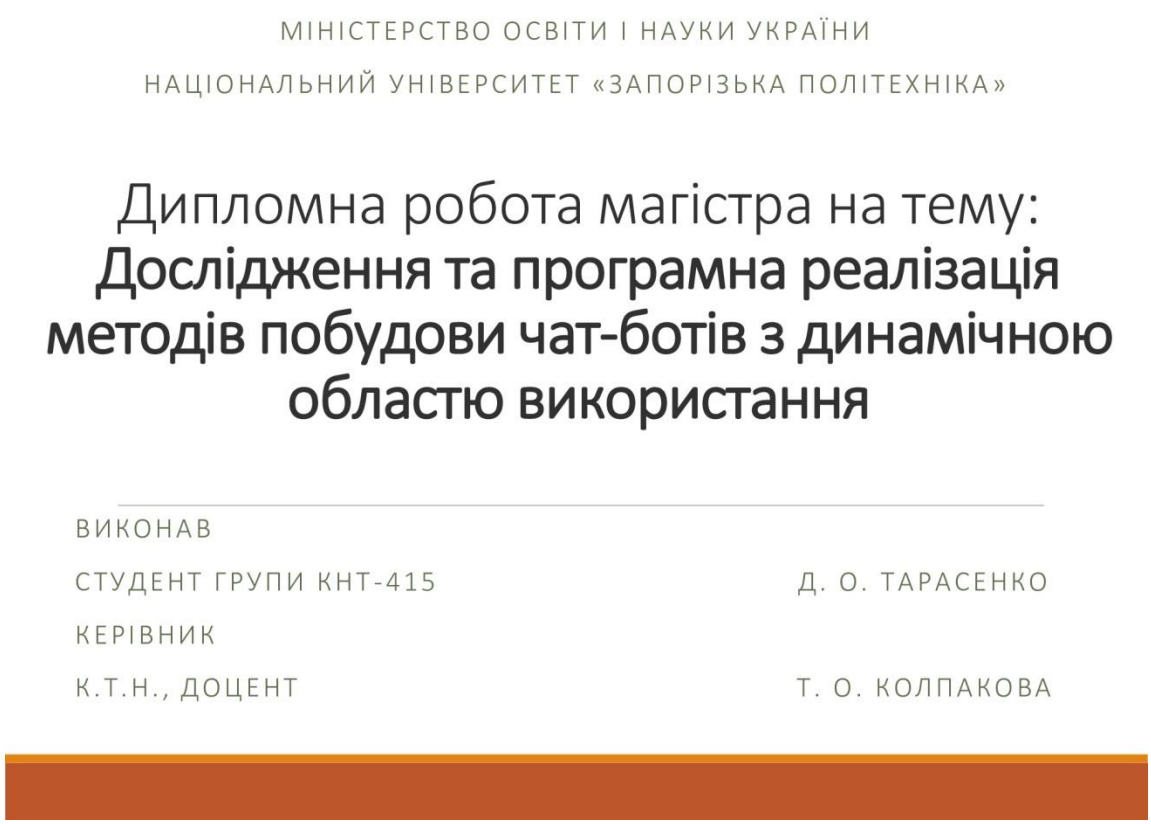

Рисунок Б.1 – Слайд №1

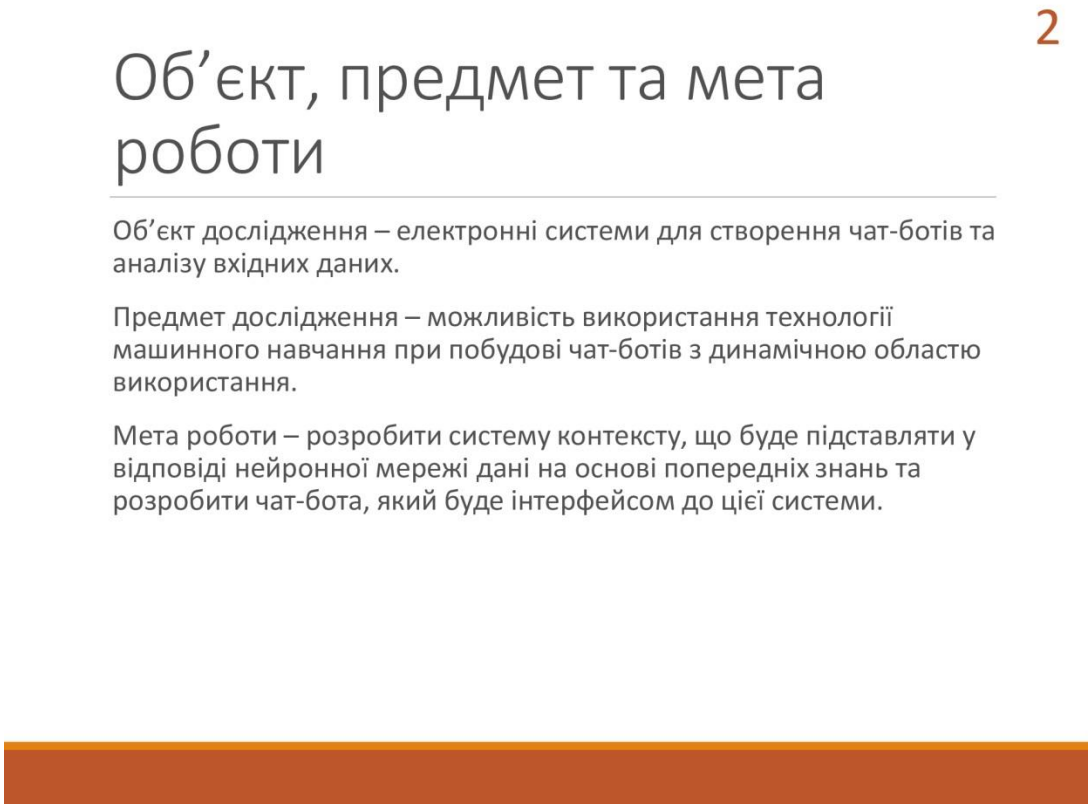

Рисунок Б.2 – Слайд №2

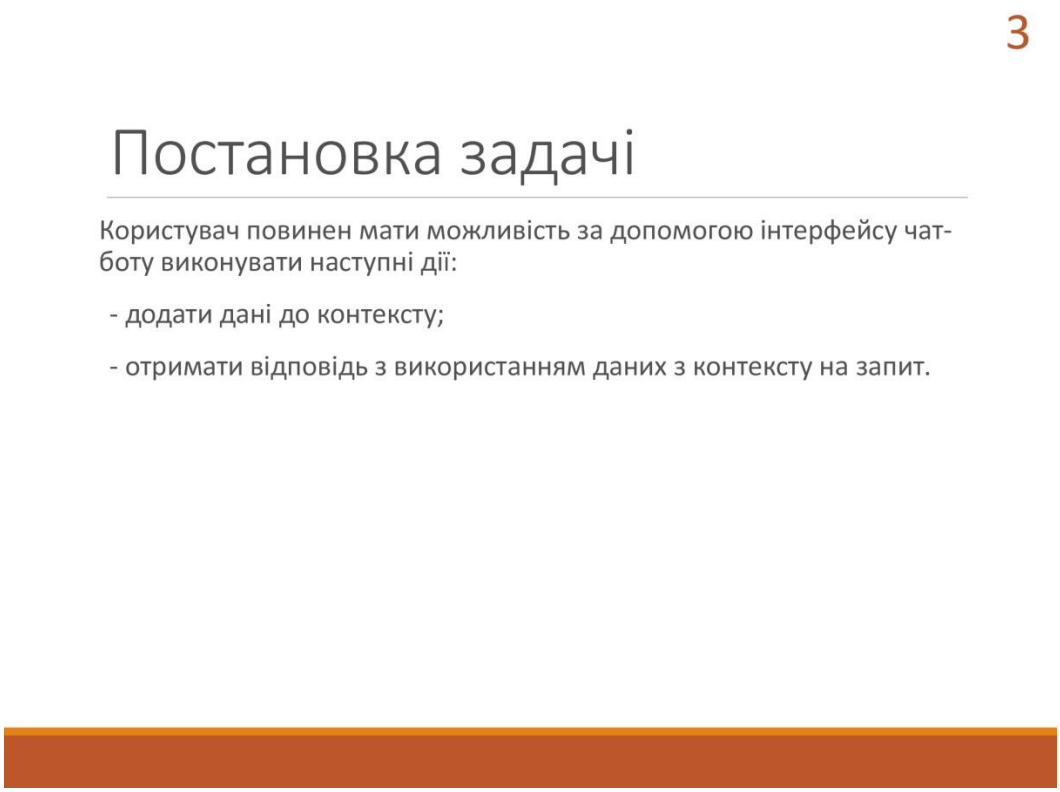

Рисунок Б.3 – Слайд №3

4

Аналоги

Назва	Безкоштовний	Відкритий код	Може оброблювати нові формати даних
Dialogflow	+	+	-
AllenNLP	+	+	-

Рисунок Б.4 – Слайд №4

5

Порівняння фреймворків глибинного навчання

Назва	Відкрите	Платформа	Підтримка CUDA	Рекурентні мережі	Згорткові мережі	Паралельне виконання
Apache Singa	Так	Linux, Mac OS X, Windows, Android (багатоплатформне)	Так	Так	Так	Так
Caffe	Так	багатоплатформне	Так	Так	Так	?
Deeplearning4j	Так	багатоплатформне	Так	Так	Так	Так
Dlib	Так	багатоплатформне	Так	Ні	Так	Так
Keras	Так	багатоплатформне	Так	Так	Так	Так
Microsoft Cognitive	Так	Windows, Linux	Так	Так	Так	Так
Neural Designer	Ні	багатоплатформне	Ні	Ні	Ні	?
Open NN	Так	багатоплатформне	Ні	Ні	Ні	?
TensorFlow	Так	багатоплатформне	Так	Так	Так	Так
Theano	Так	багатоплатформне	Так	Так	Так	Так
Torch	Так	багатоплатформне	Так	Так	Так	Так
Mathematica	Ні	Багатоплатформне, хмарні обчислення	Так	Так	Так	Так

Рисунок Б.5 – Слайд №5

6

Схема роботи застосунку

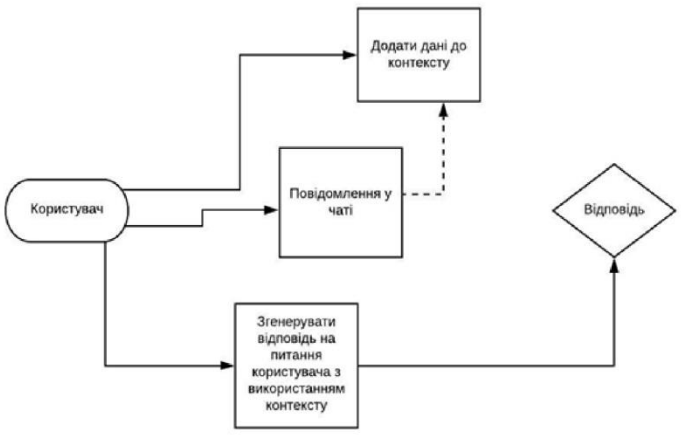

Рисунок Б.6 – Слайд №6

7

Архітектура застосунку

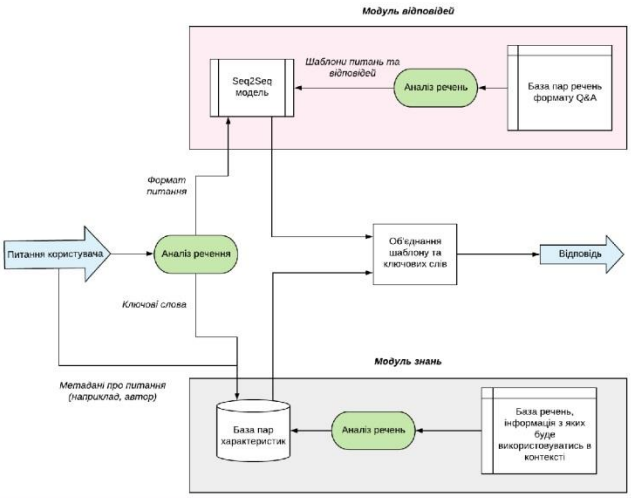

Рисунок Б.7 – Слайд №7

8

LSTM & Seq2Seq

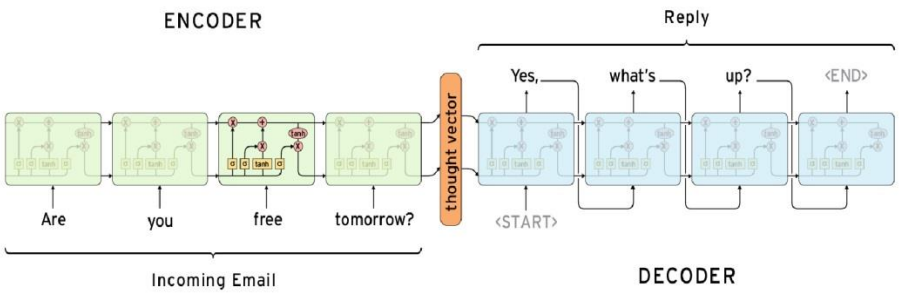

Рисунок Б.8 – Слайд №8

9

Приклад аналізу речення

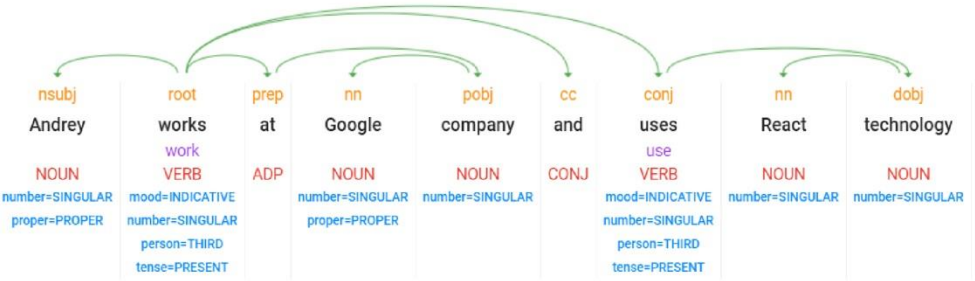

Рисунок Б.9 – Слайд №9

10

Приклад аналізу речення

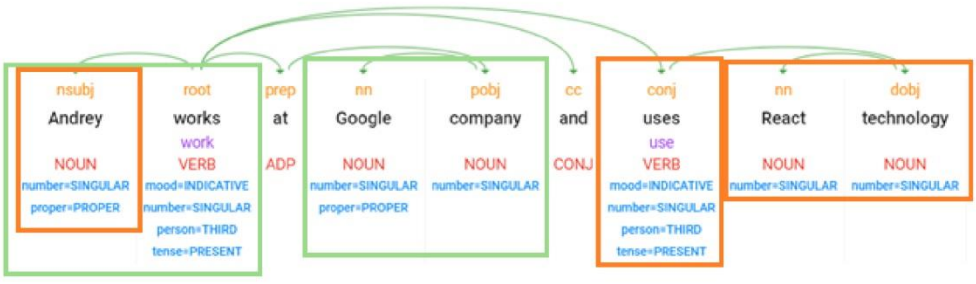

Рисунок Б.10 – Слайд №10

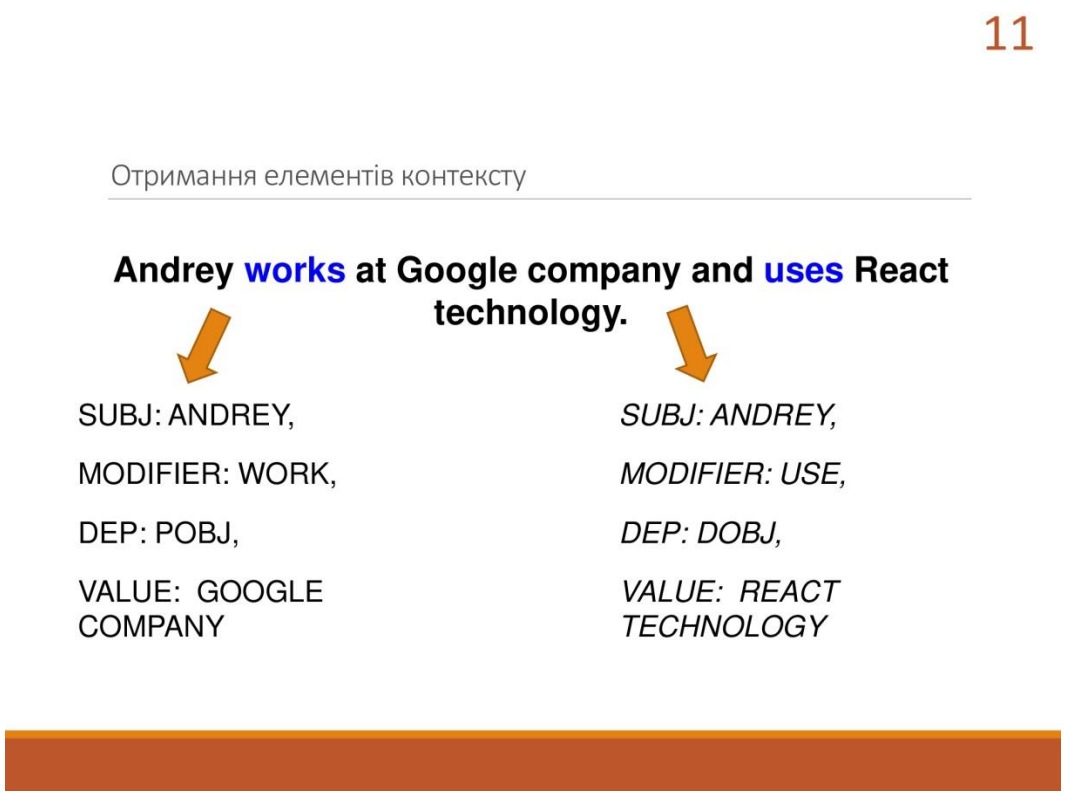

Рисунок Б.11 – Слайд №11

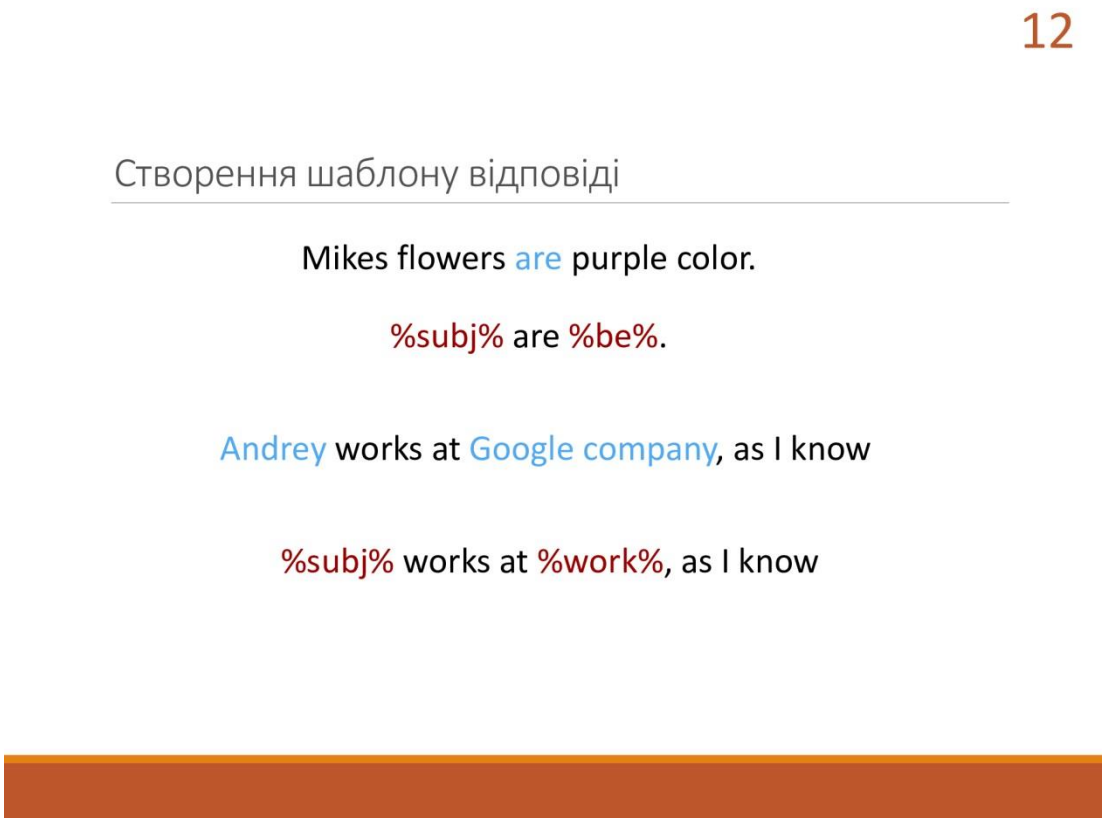

Рисунок Б.12 – Слайд №12

13

Приклад генерації відповідей та роботи контексту

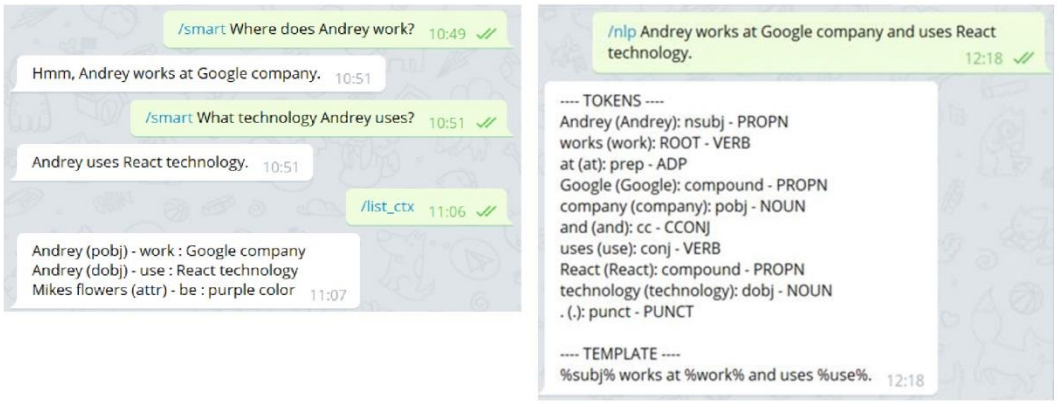

Рисунок Б.13 – Слайд №13

14

Висновки

Досліджено предметну область моделей рекурентних нейронних мереж та алгоритми NLP.

Проаналізовано проблему жорсткої залежності від чітко визначеної структури вхідних даних моделі рекурентної нейронної мережі.

Запропоновано використання можливостей контексту в зв'язці з нейронною мережею для генерації відповідей.

Створено застосунок направлений на аналіз повідомлень та коротких текстів для генерації відповідей.

Розроблений застосунок відповідає наступним умовам:

- має зрозумілий для користувача інтерфейс чат-боту;
- може проводити синтаксичний аналіз складних речень;
- має зрозумілий код з сучасною архітектурою та коментарями.

Рисунок Б.14 – Слайд №14

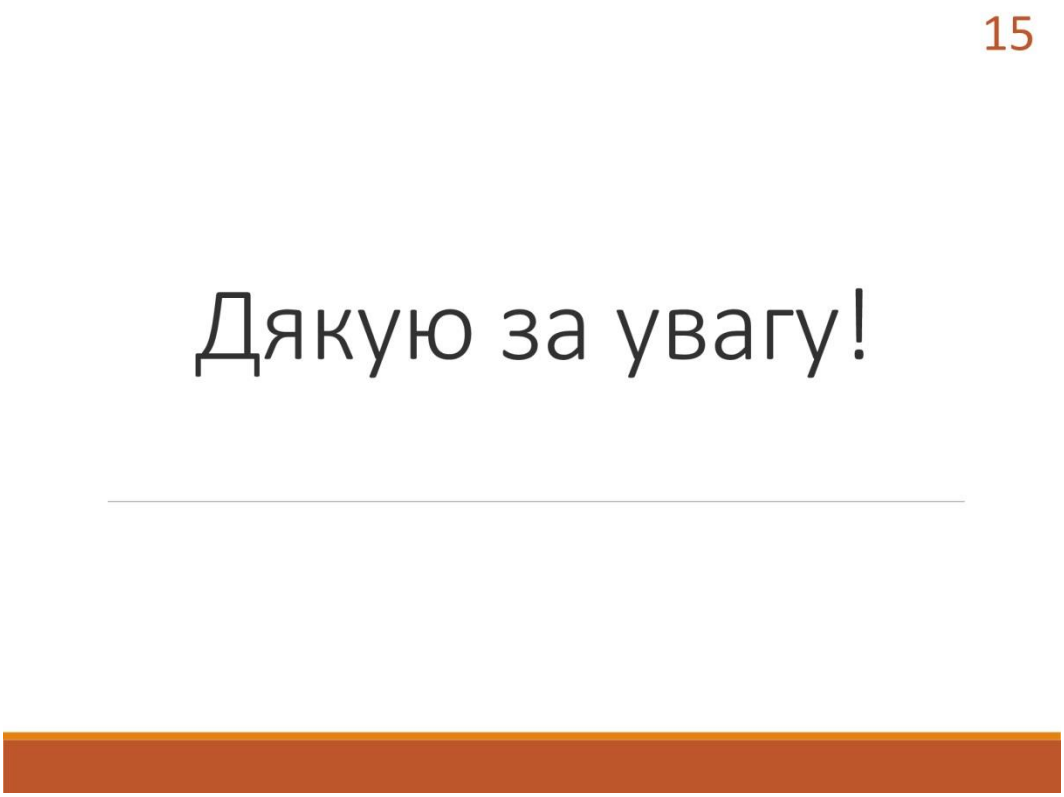

Рисунок Б.15 – Слайд №15