ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розроблення веб-застосунку голосового асистента з використанням технології штучного інтелекту»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
освітньо-професійної програми Штучний інтелект

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Сергій ОЛІЩУК
(підпис)

Виконав:
здобувач вищої освіти Сергій ОЛІЩУК
група ШІД-41

Керівник: Максим ФЕСЕНКО
к.т.н., доцент

Рецензент:
_____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Штучного інтелекту
Ступінь вищої освіти Бакалавр
Спеціальність 122 Комп'ютерні науки
Освітньо-професійна програма Штучний інтелект

ЗАТВЕРДЖУЮ
Завідувач кафедрою Штучного інтелекту

_____ Ольга ЗІНЧЕНКО
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Оліщуку Сергію Олександровичу

1. Тема кваліфікаційної роботи: Розроблення веб-застосунку голосового асистента з використанням технології штучного інтелекту.

керівник кваліфікаційної роботи Максим ФЕСЕНКО к.т.н., доцент,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» 02.2024р. № 36

2. Строк подання кваліфікаційної роботи «31» травня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Дослідження історії розвитку NLP, логістичної регресії, систем синтезу та розпізнавання мови.
Аналіз принципу роботи NLP, логістичної регресії, систем синтезу та розпізнавання мови.
Розробка системи голосового асистента з використанням технологій штучного інтелекту.
Аналіз інструментів веб-розробки.
Створення веб-застосунку голосового асистента на основі розробленої системи.

5. Перелік графічного матеріалу: *презентація*

1. Використані технології для розробки веб-застосунку голосового асистента з використанням штучного інтелекту.
2. Демонстрація створеного веб-застосунку.

6. Дата видачі завдання«27» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Дослідження еволюції технологій штучного інтелекту, а саме NLP, логістична регресія, синтез та розпізнавання природної мови.	27.02–03.03.24	Виконав
2	Аналіз принципів роботи NLP, логістичної регресії, синтезу та розпізнавання мови.	04.03–17.03.24	Виконав
3	Розробка системи голосового асистента з використанням технологій штучного інтелекту	18.03–16.04.24	Виконав
4	Дослідження основних технологій веб-розробки	17.04–22.04.24	Виконав
5	Розробка веб-застосунку	23.04-30.04.24	Виконав
6	Інтеграція розробленої системи у веб-застосунок	01.04-15.05.24	Виконав
7	Тестування веб-застосунку	16.05-20.05.24	Виконав
8	Оформлення роботи: вступ, висновки, реферат	21.05-27.05.24	Виконав
9	Створення демонстраційних матеріалів	28.05-31.05.24	Виконав

Здобувач вищої освіти _____ Сергій ОЛІЩУК

Керівник
кваліфікаційної роботи _____ Максим ФЕСЕНКО

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра 62 сторінки, 16 рис., 1 табл., 21 джерело.

Мета роботи – підвищення ефективності використання технологій штучного інтелекту, таких як NLP, логістична регресія, синтез та розпізнавання мови.

Об'єкт дослідження – процес розробки веб-застосунку голосового асистента.

Предмет дослідження – система голосового асистента.

Короткий зміст роботи: у роботі проведено дослідження еволюції технологій штучного інтелекту, таких як NLP, логістична регресія, синтез та розпізнавання мови, а також проаналізовано їх принципи роботи. Було досліджено можливості фреймворку Django для Python для розробки веб-застосунків. Розроблено прототип системи голосового асистента з використанням штучного інтелекту. Також використано засоби веб-розробки, такі як HTML, CSS, JavaScript для клієнтської частини і фреймворк Django для серверної частини, для створення веб-застосунку голосового асистента на основі розробленої системи.

КЛЮЧОВІ СЛОВА: СИНТЕЗ, РОЗПІЗНАВАННЯ, МОВА, NLP, РЕГРЕСІЯ, HTML, CSS, DJANGO, PYTHON, ВЕБ-ЗАСТОСУНОК, ВЕБ-ДОДАТОК.

ЗМІСТ

ВСТУП

Сучасний світ дуже стрімко розвивається. За останні декілька років відбулась глобальна цифровізація у всіх аспектах життя. Технології штучного інтелекту також стали не від'ємною частиною життя та займають дедалі важливіше місце у різних його сферах, створюючи нові можливості для автоматизації та оптимізації процесів, вони роблять наше життя більш легким. Одним із найбільш перспективніших напрямків розвитку технологій штучного інтелекту є створення голосових асистентів, які значно полегшують взаємодію користувача з інформаційними системами, використовуючи природну мову для обміну інформацією. Вони дозволяють сказати щось просто голосом, а телефон чи комп'ютер виконає те, що ви сказали.

Голосові асистенти вже продемонстрували свою ефективність у виконанні широкого спектру завдань, від простих пошукових запитів до управління розумними пристроями. Також, після появи інтелектуальних мовних моделей, з'явилася можливість спілкуватися з асистентом в режимі реального часу і він буде відповідати так, ніби це людина. Інтеграція таких моделей у голосові асистенти відкриває нові горизонти для створення інтерактивних сервісів.

Використання голосових асистентів у веб-застосунках не лише підвищує зручність для кінцевого користувача, але й відкриває нові можливості для бізнесу та сервісів, які хочуть надати більш інтерактивні та персоналізовані послуги. Інтеграція таких асистентів у веб-застосунки дозволяє значно розширити функціональність останніх, розробляючи їх доступнішими для широкого кола користувачів, включаючи людей з обмеженими можливостями, які можуть мати труднощі з використанням традиційних інтерфейсів.

Інтерес до створення спеціалізованих веб-застосунків з використанням голосових асистентів зростає також через розвиток інтернет технологій та збільшення кількості підключених пристроїв. Це створює необхідність у створенні нових підходів до взаємодії з користувачами, де голосові команди і аудіо відповіді можуть стати ключовими елементами. Веб-застосунки, що підтримують голосових

асистентів, мають потенціал стати важливим інструментом у багатьох сферах, від електронної комерції до освітніх платформ та систем охорони здоров'я.

Метою даної дипломної роботи є підвищення ефективності використання технологій штучного інтелекту, таких як обробка природної мови, логістична регресія, синтез та розпізнавання мови, через розроблення веб-застосунку голосового асистента. Об'єктом дослідження є процес розробки веб-застосунку голосового асистента. Предметом дослідження є система голосового асистента.

Для досягнення мети були поставлені наступні задачі:

1. Вивчення історії появи технологій розпізнавання та синтезу мови, NLP, логістичної регресії.
2. Вивчення принципу роботи технологій розпізнавання та синтезу мови, NLP, логістичної регресії для правильного їх використання при розробці голосового асистента.
3. Аналіз роботи фреймворку Django для Python для розробки веб-застосунку.
4. Реалізація системи голосового асистента.
5. Створення веб-застосунку голосового асистента.

Актуальність теми розробки голосового асистента з використанням технологій штучного інтелекту обумовлена швидким розвитком цих технологій та впровадженням їх у різні аспекти життя людини. Голосові асистенти дозволяють користувачам отримувати швидкий та зручний доступ до інформації та послуг, що є особливо важливим в умовах сучасного суспільства. Розробка такого веб-застосунку дозволяє не лише підвищити ефективність взаємодії користувачів з системами, але й розширити можливість існуючих сервісів.

Для досягнення поставленої мети були використані такі методи дослідження:

1. Аналіз літературних джерел – дослідження наукових статей, книг, онлайн ресурсів та інших публікацій, присвячених темі голосових асистентів та технологій штучного інтелекту.
2. Практичне моделювання та розробка – створення веб-застосунку, його програмування та тестування.

3. Емпіричні дослідження – проведення експериментів з використанням веб-застосунку.

1 АНАЛІЗ ТЕОРЕТИЧНИЙ ВІДОМОСТЕЙ

1.1 Історія розвитку систем розпізнавання мови

За своє довге існування технології розпізнавання мовлення пройшли довгу дорогу, щоб зараз з легкістю та неймовірною швидкістю розпізнавати людську мову. Перші системи розпізнавання мови були сфокусовані на цифрах, а не на словах. Враховуючи важкість мови, це і не дивно, що інженери спочатку сфокусували свою увагу на цифрах. Найперша система розпізнавання мови була створена 1952 році компанією Bell Laboratories. Вона називалась "Audrey" та могла розпізнавати цифри, сказані одним словом. Десять років потому, у 1962 році, IBM представила світу "Shoebox" систему, яка уже могла розпізнавати 16 слів англійською в додаток до цифр від 0 до 9.[3]

По всьому світу інші країни також розробляли обладнання, що могло розпізнавати звук та мову. До кінця 60-х років технологія могла підтримувати слова з чотирма голосними та дев'ятьма приголосними звуками. Хоча і цим системам далеко до сучасних, але вони дали вражаючий старт, особливо якщо врахувати те, що комп'ютери того часу були достатньо примітивними.

У 1970-х роках були зроблені декілька значимих досягнень у розпізнаванні мови. Це сталось здебільшого через Міністерство оборони США та DARPA. Програма дослідження розуміння мовлення, яку вони проводили, була однією з найбільших у в історії розпізнавання мови. Система "Harpy", розроблена університетом Карнегі-Меллон, могла розуміти більше тисячі слів, що приблизно рівняється те саме, що словниковий запас трьох річної дитини. "Harpy" була значним проривом, адже вона використовувала більш ефективних підхід пошуку, який називається Beam search. [14]

Також важливим є те, що в 70-х роках минулого сторіччя була основана перша комерційна компанія Threshold Technology, яка представила світу систему, яка могла інтерпретувати різні голоси.[3]

У 1980-х роках, використовуючи нові підходи і технології, словниковий запас подібних систем виріс з декількох сотень до декількох тисяч слів, а також уже

мав потенціал розпізнавання необмеженої кількості слів. Однією із причин був новий статичний метод, більш відомий як прихована Маркова модель (HMM). На основі шаблонів для слів і звукові шаблони, вона розглядала можливість того, що невідомі звуки могли бути словами. Ця модель потім використовувалась іншими системами протягом двадцяти років.[18]

З розширеним словниковим запасом системи розпізнавання мови почали використовувати і в комерційних додатках для бізнеса та спеціалізованих галузей, таких як медицина. Вони навіть ввійшли в будинки людей у 1987 році у вигляді ляльки "Words of Wonder's Julie doll", яку діти могли натренувати, щоб вона розпізнавала їх голос.[3]

Хоча і програмне забезпечення могло розпізнавати до 5000 слів, але в них був величезний недолік - для того, щоб вони могли зрозуміти сказане, людина повинна була зупинятись після кожного слова, щоб програма могла її опрацювати.

У 90-х роках минулого століття комп'ютери отримали швидкі процесори і системи для розпізнавання мови стали більш життєздатними.

У 1990 році з'явилась перша загальнодоступна програма Dragon Dictate, яка коштувала 9000 доларів. Через 7 років вийшла її покращена версія Dragon NaturallySpeaking, яка уже коштувала 695 доларів. Додаток міг розпізнавати нормальне мовлення, люди могли говорити в нормальному темпі близько 100 слів в хвилину. Але все одно, потрібно було тренувати програму протягом 45 хвилин перед використанням. [3]

Поява першого голосового порталу VAL від BellSouth відбулось у 1996 році. Це була перша інтерактивна система розпізнавання мови, яка давала інформацію, спираючись на те, що люди казали у трубку телефона.[14] VAL створила дорогу для всіх неточних голосових меню, які використовувались наступні 15 років.

До 2001 року розпізнавання мови піднялось до 80 відсоткової точності і прогрес технологій зупинився. Системи розпізнавання працювали відмінно, поки мовний всесвіт був обмежений, але вони досі "догадувались" за допомогою статичної моделі серед схожих слів, а мовний всесвіт ріс разом з ростом Інтернету.

Технології розпізнавання мовлення отримали друге дихання після створення додатку Google Voice Search для iPhone. Вплив цього додатку був значним через те, що, по-перше, мобільні девайси - це ідеальний об'єкт для розпізнавання мовлення і бажання замінити маленьку екранну клавіатуру іншим способом введення було дуже великим, а, по-друге, у компанії Google була можливість розгрузити і прискорити цей процес, використовуючи свої хмарні дата-центри, спрямовуючи їх можливості для великомасштабного аналізу даних для пошуку збігів між величезною кількістю записаних зразків голосових запитів, які вони отримували і словами користувачів. [3]

Ахіллесовою стопою систем розпізнавання мовлення завжди була доступність даних і ефективність їх обробки. Додаток, в свою чергу, додав до аналізу мільярди пошукових запитів, щоб краще передбачати, що сказала людина.

У 2010 році компанія Google додала персональне розпізнавання мовлення в голосовий пошук для телефонів під управлінням Android. Застосунок записував голосові запити користувачів, які потім використовувались для покращення голосової моделі. В середині 2011 року компанія додала можливість розпізнавання голосу у свій браузер Chrome. Система Google зараз дає можливість розпізнавали 230 мільйонів слів.[3]

Сьогодні деякі найбільші технологічні компанії змагаються за те, щоб отримати найбільш точну систему розпізнавання мовлення. У 2016 році IBM змогла досягли похибки лише у 6.9%. У 2017 році Microsoft обігнала IBM з похибкою у 5.9%. Трохи пізніше IBM покращила свій результат - похибка становила 5.5%. Однак Google має найнижчий показник - 4.9%.[3]

1.2 Механізми функціонування систем розпізнавання мови

З моменту створення систем розпізнавання мови пройшло дуже багато років, але все таки ця проблема не вирішена і сьогодні. Вона вирішена для певних ситуацій, але універсального рішення поки не існує. Це відбувається через ряд проблем, з якими стикаються розробники таких систем:

1. залежність від домену:

1.1. різні диктори;

1.2. "акустичний" канал запису звука: кодеки, спотворення;

1.3. різне оточення: шум в телефоні, в місті, фонові голоси;

1.4. різний темп і підготовка мови;

1.5. різна стилістика та тематика мови;

2. великі і не стандартизовані набори даних;

3. не до кінця зрозуміла метрика якості.

Для початку варто розібратись з типами систем розпізнавання мовлення. Такі системи бувають двох типів: гібридні і end2end. End2end системи перекладають послідовність звуків в послідовність букв. У них є перевага над гібридними системами: рідковживані слова, а також власні назви розпізнаються точніше. Гібридні ж системи мають акустичну та мовну модель, які працюють незалежно один від одного. Акустична модель розбирає звук на фрейми, а потім витягує з них букви, а мовна модель - з'єднує ці букви у слова і перевіряє їх наявність в своєму словнику та повертає найбільш відповідне слово. Така модель працює значно швидше ніж end2end, але слова, яких немає в словнику, не розпізнаються. [11]

Так як голосовий асистент повинен розпізнавати команди та відповідати швидко, то використання гібридного типу розпізнавання мови буде доцільнішим. Тому далі у підрозділі буде розповідатись про нього.

Принцип роботи гібридної системи розпізнавання мови:[11]

1. Нейронна мережа класифікує кожен конкретний фрейм (відрізок аудіо заданої довжини) звуку.

2. HMM (hidden Markov model) моделює "динаміку", "лексикон" та "лексику" спираючись на апостеріорі нейронної мережі.

3. Алгоритм Viterbi (Viterbi decoder, beam-search) займається пошуком по HMM оптимального шляху, з урахуванням апостеріорі класифікатора.

Загальну схему побудови гібридної системи розпізнавання мови можна побачити на рисунку 1.1.

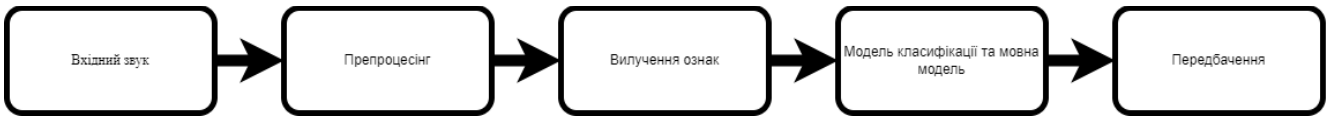

Рис. 1.1. Загальна схема побудови гібридної системи розпізнавання мови.

Першим кроком в гібридний системі розпізнавання мови виділяються ознаки. Як правило, це дрібно частотні кепстральні коефіцієнти. Потім акустична модель вирішує задачу класифікації фреймів. Далі використовується Viterbi-decoder (пошук по променю). Він використовує передбачення акустичної моделі і статистику мовної моделі, яка по ngram показує ймовірність зустрічальності звуків та слів. Потім відбувається оновлення оцінок і видається найбільш імовірне слово.[11] Схему влаштування гібридної архітектури систем розпізнавання мови можна побачити на рисунку 1.2.

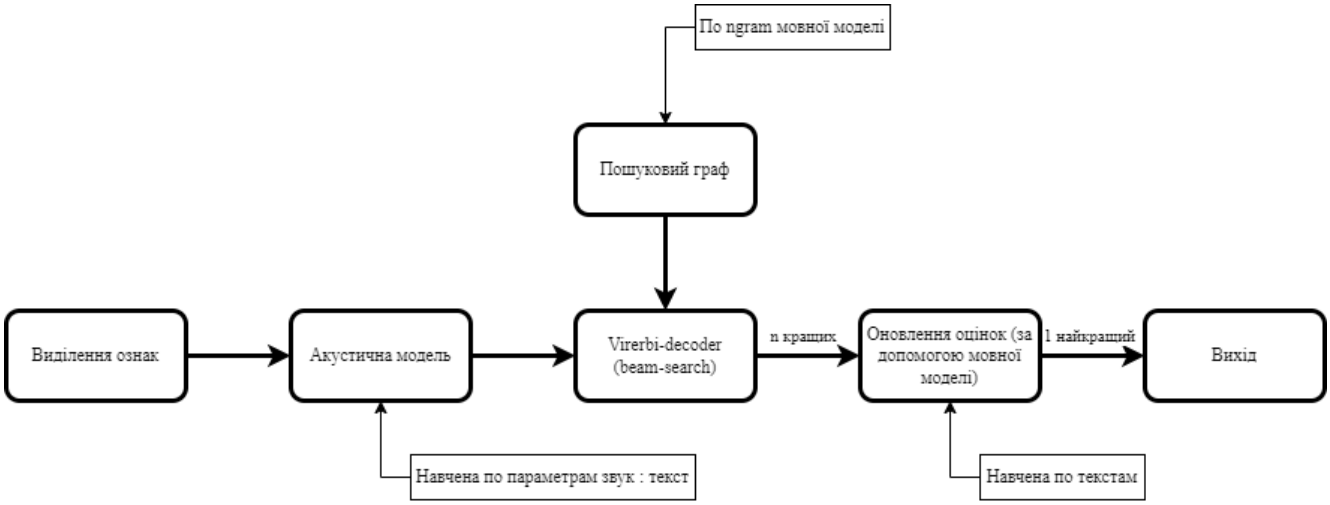

Рис. 1.2. Влаштування гібридної архітектури систем розпізнавання мови.

На рисунку 1.3 продемонстрована розділення аудіо слова «так» на фрейми.

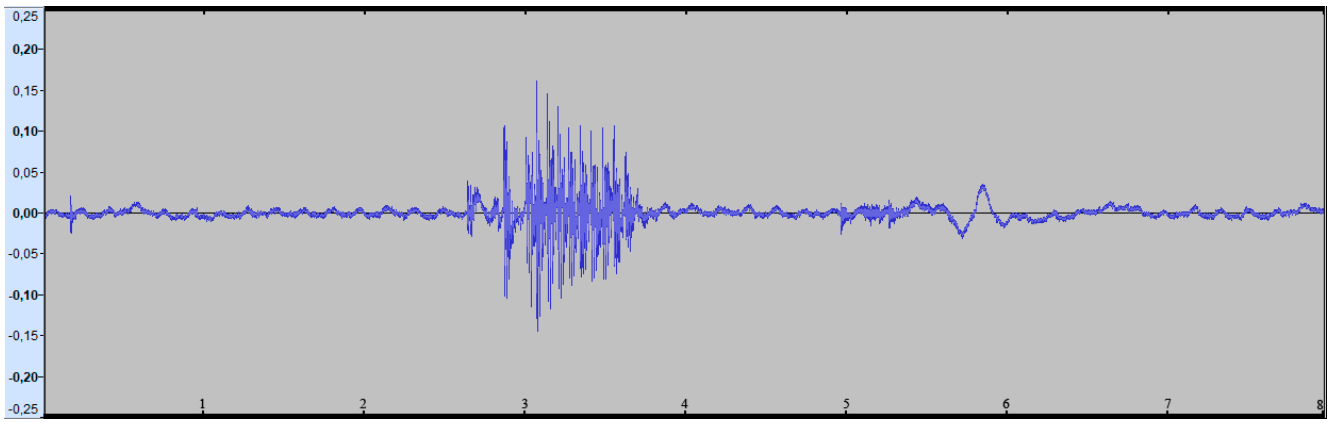

Рис. 1.3. Ілюстрація аудіо слова «так», розділене на фрейми.

Ймовірності для кожної фонеми у фреймах записані у відповідні клітинки таблиці 1.1.

Таблиця 1.1

Ймовірності розпізнавання фонем у фреймах

№ фрейма / Фонема	1	2	3	4	5	6	7	8
Т	0	0.2	0.8	0.2	0.2	0.1	0	0
А	0	0	0.1	0.8	0.2	0.1	0	0
К	0	0.1	0.2	0.1	0.7	0.5	0.1	0
Тиша (SIL)	1	0.6	0.1	0	0.1	0.3	0.8	1

Далі розглянемо принцип роботи обчислювального графа на прикладі зі словом «так». Нехай у першому фреймі класифікатор виявив фонему «т» і так 10 разів підряд. Такий цикл буде виконуватись поки не з’явиться наступна фонема – «а». У свою чергу, такий цикл буде виконуватись до тих пір, поки не з’явиться фонема «к». Він зупиниться, коли виявить фонему «SIL», тобто тиша. І якщо слово міститься у словнику – воно буде записано. На рисунку 1.4 наочно проілюстровано даних приклад.

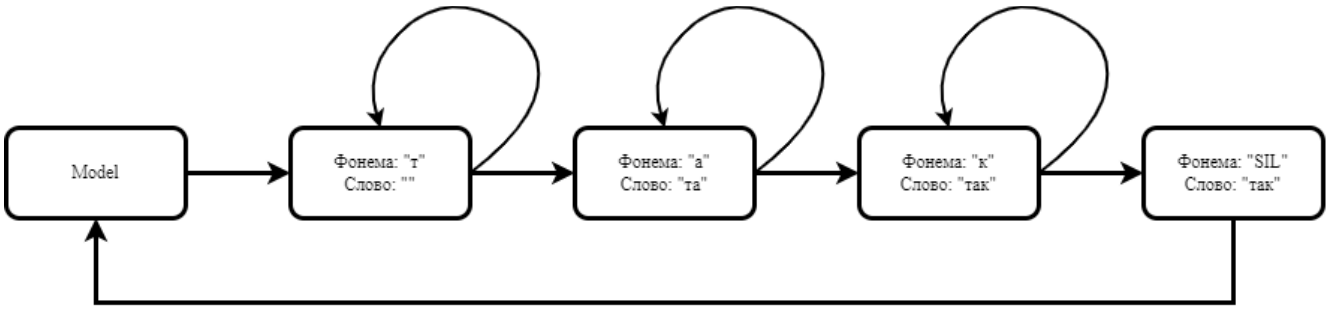

Рис. 1.4. Візуалізація принципу роботи пошукового графа.

1.3 Історія розвитку систем синтезу мови

У кінці 1950-х років, була створена перша система синтезу людської мови. Це була перша система на комп'ютерній базі. У 1961 році Джон Ларрі Келлі молодший, фізик у Bell Labs, використав комп'ютер компанії IDM для синтезу мови. Його синтезатор мови відтворив пісню Daisy Bell. [21]

В той час поки Келлі покращував свій синтезатор, Артур К. Кларк, автор книги "2001: Космічна одіссея", використав винахід Келлі в сценарії своєї книги.

У 1966 році з'явилось лінійне кодування мови з прогнозуванням. Таку форму кодування почали розробляти Фумітада Ітакура і Шузо Саіто. Бішну С. Атал і Манфред Р. Шредер також зробили свій внесок у розвиток лінійного прогнозуючого кодування. [21]

У 1975 році Ітакурою був розроблений метод спектральних пар. Цей метод кодування мовлення з високим ступенем стиснення допоміг Ітакурі дізнатися більше про аналіз і синтез мовлення, знайти слабкі місця та з'ясувати, як їх виправити чи покращити.[21]

Протягом цього року також була випущена MUSA. Ця автономна система синтезу мовлення використовувала алгоритм для читання вголос італійською мовою. А версія, яка вийшла 3 роки згодом могла навіть співати італійською.

У 70-х роках минулого століття був розроблений перший артикуляційний синтезатор, заснований на голосовому тракті людини. Перший відомий синтезатор був розроблений Томом Бером, Полом Мермельштейном і Філіпом Рубін у Haskins Laboratories. Це тріо використовувало інформацію з моделей голосового тракту, створених у Bell Laboratories у 60-70-х роках.[21]

У 1976 році були представлені читальні машини Kurzweil для сліпих. Хоча ці пристрої були надто дорогими для широкого загалу, але бібліотеки часто надавали їх людям із вадами зору, щоб вони могли слухати книги.

Лінійне кодування з прогнозуванням стало відправною точкою для мікросхем синтезатора. Іграшки Texas Instruments LPC Speech Chips, і іграшки Speak & Spell кінця 1970-х років використовували технологію мікросхем синтезаторів. Ці іграшки були прикладами синтезу людського голосу з точними інтонаціями, що відрізняло голос від синтезованих голосів того часу, які зазвичай звучали роботами. Багато кишенькової електроніки зі здатністю синтезувати мову стали популярними протягом цього десятиліття, включаючи калькулятор Telesensory Systems Speech+ для сліпих. Fidelity Voice Chess Challenger, шаховий комп'ютер, який міг синтезувати мову, був випущений у 1979 році.[21]

В 1980-х роках синтез мови почав з'являтись навіть у світі відеоігор. Випуск 1980 року Stratovox (аркадна гра в стилі стрільби) був випущений компанією Sun Electronics. Manbiki Shoujo (переклад "Магазинна дівчина") була першою персональною комп'ютерною грою з можливістю синтезу мовлення. Електронна гра Milton, що була випущена у 1980 році, - це була перша електронна гра The Milton Bradley Company, яка могла синтезувати людський голос. У 1983 році був створений автономний акустико-механічний мовний апарат під назвою DECtalk. DECtalk розумів фонетичне написання слів, дозволяючи налаштовувати вимову незвичайних слів. Ці фонетичні написання можуть також включати індикатор тону, який DECtalk використовуватиме під час виголошення фонетичних компонентів. Це дозволило DECtalk співати.[21]

У пізніх 80-х, Стів Джобс створив NeXT - систему, яка була розроблена Trillium Sound Research. Незважаючи на те, що NeXT не злетів, Джобс зрештою об'єднав програму з Apple у 90-х.

У 1999 році компанія Microsoft випустила Narrator - програму зчитування з екрану, яка зараз є в кожній копії Microsoft Windows.

Синтез мовлення зіткнувся з деякими проблемами протягом 2000-х, оскільки розробники намагалися створити узгоджені стандарти для синтезованого мовлення. Оскільки мовлення дуже індивідуальне, людям у всьому світі важко об'єднатися й домовитися про правильну вимову фонем, дифонів, інтонації, тону, відтворення шаблонів і флексії. [21]

Якість мовного аудіо синтезу формантів також стала більшою проблемою в 90-х роках, оскільки інженери та дослідники помітили, що якість систем, які використовуються в лабораторії для відтворення синтезованого мовлення, часто була набагато кращою, ніж обладнання, яке було у користувача. Думаючи про синтез мовлення, багато людей згадують синтезатор голосу Стівена Хокінга, який створював роботизоване звучання голосу з невеликим людським відтінком.

У 2005 році дослідники нарешті прийшли до якоїсь згоди і почали використовувати звичайні мовні дата сети, які дозволяли їм працювати з однаковими базовими ідеями, при створенні високорівневих систем синтезу мови.

У 2007 році було проведено дослідження, яке показало, що слухачі можуть легко визначити, чи посміхається людина, яка говорить. Дослідники продовжували працювати над визначенням того, як створити системи розпізнавання та синтезу мови, які були б більш природніми.

Сьогодні продукти, які використовують системи синтезу мови, всюди. Електроні синтезатори мови не тільки роблять світ легшим - вони також роблять його веселішим. Люди використовуються TTS системи для прослуховування улюблених романів в дорозі, а також такі синтезатори полегшують вивчення іноземних мов.

1.4 Механізми функціонування систем синтезу мови

Задачу синтезу мови сьогодні вирішують двома підходами:

- Unit selection або конкатенативний підхід. Він оснований на склеюванні фрагментів записаного аудіо. З кінця 90-х років довгий час він вважався стандартом для розробки систем синтезу мови. Наприклад, голос, який звучить методом unit selection, можна було зустріти в Siri.[21]
- Параметричний синтез мови, суть якого полягає в побудові ймовірнісної моделі, яка може передбачати акустичні властивості аудіо сигналу для даного тексту.[21]

Мова моделей unit selection має високу якість, низьку варіативність і потребує більшого об'єму даних для навчання. В той же час, для тренування параметричних моделей потрібно набагато менше даних, вони генерують більш різноманітні інтонації, але до недавнього часу страждали від достатньо низької якості звуку, порівнюючи з unit selection підходом.[4]

З розвитком технологій глибокого навчання моделі параметричного синтезу суттєво виросли по якості і можуть створювати мову, майже невідмінну від людської.

Розберемось як же все таки працює синтез мови. Перший крок побудови системи синтезу мови - збір даних для навчання. Зазвичай це аудіозаписи високої якості, на яких диктор читає спеціально підібрані фрази. Приблизний розмір дата

сету, який потрібен для навчання моделей unit selection, складає приблизно 10-20 годин чистого мовлення, в той час як для параметричних моделей верхня оцінка дорівнює приблизно 25 годинам.

Для початку розглянемо як працює unit selection метод. Зазвичай записана мова диктора не може покрити всіх можливих випадків, в яких буде використовуватись синтез. Тому суть методу полягає в тому, щоб розбити всю аудіо базу на невеликі фрагменти, які називаються юнітами, які потім склеюються один з одним з використанням мінімальної пост обробки. В якості юнітів зазвичай виступають мінімальні акустичні одиниці мови, такі як напівфони або дифони.[4]

Весь процес генерації складається із двох етапів: NLP frontend, який відповідає за вилучення лінгвістичного представлення тексту, і backend, який обчислює функцію штрафу юнітів для заданих лінгвістичних ознак. В NLP frontend входять:

1. Задача нормалізації тексту - переклад всіх небуквених символів (цифр, знаків відсотків, валют і так далі) в їх словесне представлення. Наприклад, "5 %" має бути перекладено в "п'ять відсотків".
2. Вилучення лінгвістичних ознак із нормалізованого тексту: фонемне представлення, наголошення, частини мови і таке інше.

Зазвичай NLP frontend реалізований за допомогою вручну прописаних правил для конкретної мови, але останні роки відбувається все більший уклон в бік використання моделей машинного навчання.

Штраф, який оцінює backend підсистема - це сума відповідності акустичного представлення юніта для конкретної фонеми і доречності з'єднання двох сусідніх юнітів. Для оцінки штрафу функцій можна використовувати правила або уже навчену акустичну модель параметричного синтезу. Вибір найбільш оптимальної послідовності юнітів з точки зору вище вказаних штрафів відбувається за допомогою алгоритму Вітербі.

Схематичний принцип роботи такого методу проілюстровано на рисунку 1.5.

Рис. 1.5. Принцип роботи unit selection методу.

Переваги unit selection підходу:

- природність звучання;
- висока швидкість генерації;
- невеликий розмір моделей - це дозволяє використовувати синтез прямо на мобільному пристрої;

Недоліки такого підходу:

- синтезована мова монотонна, не містить емоцій;
- характерні артефакті склейки;
- потребує достатньо великої тренувальної бази аудіо даних для покриття всіх можливих контекстів;
- не може генерувати звук, який не зустрічався в навчальній виборці;

Тепер перейдемо до параметричного методу синтезу мови. В основі параметричного підходу лежить ідея про побудову імовірнісної моделі, що буде оцінювати розподіл акустичних ознак заданого тексту [4]. Процес генерації мовлення в параметричному синтезі можна розділити на 4 етапи:

1. NLP frontend - така сама стадія початкової обробки даних, як і у unit selection підході, результат якої - велика кількість контекстно-залежних лінгвістичних ознак.
2. Duration model, яка буде передбачати довжину фонем.
3. Акустична модель, яка буде відновлювати розподіл акустичних ознак по лінгвістичним. До акустичних ознак входять значення фундаментальної частоти, спектрального представлення сигналу і таке інше.
4. Вокодер, який буде перекладати акустичні ознаки у звукову хвилю.

Для навчання duration і акустичних моделей можна використовувати приховані Марківські моделі, глибокі або рекурентні нейронні мережі. Традиційний вокодер - це алгоритм, заснований на source-filter моделі, яка припускає, що мовлення - це результат застосування лінійного шуму до початкового сигналу. [4]

Загальна кількість параметричних методів виявляється достатньо низькою через велику кількість незалежних припущень про будову процесу генерації звуку.

Однак з приходом технологій глибокого навчання стало можливим навчати end-to-end моделі, які можуть напряму передбачати акустичні ознаки по буквам. Також в якості вокодерів також почали використовувати нейронні мережі замість алгоритмів цифрової обробки сигналів. [4]

Переваги параметричного синтезу:

- природнє і плавне звучання при використанні end-to-end підходу;
- більша різноманітність в інтонаціях;
- використання меншого об'єму даних в порівнянні з моделями unit selection.

Недоліки:

- низька швидкість роботи в порівнянні з unit selection;
- велика обчислювальна складність.

1.5 Історія розвитку NLP

Всього 50 років, як людство почало використовувати системи обробки природної мови, але зараз ми використовуємо найрізноманітніші моделі NLP в різних пошукових запитах, перекладачах, чат ботах та інше. NLP з’явилось при злитті штучного інтелекту і лінгвістики (науки, що вивчає мови, їх семантику, фонетику, синтаксис).

Ноам Чомскі був ученим-лінгвістом, який зробив революцію в області лінгвістики і змінив звичайне розуміння синтаксису. Саме його дослідження і дали поштовх до створення систем, які могли б обробляти людську мову. [9]

NLP можна розділити на дві великі частини: NLU - Natural Language Understanding і NLG - Natural Language Generating. Першою задачею, яку потрібно було вирішити вченим 1950-х років було саме навчити машину розуміти текст і робити висновки з матеріалу, який їй дали.

Першим використанням обробки природної мови був машинний переклад. Ціллю було створення машини, яка зможе перекладати людську мову чи текст. Перші кроки в цій області були зроблені Джорджтаунським університетом і компанією IBM Company. Програма змогла перекласти 60 речень з однієї мови на іншу. Цей прорив дав представлення про те, як мають розвиватися майбутні технології.[9]

В кінці 60-х років Террі Виноград із Массачусетського технологічного інституту розробив SHRDLU - програму обробки природної мови. Вона могла відповідати на питання і враховувати при цьому нові фактори про світ. SHRDLU могла поєднувати важкий синтаксичний аналіз з достатньо загальною дедуктивною системою. Машина могла відповідати на прості питання і здавалось, що якщо витратити достатньо зусиль на передачу суті і обмежити себе деякою областю, SHRDLU зможе досягти природної комунікації. Але цей ранній підхід мав свої недоліки, які не дозволяли розвивати його далі.[9]

Далі у 1969 році, Роджер Шанк розробив концептуальну систему залежностей, яку він описав як "стратифікована лінгвістична система, яка дозволяє надати обчислювальну теорію модельованої продуктивності". Ця концепція створення лексем, які дозволяли отримувати з тексту більше сенсу. Ці лексеми могли містити різні об'єкти реального світу. Комбінація токенів в різних аспектах покликана врахувати всю сукупність мовної діяльності на концептуальному рівні. Якщо користувач говорить, що з конструкцією все добре, вона додається в пам'ять, в іншому випадку конструкція шукається в списку метафор або переривається. Таким чином, система використовує запис того, що вона чула раніше, для аналізу того, що вона чує зараз.[9]

В своїй роботі Роджер вхопився за ідею про те, що до того, як комп'ютери почнуть розуміти природню мову, вони мають навчитись приймати рішення про те,

що саме їм говорять. Синтаксичний аналізатор вченого був орієнтований на семантику мови. Він зміг навчити комп'ютер розрізняти важливі концептуальні відносини.[9]

У 1970-х роках, Вільям Вудс розробив свою систему розпізнавання і обробки тексту, він представив доповнену мережу переходу (ATN). На основі якої пізніше була розроблена програма LAS, яка дозволяла створювати класи слів і розуміти правила формування речень. ATN дозволяла не тільки формувати нові речення, а ще розуміти саму мову. Програма формувала класи слів. Вона обіцяла бути такою ж адаптивною, як і людина. Вивчаючи новий матеріал людина знайомилась з новою лексикою, новими синтаксичними конструкціями, щоб поділитись новою інформацією. LAS була написана на мові LISP, вона дозволяла отримувати на вхід декілька рядків, які вона описувала як сцени, закодовані у вигляді асоціативних мереж. Таким чином програма могла підкорюватись командам, розуміти, писати, вчитись. Ядром всієї системи була граматика доповненої мережі ATN.[9]

До появи алгоритмів машинного навчання у 1980-х роках вся обробка природної мови зводилась до рукописним і не автоматизованим правилам. Тим не менше, ще до того часу з'явились перші ідеї про створення машин, які могли би працювати схоже до людського мозку, через нейронні зв'язки. Це стало прообразом штучного інтелекту, побудованого на нейронних мережах майбутнього.[9]

З початку 21 століття і до сьогодні, розвиток машинного навчання почав набирати обороти. Нові методи і підходи по обробці слів створюються і сьогодні, що робить дослідження NLP актуальним досі.

1.6 Базові концепції та методи NLP

Обробка природної мови включає в себе комп'ютерну лінгвістику, машинне навчання і моделі глибокого навчання. Розглянемо кожен аспект.

Комп'ютерна лінгвістика - це наука про розуміння і побудову моделей людського мовлення за допомогою комп'ютерів і програмних інструментів. Дослідники використовують методи комп'ютерної лінгвістики, такі як синтаксичний і семантичний аналіз, для створення платформ, які допомагають

машинам розуміти розмовну мову. Перекладачі, синтезатори мовлення і системи розпізнавання мови: всі вони основані на комп'ютерній лінгвістиці.

Машинне навчання - технологія, яка навчає комп'ютер за допомогою вибіркових даних. Людське мовлення має декілька особливостей, наприклад, сарказм, метафори, варіації в структурі речень, на вивчення яких у людей йдуть роки. Програмісти використовують методи машинного навчання, щоб навчити комп'ютер розпізнавати і точно розуміти функції з самого початку.

Глибоке навчання - це область машинного навчання, яка використовує багатошарові штучні нейронні мережі для аналізу і обробки даних. З допомогою цього методу комп'ютери мають змогу розпізнавати, класифікувати і знаходити закономірності у вхідному тексті.

Для створення систем NLP збираються неструктуровані дані, які потім оброблюються.

Обробка відбувається за допомогою наступних методів:

1. токенізація;
2. лематизація;
3. стемінг;
4. видалення стоп-слів.

Потім на основі оброблених даних з використанням машинного навчання для тренування моделей NLP, щоб потім на основі цього можна було аналізувати тексти.

Розглянемо, що можна робити використовуючи уже навчені моделі NLP.

По-перше, оброблений такою системою текст можна токенізувати по реченням, тобто розділити його на речення. Ідея виглядає достатньо простою, але насправді, це не зовсім так. Кожне речення закінчується крапкою, але вона використовується ще й для скорочень. В такому випадку дуже допомагає таблиця всіх можливих скорочень, яка являється невід'ємною частиною систем NLP. Вона дозволяє уникнути неправильного розміщення кордонів речення.

Також системи NLP дозволяють токенізувати текст на менші одиниці - на слова. Ця задача також здається простою, адже між словами має бути пробіл, по

якому можна розділяти, але це не зовсім так. Наприклад, в англійській мові складові іменники пишуться по різному і інколи через пробіл. Навчені системи NLP у цьому випадку також допомагають.

Наступним важливим аспектом є лематизація. Це процес приведення слова у його базову словникову форму. Наприклад, у тексті можуть зустрічатися слова "бігати", "бігла", "біжать", людина відразу розуміє, що це все одне і те саме слово у різних його формах. Системи NLP дозволяють лематизувати їх до спільної форми "бігати".

Також схожим методом на попередній є стемінг - це процес відрізання зайвого від кореня слова, що часто призводить до втрати словотворчих суфіксів. При лематизації використовується також контекст в якому було вжито слово, що дозволяє відрізняти омоніми один від одного. При стемінгу контекст не враховується, тому такий підхід працює швидше, а точність не завжди може мати значення.

Також навчені системи NLP мають свій список стоп-слів. Це такі слова, які додають певний шум у текст, хоча і не міняють його сенс. Стоп-слова це зазвичай артиклі, вигуки, сполучники і таке інше.

І, на мою думку, найважливіше, що дозволяють навчені системи NLP - встановлювати зв'язки між словами. Розглянемо на прикладі речення "Штучний інтелект - це майбутнє людства.". На рисунку 1.6 можна побачити зв'язки між словами, де:

- amod - дієприкметниковий модифікатор;
- nsubj - іменний предмет;
- expl - вставне слово;
- nmod - іменний модифікатор.

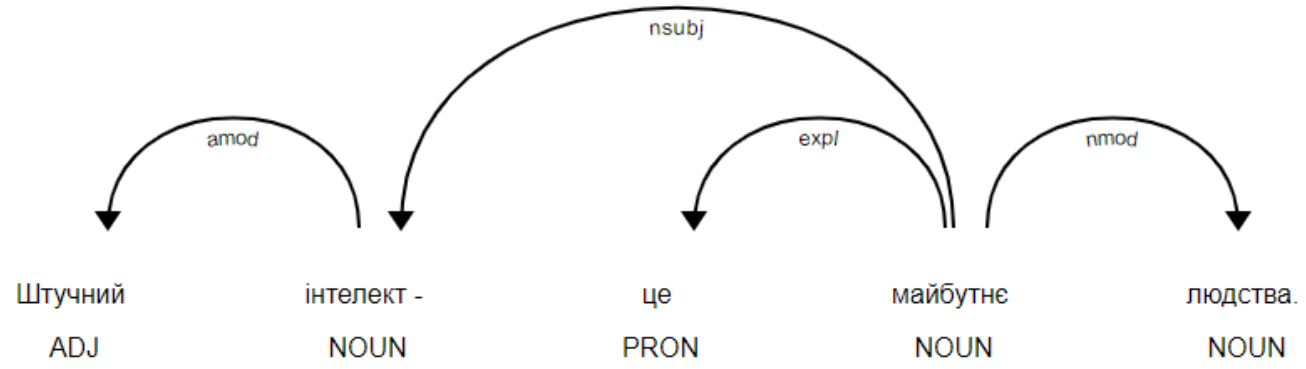

Рис. 1.6. Ілюстрація зв’язків між словами у реченні

Все вище сказане дозволяє розробити продукт, який зможе, наприклад, розуміти і виконувати команди.

1.7 Логістична регресія

Логістична регресія являється одним із статичних методів класифікації з використанням лінійного дискримінанта Фішера. Також вона входить в топ самих використовуваних алгоритмів в статистиці.[15]

На відміну від звичайної лінійної регресії, в методі логістичної регресії не відбувається передбачення значень числової змінної виходячи із вибірки вхідних значень. Замість цього, значенням функції являється ймовірність того, що дане вхідне значення відноситься до певного класу. Для простоти, давайте представимо, що у нас є тільки два класи, і ймовірність, що вхідні дані відносяться до першого класу, дорівнює P, таким чином імовірність, що дані відносяться до другого класу, дорівнює Q, де:

$$Q = 1 - P \tag{1.1}$$

Таким чином, результат логістичної регресії завжди знаходиться в інтервалі від 0 до 1.

Основна ідея логістичної регресії заключається в тому, що простір вхідних значень може бути розділений лінійною границею (для двовимірного простору - лінією, для тривимірного - площиною і так далі), в нашому випадку, на дві відповідні класам області. Ця границя задається в залежності від вхідних даних і

алгоритму навчання. Якщо точки вхідних даних задовольняють це вимогу, то їх називають лінійно подільними[15]. Розглянемо рисунок 1.7. Вказана розділяюча площина називається лінійним дискримінатором, так як вона являється лінійною з точки зору своєї функції, і дозволяє моделі виконувати розділення, дискримінацію точок на різні класи.

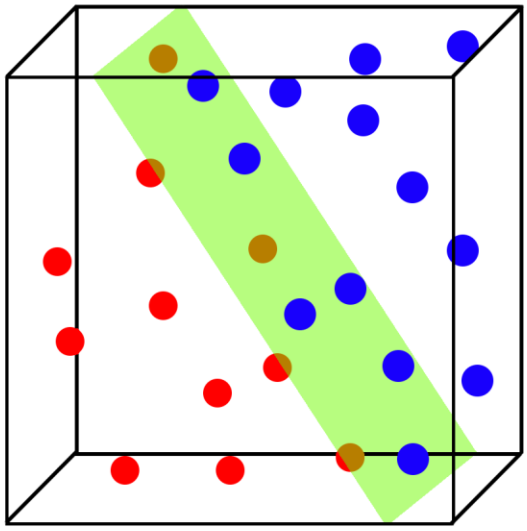

Рис. 1.7. Ілюстрація класифікації об'єктів лінійним дискримінантом

Розглянемо як відбувається розділення об'єктів у логістичній регресії. По-перше, потрібно зрозуміти геометричний зміст розділення вхідного простору на дві області. Для простоти візьмемо дві вхідні змінні x_1 і x_2, тоді функція, яка відповідає границі, буде мати вигляд:

$$F(x_1; x_2) = n_0 + n_1 x_1 + n_2 x_2 \quad (1.2)$$

, де n_0, n_1, n_2 - коефіцієнти прямої,

x_1, x_2 - вхідні змінні.

Розглянемо точку (a, b). Підставляючи її замість x_1, x_2 у формулу (1.2) отримаємо певний результат F(a,b). В залежності від положення точки (a, b) потрібно розглянути 3 варіанти:

- Точка (a, b) лежить в області вище лінії лінійного дискримінатора. Тоді результат F(a,b) буде додатнім і чим більше його величина, тим більше відстань між дискримінатором і точкою, а це в свою чергу означає більшу ймовірність того,

що точка (a, b) належить класу, що знаходиться над лінією. Відповідно ймовірність P буде знаходитися в інтервалі (0.5, 1].

- Точка (a, b) лежить в області нижче лінії лінійного дискримінатора. Тоді результат F(a,b) буде від'ємним і чим більше його абсолютна величина, тим більше відстань між дискримінатором і точкою, а це в свою чергу означає більшу ймовірність того, що точка (a, b) належить класу, що знаходиться під лінією. Відповідно ймовірність Q буде знаходитися в інтервалі [0, 0.5).
- Точка (a, b) лежить на самій лінії дискримінатора. В такому випадку результат F(a,b) буде дорівнювати 0. Це означає, що модель не може визначити до якого класу належить точка (a,b), в результаті P = Q = 0.5.

Отже, ми маємо функцію, за допомогою якої можна отримати значення в межах (-∞; +∞) маючи точку вхідних даних. Для того, щоб отримати шукану ймовірність P, межі якої [0; 1] потрібно використати функцію відношення шансів(OR):

$$OR(X) = \frac{P(X)}{1-P(X)} \tag{1.3}$$

, де P(X) - це ймовірність події X.

Очевидно, що P(X) знаходиться в межах [0;1], тоді OR(X) буде знаходитись у межах (0; +∞]. Це значить, що потрібна ще одна дія, так як використана гранична функція видає значення в межах (-∞; +∞). Потрібно обчислити натуральний логарифм від OR(X), що називається логарифмом відношення шансів, що буде в межах (-∞; +∞).

Таким чином, ми отримали спосіб інтерпретації результатів, підставлених у граничну функцію вхідних даних. На двовимірному прикладі, при наявності точки (a, b) алгоритм логістичної регресії буде виглядати наступним чином:

1. Обчислити значення граничної функції (1.2).
2. Обчислити відношення шансів:

$$OR = e^{F(x_1;x_2)} \tag{1.3}$$

3. Маючи значення OR, обчислити ймовірність P:

$$P = \frac{OR}{1+OR} \tag{1.4}$$

Залишилось тільки зрозуміти, як навчається гранична функція. Розглянемо функцію g(x), де x - точка даних навчальної вибірки. Дана функція проводить кількісну оцінку ймовірності того, що точка навчальної вибірки класифікується моделлю правильним чином. Тому середнє значення для всієї навчальної вибірки показує ймовірність того, що випадкова точка даних буде коректо класифікована системою незалежно від можливого класу. Механізм навчання логістичної регресії старається максимізувати середнє значення функції g(x). Назва такого методу навчання - метод максимальної правдоподібності.[15]

1.8 Django – фреймворк для розробки веб-застосунків на Python

Django - безкоштовний фреймворк для веб-застосунків, написаний на мові Python. Він був створений у 2005 році, коли веб-розробники з газети Lawrence Journal-World почали використовувати Python для створення веб-сайтів. У 2008 році вийшов перший публічний реліз фреймворку. Django продовжує розвиватись і сьогодні, актуальною стабільною версією являється версія 5.0.6. Кожен новий реліз фреймворку виходить в середньому кожні 8 місяців. Крім того, постійно виходять обновлення і виправлення проблем безпеки. Крім того, є можливість використовувати більш старі версії (на даний момент це версія 4.2), адже деякі з них являються більш стабільними, їх весь час доповнюють і у них менше помилок.[5]

Django являється достатньо популярним. Він використовується на багатьох сайтах, наприклад, Pinterest, Instagram, Washington Times і інші.

Фреймворк являється безкоштовним і розвивається як open source, тобто його вихідний код відкритий.

Майже кожного разу при розробці веб-додатків потрібно розробляти схожі компоненти: вхід, вихід, реєстрацію користувачів, панель управління сайтом, інструменти для загрузки файлів і так далі. Django ж по замовчуванню пропонує готову функціональність для таких компонентів, завдяки чому можна не винаходити велосипед, а достатньо взяти уже готові рішення.

В Django велика увага приділяється безпеці, завдяки чому фреймворк допомагає розробникам уникнути багатьох поширених проблем в системі безпеки, наприклад, SQL-ін'єкцій.

Фреймворк реалізує архітектурний шаблон Model-View-Template (MVT), який являється модифікацією популярного в веб-програмуванні шаблону Model-View-Controller (MVC).[5]

Схематичну ілюстрацію архітектури MVT в Django можна побачити на рисунку 1.8.

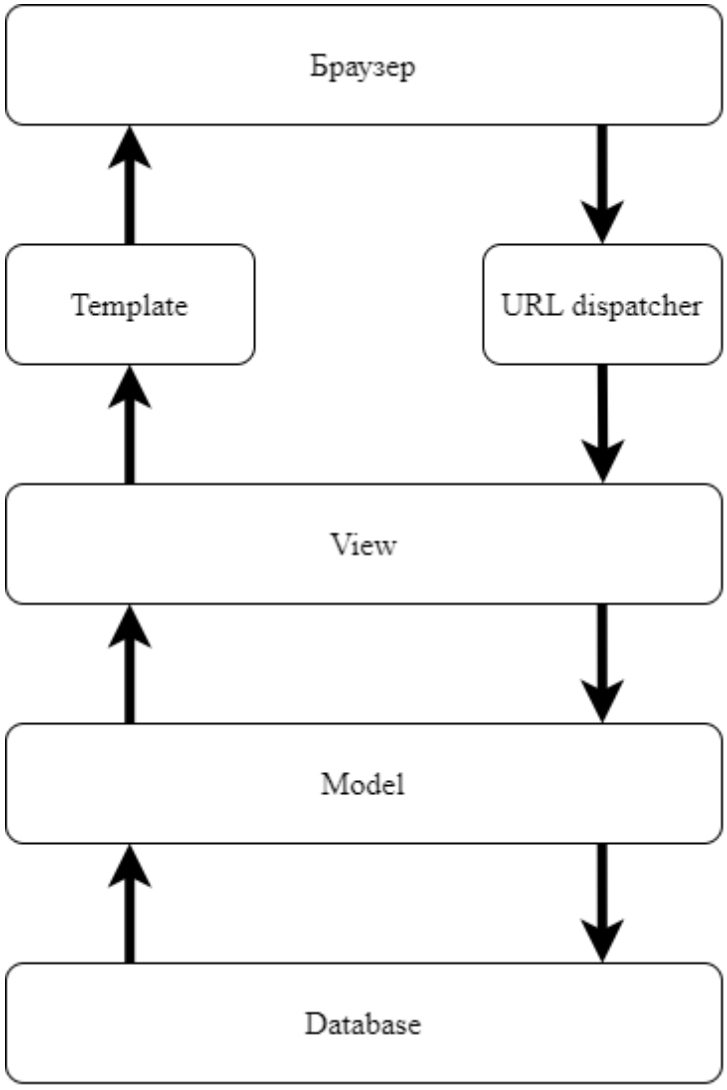

Рис. 1.8. Ілюстрація архітектури MVT в Django

Основні елементи шаблона:

- URL dispatcher: при отриманні запиту на основі запиту адреси URL, визначає, який ресурс повинен обробити даний запит.

- View: отримує запит, обробляє його і відправляє певну відповідь користувачеві. Якщо для обробки запиту потрібно звернення до моделі і бази даних, то View взаємодіє з ними. Для створення відповіді може використовувати Template. В архітектурі MVC цьому компоненту відповідає відповідний контролер.
- Model: описує дані, які використовує веб-додаток. Окремі класи, як правило, відповідають таблицям в базі даних.
- Template: представляє логіку представлення у вигляді згенерованної HTML розмітки. В MVC цьому компоненту відповідає View, тобто представлення.

Коли до веб-застосунку приходить запит, то URL dispatcher визначає, з яким ресурсом зіставляється даний запит і передає його вибраному ресурсу. Ресурс фактично представляє собою функцію або View, який отримує запит і певним чином обробляє його. В процесі обробки View може звертатись до моделей і бази даних, отримувати від неї якісь дані, або навпаки зберігати їх. Результат обробки запиту відправляється назад і цей результат користувач бачить у своєму браузері. Як правило, результат обробки запиту представляє собою згенерований HTML код, для створення якого використовуються шаблони (Template).[5]

Django - це дуже зручний, а головне функціональний фреймворк, для створення backend частини веб-застосунку.

2 ЗАСОБИ РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКУ ГОЛОСОВОГО АСИСТЕНТА З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ ШТУЧНОГО ІНТЕЛЕКТУ

2.1 Засоби створення клієнтської частини веб-застосунку

Створення веб-застосунку починається зі створення його зовнішнього вигляду, тобто клієнтської частини. Для цього було використано мови HTML, CSS та JavaScript.

HTML (HyperText Markup Language) - це стандартизована мова розмітки документів для перегляду веб-сторінок у браузері. З її допомогою написаний каркас сторінки, тобто всі елементи, які бачить користувач, написані за допомогою її компонентів. Браузер отримує HTML документ, інтерпретує код і перетворює його на візуальний інтерфейс, який бачить користувач.[8]

Елементи HTML коду, які називаються тегами, дозволяють вставляти різні компоненти на сторінку, наприклад, текст, зображення, відео, аудіо, кнопки, інтерактивні форми та інше. Також він дозволяє використовувати мову CSS, для визначення зовнішнього вигляду та компонування тегів, а також скриптові мови, як JavaScript, для задання поведінки веб-сторінки та зміни її змісту.

CSS (Cascading Style Sheets) - це мова, яка використовується для опису зовнішнього вигляду елементів веб-сторінки, яка написана мовою HTML. Сторінки, написані без використання каскадної таблиці стилів, достатньо прості та однотипні, а CSS дозволяє стилізувати їх. Каскадні таблиці стилів замінили табличний макет веб-сторінок та дозволили відокремити вміст сторінки від його зовнішнього вигляду, що значно полегшує та оптимізує їх створення.

JavaScript - це динамічна об'єкто-орієнтована прототипна мова програмування, яка використовується для створення сценаріїв веб-сторінок[12]. Вона працює на боці користувача та дає йому можливість динамічно знаємодіяти зі сторінкою асинхронно працювати з сервером, змінювати структуру та зовнішній вигляд веб-сторінок.[7]

2.2 Засоби створення серверної частини веб-застосунку

Для створення серверної частини веб-застосунку було обрано мову програмування Python.

Python - це інтерпретована об'єктно-орієнтована мова програмування високого рівня із суворою динамічною типізацією. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Хоча і за означенням Python це інтерпретована мова програмування, але до кінця її такою назвати не можна, адже при першому запуску Python файлу, інтерпретатор створює кеш, який значно пришвидшує подальший запуск даного файлу. Це робить дану мову дуже швидкою серед інших інтерпретованих мов.

Переваги мови Python:

- чистий синтаксис (для виділення блоків використовуються відступи);

- переносність програми;

- стандартний дистрибутив Python має велику кількість стандартних модулів, які являються дуже корисними;

- зручний для розв'язання математичних завдань (без використання додаткових модулів, є можливість працювати з комплексними числами, підносити до степеню, може оперувати з цілими числами будь-якої довжини);

- відкритий код;

- швидкість роботи з даними, що виділяє цю мову серед інших для розробки штучного інтелекту.

Серед недоліків мови Python можна виділити:

- низька швидкодія: через те, що Python це все-таки інтерпретована мова програмування, то вона має нижчу швидкість виконання програм, ніж компільовані мови;

- відсутність статичної типізації: це не можна назвати прямо недоліком, але не всім розробникам подобається працювати з такими мовами, а також це являється ще однією причиною низької швидкодії;

- неможливість модифікувати вбудовані класи: у Python неможливо якось модифікувати класи такі, як int, str, float, list і інші;

- глобальне блокування інтерпретатора (GIL): при своїй роботі інтерпретатор Python постійно використовує велику кількість потоконебезепечних даних, для уникнення їх втрати, потік інтерпретатора захоплює GIL, внаслідок цього, в один момент часу може виконуватись лише одна частина Python коду.

Для написання самої серверної частини використано фреймворк Django. Вона дозволяє створювати багатофункціональні веб-застосунки і значно полегшує їх створення.

2.3 Засоби зберігання даних

Для зберігання даних на сервері було обрано PostgreSQL.

PostgreSQL - це потужна реляційна система управління базами даних з відкритим вихідним кодом. Вона відома своєю надійністю, великим набором функцій і відповідністю стандартам SQL.[16]

Переваги PostgreSQL [17]:

- відкритий вихідний код;
- безкоштовність;
- відповідність стандартам SQL;
- підтримка ACID-транзакцій;
- потужні функції обробки даних;
- можливість до масштабування;
- продуктивність;
- безпека;
- широка підтримка платформ.

Серед недоліків цієї системи управління базами даних можна виділити [17]:

- складність налаштування і адміністрування;
- продуктивність при високому навантаженні на систему;
- обмежена підтримка вертикального масштабування.

PostgreSQL - це потужний інструмент управління базами даних, яка підходить для багатьох задач завдяки своїй надійності і великому набору функцій. Але її використання і вимоги до адміністрування можуть стати перешкодою для деяких користувачів.

2.4 Мовна модель

На даний момент існує велика кількість мовних моделей, серед найпопулярніших можна виділити GPT, Gemini, Copilot, Claude, Cortana. Не кожен з них має API доступ. Також хотілось би, щоб користування створеним голосовим асистентом було безкоштовним. Тому було обрано мовну модель Claude, API якого я умовно безкоштовним (при реєстрації користувачеві дається 5 доларів для користування цією моделлю).

Серед переваг цієї інтелектуальної моделі можна виділити:

- безпеку;
- якість відповіді;
- гнучкість;
- постійне навчання.

Серед недоліків основними є:

- вартість;
- обмеження в знаннях.

2.5 Засоби розпізнавання мови

На Python існує велика кількість бібліотек для розпізнавання мови, наприклад, SpeechRecognition, Google Cloud Speech-to-Text, PocketSphinx, Wit.ai. Кожна з них має свої переваги та недоліки і вибір залежить від області застосування та використання.[9]

Мій вибір впав на бібліотеку SpeechRecognition. Головною перевагою використання даної бібліотеки є її гнучкість та універсальність: вона підтримує як онлайн так і офлайн розпізнавання, а також дозволяє використовувати різні двигуни та API, включаючи Google Web Speech API, Microsoft Bing Voice Recognition, IBM Speech to Text та інші. Серед недоліки можна виділити те, що для

її коректної роботи, потрібно використовувати тільки аудіо файли з розширенням ".wav", що у деяких випадках може бути не зручно, але дану проблему легко вирішити.

Для розпізнавання я вибрав Google API. Переваги використання Google API [1]:

- онлайн розпізнавання на серверах Google, що дозволяє швидко та точно розпізнавати мову, без великого навантаження на систему;
- нейронна мережа нетренована на даних компанії Google, що значно збільшує точність розпізнавання.

Серед недоліків можна виділити лише один: розпізнавання не може відбуватися без підключення до мережі Інтернет, але дана проблема в моєму випадку відразу відпадає, адже сервер завжди повинен бути підключеним до мережі.[1]

2.6 Засоби конвертації аудіо

Так як для розпізнавання мови за допомогою бібліотеки SpeechRecognition, потрібно використовувати файли з розширенням ".wav", аудіо, отримане від користувача, потрібно конвертувати у такий формат. Для конвертації було використано утиліту FFmpeg.

FFmpeg - це потужний набір програм з відкритим кодом для обробки відео та аудіо файлів. FFmpeg включає в себе бібліотеку libavcodec для кодування і декодування, а також багато інструментів командного рядка для роботи з мультимедійними файлами та потоками. [6]

Переваги FFmpeg:

- багатофункціональність: FFmpeg підтримує широкий спектр форматів аудіо та відео, включачи MP4, AVI, MP3, AAC, FLAC і багато інших, а також підтримує конвертацію, запис і відтворення мультимедіа.
- висока продуктивність
- гнучкість

- легкість інтеграції: так як це інструмент командного рядка, то його легко використовувати при розробці будь-яких програмних продуктів.

Недоліки:

- відсутність графічного інтерфейсу
- можливі проблеми з сумісністю
- продуктивність на слабких системах

2.7 Засоби синтезу мови

Для синтезу мови в Python існує багато різних бібліотек, наприклад, pyttsx3, gTTS, Microsoft Azure Text-to-Speech, espeak. Кожна з них має свої особливості і вибір залежить від поставленої задачі та області використання.

Я вибрав бібліотеку gTTS (Google Text-to-Speech), адже вона ідеально підходить в моєму випадку. Серед переваг даної бібліотеки можна виділити [10]:

- онлайн синтез мови на серверах Google, що дозволяє швидко та точно синтезувати мову без великого навантаження на систему;
- нейронна мережа, що синтезує мову, натренована на даних компанії Google, що значно збільшує якість синтезу;
- немає прив’язки до голосу системи: у деяких бібліотек, синтезувати мову можна лише голосом, який встановлений у операційну систему;
- підтримка великої кількості мов для синтезу.

Серед недоліків можна виділити:

- синтез мови не може відбуватися без підключення до мережі Інтернет, але цей недолік нівелюється, адже сервер завжди повинен бути підключений до мережі;
- синтез відбувається роботизованим голосом;
- некоректний синтез мови, при використанні декількох мов в тексті, але це поширена проблема всіх бібліотек.

Через те, що для голосового асистента дуже важливим аспектами є швидкість та точність синтезу мови, було вибрано саме таку бібліотеку.

2.8 Засоби NLP

Існує велика кількість бібліотек для NLP у Python, серед найпопулярніших можна виділити:

- NLTK (Natural Language Toolkit);
- spaCy;
- TextBlob;
- Transformers (від Hugging Face);
- Gensim;
- StanforNLP (Stanza).

У всіх них є свої переваги та недоліки, а вибір конкретної залежить від області використання. Мій вибір впав на бібліотеку spaCy.

spaCy - це одна з найпопулярніших і сучасних бібліотек для NLP в Python. Вона розроблена з акцентом на високу продуктивність і ефективність при обробці тексту, що є дуже важливим аспектом для голосового асистента. [20]

Переваги даної бібліотеки [20]:

- висока продуктивність: spaCy оптимізована для роботи з великими об'ємами даних і надає швидкі алгоритми для токенізації, синтаксичного аналізу і інших задач NLP;
- підтримка сучасних моделей: бібліотека підтримує сучасні статистичні моделі, такі як BERT, GPT та інші;
- комплексний набір інструментів: spaCy надає інструменти для токенізації, лематизації, POS-терінгу, синтаксичного аналізу, Named Entity Recognition, векторизація тексту та інше;
- інтегрованість: бібліотека легко інтегрується з іншими популярними інструментами та бібліотеками NLP, такими як scikit-learn, TensorFlow і PyTorch;
- підтримка великої кількості мов: spaCy підтримує дуже багато мов, включаючи французьку, англійську, українську, німецьку та інші.

Серед недоліків spaCy можна виділити:[20]

- об'єм займаної пам'яті: моделі бібліотеки можуть займати значні об'єми пам'яті, особливо для мов, які мають велику кількість граматики і складними структурами;
- відсутність деяких функцій: не дивлячись на те, що бібліотека підтримує багато функцій NLP, її функціонал менший ніж, наприклад, у NLTK;
- менша гнучність в навчанні моделей: spaCy надає можливість додаткового навчання існуючих моделей, але цей процес менш гнучкий в порівнянні з використанням бібліотек таких як PyTorch або Tensorflow;
- менша гнучкість в налаштуваннях токенізації: вбудований токенізатор достатньо ефективний, але його налаштування можуть бути недостатньо гнучкими для специфічних задач.

Була вибрана саме spaCy тому, що це дуже швидка бібліотека, що є дуже важливим для голосового асистента, а також вона надає весь потрібний функціонал.

2.9 Засоби реалізації логістичної регресії

В Python існує декілька бібліотек, які надають інструменти для реалізації логістичної регресії. Серед найпопулярніших можна виділити:

- scikit-learn;
- Statsmodels;
- TensorFlow;
- PyTorch;
- XGBoost.

Ці бібліотеки надають потужні інструменти для реалізації логістичної регресії, а вибір потрібної бібліотеки залежить від запиту та завдання розробки.

Я вибрав бібліотеку scikit-learn - це одна з найпопулярніших бібліотек машинного навчання на Python. Вона надає простий та ефективний інструментарій для аналізу даних і побудови моделей, підтримує широкий спект алгоритмів машинного навчання. [2]

Основні переваги scikit-learn [19]:

- великій набір алгоритмів: підтримує основні методи класифікації, регресії і кластеризації;
- простота використання: бібліотека має простий та інтуїтивно зрозумілий API, що полегшує його використання;
- хороша інтеграція з іншими бібліотеками: scikit-learn легко інтегрується з numpy, scipy і pandas, що робить її прекрасним інструментом для обробки даних;
- потужні утиліти для моделювання та оцінки: бібліотека підтримує кросс-валідацію, підбір гіперпараметрів, побудову конвеєрів та багато іншого.

Основні недоліки scikit-learn [19]:

- обмежена підтримка методів глибокого навчання;
- обмеження продуктивності;
- обмежена підтримка GPU;
- обмежені можливості візуалізації;
- менша гнучкість при налаштуванні моделей;
- відсутність вбудованої підтримки нейронних мереж.

Scikit-learn - це потужна та універсальна бібліотека для машинного навчання, яка підходить для багатьох задач аналізу та моделювання. Хоча у неї є свої обмеження, але вона прекрасно підходить для реалізації логістичної регресії. [2]

3 РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ ГОЛОСОВОГО АСИСТЕНТА З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 База даних: користувачі та історія

Для створення бази даних у проєкті було використано Django ORM. Модель користувачів був перевизначений під свої потреби, а також було створено модель для зберігання історії спілкування з голосовим асистентом. Код створеної моделі користувачів, а також функції для визначення шляху зберігання фото профілю можна побачити на лістингу 3.1, а код моделі для зберігання історії на лістингу 3.2.

Лістинг 3.1. Код Python для моделі користувачів і функція визначення шляху зберігання фото профілю

```
from django.db import models
from django.contrib.auth.models import AbstractUser
def user_directory_path(instance, filename):
    return f"user_{instance.id}/{filename}"

class User(AbstractUser):
    image = models.ImageField(
        upload_to=user_directory_path, blank=True, null=True, verbose_name="Фото профіля"
    )
    token = models.CharField(max_length=300, verbose_name="Токен")
    class Meta:
        db_table = "users"
        verbose_name = "Користувача"
        verbose_name_plural = "Користувачі"

    def __str__(self) -> str:
            return self.username
```

Лістинг 3.2. Код Python для моделі історії спілкування з асистентом

```
from django.db import models
from users.models import User
class History(models.Model):

    user = models.ForeignKey(to=User, on_delete=models.CASCADE, verbose_name='Користувач')
    text = models.TextField(verbose_name='Текст', blank=True, null=True)
    from_user = models.BooleanField(verbose_name='Від користувача')
    creating_timestamp  = models.DateTimeField(verbose_name='Час створення', auto_now_add=True)

    class Meta:
        db_table = "history"
        verbose_name = "Історія"
        verbose_name_plural = "Історія"
```

UML діаграма моделей проілюстрована на рисунку 3.1.

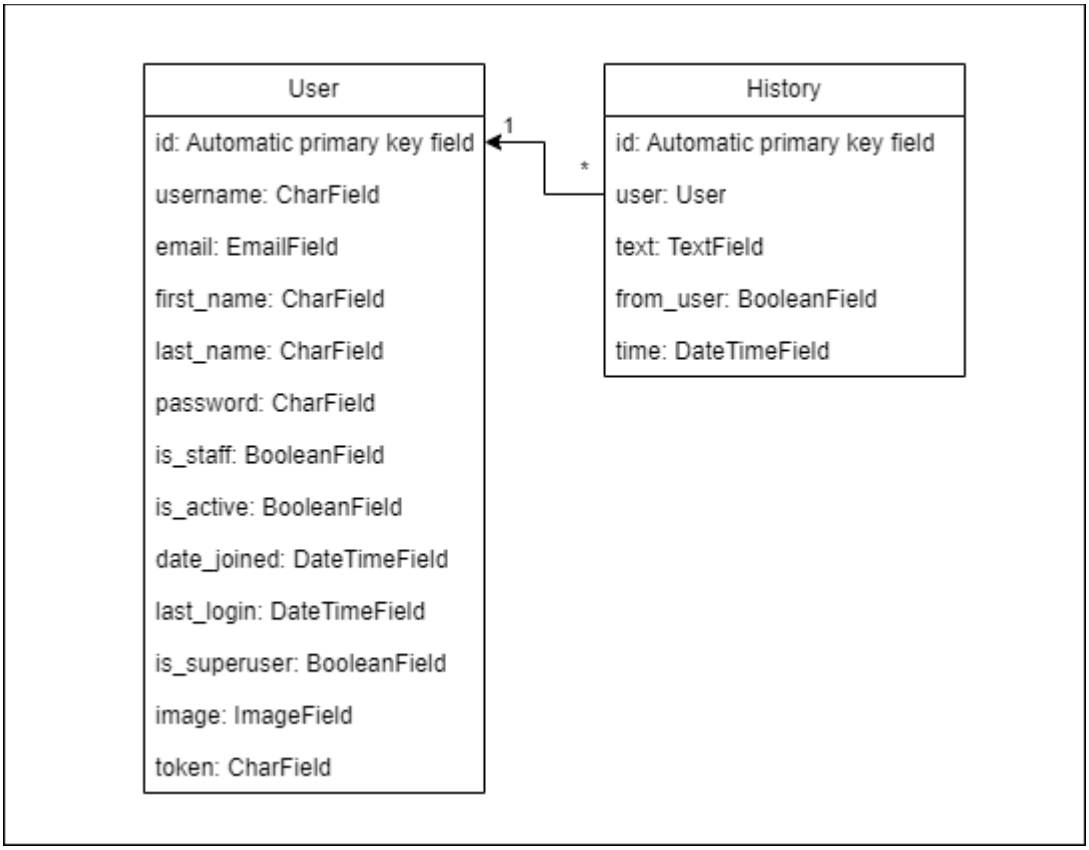

Рис. 3.1 UML діаграма моделей бази даних

3.2 Запис аудіо з мікрофону та відправка на сервер

Запис аудіо з мікрофону у браузері відбувається за допомогою класу MediaRecorder. При завантаженні сторінки створюється новий об'єкт даного класу, далі при натисканні на кнопку запису очищується масив для запису звуку та виконується метод start об'єкту, який при отриманні звуку додає його до масиву, при наступному натисканні даної кнопки, виконується метод stop, після чого дані записується у Blob, який потім відправляється на сервер на сервер з допомогою функції sendAudioToServer (лістинг 3.3). Дана функція створює об'єкт класу FormData, у який записує переданий Blob. Далі за домогою функції fetch це все відправляється на сервер за допомогою POST запиту, додаючи звичайно CSRF токен користувача. Якщо сервер відповів, що все правильно, викликається функція sendTextToServer, про яку пізніше (лістинг 3.4).

Лістинг 3.3. Код JavaScript для запису аудіо з мікрофону у браузері

```
const voice = document.getElementById('voice');
```

```
let mediaRecorder;
let chunks = [];

navigator.mediaDevices.getUserMedia({ audio: true }).then(function(stream) {

    mediaRecorder = new MediaRecorder(stream);
    mediaRecorder.ondataavailable = function(event) {
        chunks.push(event.data);
    };

    mediaRecorder.onstop = function() {
        let audioBlob = new Blob(chunks, {'type' : 'audio/vnd.wave;'});
        sendAudioToServer(audioBlob);
    };
}).catch(function(err) {
    console.error('Error accessing microphone: ', err);
    alert('Помилка доступу до мікрофону');
});

voice.addEventListener('click', function(e) {
    if (text_area.getAttribute('placeholder') == text_placeholder) {
        if (mediaRecorder) {
            text_area.setAttribute('placeholder', 'Слухаю вас');
            text_area.setAttribute('readonly', true);
            text_area.value = '';
            chunks = [];
            mediaRecorder.start();
        } else {
            console.error('MediaRecorder is not initialized');
            alert('Сталася помилка. Спробуйте перезавантажити сторінку');
```

```
        }
    } else {
            text_area.setAttribute('placeholder', text_placeholder);
            text_area.removeAttribute('readonly');
            if (mediaRecorder && mediaRecorder.state !== 'inactive') {
                mediaRecorder.stop();
            } else {
                console.error('MediaRecorder is not initialized or already stopped.');
                alert('Сталася помилка. Спробуйте перезавантажити сторінку');
            }
        }
});
```

Лістинг 3.4. Код JavaScript для відправки аудіо запису на сервер

```
function sendAudioToServer(audioBlob) {

    let formData = new FormData();
    formData.append('audio', audioBlob);

    fetch('/voice-assistant/recognize-audio/', {
        method: 'POST',
        body: formData,
        headers: {
            'X-CSRFToken': getCSRFToken()
        }
    }).then(response => {
        if (!response.ok) {
            throw new Error('Network response was not ok');
        }
        return response.json();
```

```
    }).then(data => {
        sendTextToServer(data["text"]);
    }).catch(error => {
        console.error(error);
        alert("Щось пішло не так");
    })
}
```

3.3 Відправка тексту на сервер

Також спілкуватися з голосовим асистентом можна текстом. Це працює наступним чином:

1. при натисканні кнопки відправити, перевіряється чи не пробує користувач відправити пустий текст, якщо так, то далі нічого не відбувається;
2. далі очищається поле вводу, а також викликається функція sendTextToServer, яка приймає текст, який потрібно опрацювати серверу.

Даний алгоритм можна побачити на лістингу 3.5.

Лістинг 3.5. Код JavaScript для відправки тексту на сервер

```
const text_area = document.getElementById('text_area');

document.getElementById('send').addEventListener('click', function() {
    const text = text_area.value;
    if (!text) {
        alert("Ви відправляєте пустий запит");
        return;
    }
    text_area.value = "";
    sendTextToServer(text);
});
```

Функція sendTextToServer працює по наступному алгоритму:

1. Створюється об'єкт класу FormData, в який додається текст, який потрібно відправити, а також токен користувача для взаємодії з мовною моделлю.
2. За допомогою функції fetch відправляється POST запит з створеним об'єктом і CSRF токеном користувача.
3. Якщо сервер відповідає, що щось не так, то це висвітлюється користувачеві.
4. У іншому випадку відповідь додається на екран, викликається функція scrollBottom, яка перегортає зміст сторінки донизу, а також функція playAudio, яка вмикає аудіо відповідь, яка була отримана від сервера.

Код даного алгоритму, функцій scrollBottom і playAudio, а також обробку натискання на кнопку зупинки програвання аудіо відповіді можна побачити на лістингу 3.6.

Лістинг 3.6. Код JavaScript функції для відправки тексту на сервер, обробки відповіді та відтворення аудіо відповіді

```
const scrollable = document.querySelector('.content');
let audioElement;

function scrollBottom () {
   scrollable.scrollTop = scrollable.scrollHeight;
}

function playAudio(audioUrl)  {
    audioElement  = new Audio(audioUrl);
    audioElement.play();
   document.getElementById("stop-sound").style.cursor = 'pointer';
    audioElement.addEventListener('ended',  function () {
       document.getElementById('stop-sound').style.cursor = 'not-allowed';
    });
}
```

```
function sendTextToServer(text) {
    let formData = new FormData();
    formData.append('text', text);
    formData.append('api_token', document.getElementById("api_token").textContent);

    fetch('/voice-assistant/exec-command/', {
        method: 'POST',
        body: formData,
        headers: {
            'X-CSRFToken': getCSRFToken()
        }
    }).then(respones => {
        if (!respones.ok) {
            throw new Error('Network response was not ok');
        }
        return respones.json();
    }).then(data => {
        document.getElementById("content").insertAdjacentHTML('beforeend',
data["text"]);
        scrollBottom();
        playAudio(data['audio_path']);
    }).catch(error => {
        console.error(error);
        alert("Щось пішло не так");
    });
}
document.getElementById("stop-sound").addEventListener('click', function() {
    if (audioElement) {
        audioElement.pause();
```

```
        audioElement.currentTime  = 0;
        document.getElementById('stop-sound').style.cursor = 'not-allowed';
    }
});
```

3.4 Обробка запитів

Обробка запитів від користувача відбувається за допомогою двох функцій: exec_command (яка оброблює команду від користувача) і recognize_audio (яка розпізнає текст з аудіозапису).

Почнемо з функції recognize_audio. Спочатку вона зберігає отриманий аудіозапис на сервері, потім за допомогою консольної утиліти ffmpeg, конвертує його у формат wave, який потрібен для коректного розпізнавання, після чого, викликає функцію recognize, результат якої відправляється назад користувачеві. Код даної функції можна побачити на лістингу 3.7.

Лістинг 3.7. Код Python для обробки отриманого сервером аудіо запиту

```
import os
from django.http import JsonResponse
from django.conf import settings
import subprocess
from assistant_lib.recognize import recognize
from assistant.models import History


def recognize_audio(request):
    if request.method == 'POST' and request.FILES.get('audio'):
        audio_file  = request.FILES['audio']
        save_path = os.path.join(settings.BASE_DIR, 'media', 'audio',
f'{request.user.username}')
        wav_path = os.path.join(settings.BASE_DIR, 'media', 'audio',
f'{request.user.username}_converted.wav')
        with open(save_path, 'wb') as file:
```

```
            for chunk in audio_file.chunks():
                file.write(chunk)
        subprocess.run(['ffmpeg', '-i', save_path, wav_path, '-y'],
stdout=subprocess.DEVNULL, stderr=subprocess.STDOUT)
        return JsonResponse({'success' : True, 'text' : recognize(wav_path)})
    return JsonResponse({'success' : False, 'error' : 'No audio file received'})
```

Далі функція exec_command. Її алгоритм наступний:

1. отримується текст з запиту;
2. створюється новий запис у базу даних;
3. отримується токен для взаємодії з мовною моделлю;
4. викликається функція execute_command і зберігається результат її роботи;
5. цей результат записується у базу даних;
6. після цього назад користувачеві відправляється відповідь, а також шлях до аудіо файлу з відповіддю.

Код даної функції можна побачити на лістингу 3.8.

Лістинг 3.8. Код Python для обробки отриманого сервером текстового запиту

```
import os
from django.http import JsonResponse
from django.conf import settings
from django.template.loader  import render_to_string

def exec_command(request):
    if request.method == 'POST':
        text = request.POST.get('text')
        History.objects.create(user=request.user, text=text, from_user=True)
        api_token = request.POST.get('api_token')
        audio_path = os.path.join(settings.BASE_DIR, 'media', 'audio',
f'{request.user.username}_answer.wav')
```

```
        answer = execute_command(text, audio_path, api_token)
        History.objects.create(user=request.user, text=answer, from_user=False)
        audio_url = request.build_absolute_uri('/voice-assistant/audio/' +
f'{request.user.username}_answer.wav')
        answer_html = render_to_string('assistant/user_part.html', {'user' : request.user,
'text' : text, 'answer' : answer})
        return JsonResponse({'success' : True, 'text' : answer_html, 'audio_path' :
audio_url})
    return JsonResponse({'success' : False})
```

3.5 Розпізнавання мови

Розпізнавання мови відбувається за допомогою бібліотеки SpeechRecognition. Була створена функція recognize, яка приймає шлях до аудіо файлу і працює наступним чином:

1. Створюється об'єкт класу speech_recognition WavFile, в який передається шлях до аудіо файлу.
2. Створюється об'єкт класу speech_recognition Recognizer.
3. Відкривається аудіо файл, за допомогою об'єкта класу WavFile.
4. Recognizer слухає файл декілька секунд, щоб зрозуміти, що являється шумом, а що голосом.
5. Recognizer розпізнає мову та повертає результат.

Код функції recognize можна побачити на лістингу 3.9.

Лістинг 3.9. Код Python функції розпізнавання мови

```
import speech_recognition

def recognize(sample_path: str):
    sample = speech_recognition.WavFile(sample_path)
    recognizer = speech_recognition.Recognizer()
```

```
    with sample as audio:
        recognizer.adjust_for_ambient_noise(audio)
        content = recognizer.record(audio)
        return recognizer.recognize_google(content, language='uk-UK')
```

3.6 Синтез мови

Синтез мови відбувається за допомогою бібліотеки gTTS. Створено функцію say, яка приймає текст, який потрібно сказати, а також шлях, куди потрібно зберегти аудіо файл. Алгоритм функції наступний:

1. створюється об'єкт класу gtts gTTS, в який передається текст і мова, якою потрібно синтезувати;
2. зберігається синтезоване мовлення.

Код функції say можна побачити на лістингу 3.10.

Лістинг 3.10. Код Python функції для синтезу мови

```
from gtts import gTTS
def say(text: str, save_path: str):
    tts = gTTS(text, lang='uk', slow=False)
    tts.save(save_path)
```

3.7 Взаємодія з мовною моделлю

Для взаємодії з мовною моделлю була використана бібліотека anthropic і мовною моделлю claude. Розроблено функцію lang_model_answer, яка приймає команду, яку потрібно відправити моделі, шлях, куди потрібно зберегти аудіо відповідь моделі, а також API ключ, за допомогою якого відбувається звернення до моделі. Код функції lang_model_answer можна побачити на лістингу 3.11.

Лістинг 3.11. Код Python функції для взаємодії з мовною моделлю Claude

```
import anthropic
from .speak import say
```

```
def lang_model_answer(command, save_audio_path, api_key):
    try:
        client  = anthropic.Anthropic(api_key=api_key)
        message = client.messages.create(
            model="claude-3-opus-20240229",
            max_tokens=1000,
            temperature=0,
            system="Ти помічник для користувача",
            messages=[
                {
                    "role": "user",
                    "content": [
                        {
                            "type": "text",
                            "text": command,
                        }
                    ],
                }
            ],
        )
    except anthropic.AuthenticationError:
        say(CONFIG["lang_model_answer"]["error_auth_message"], save_audio_path)
        return CONFIG["lang_model_answer"]["error_auth_message"]
    answer = message.content[0].text
    answer_to_say = answer.replace("\n", " ")
    answer_to_print = answer.replace("\n", "<br>")
    say(answer_to_say, save_audio_path)
    return answer_to_print
```

3.8 Функції відповіді голосового асистента

Інші функції, які відповідають за відповідь голосового асистента користувачу, працюють всі схожим чином. Всі вони приймають текст команди і шлях, куди потрібно зберегти аудіо відповідь. Код таких функцій можна побачити на лістингу 3.12.

Лістинг 3.12. Код Python функцій для відповіді користувачу

```
from random import choice
import requests
from .params_parser import extract_params, normilaze_city
from .speak import say

def no_results_handle(save_audio_path):
    say('Я не можу вас зрозуміти', save_audio_path)
    return 'Я не можу вас зрозуміти'

def find_in_google(command, save_audio_path):
    keywords = CONFIG["find_in_google"]["keywords"]
    ban_words = CONFIG["find_in_google"]["ban_words"]
    params = extract_params(command, keywords, ban_words)
    if params is None:
        return no_results_handle(save_audio_path)

    pattern_phrases = CONFIG["find_in_google"]["pattern_phrases"]
    answer_to_say = choice(pattern_phrases)
    link = "https://www.google.com/search?q=" + "+".join(params.split())
    say(answer_to_say, save_audio_path)
    return answer_to_say + f'<br><a href="{link}">{params}</a>'

def find_video(command, save_audio_path):
```

```
    keywords = CONFIG["find_video"]["keywords"]
    ban_words = CONFIG["find_video"]["ban_words"]
    params = extract_params(command, keywords, ban_words)
    if params is None:
        return no_results_handle(save_audio_path)

    pattern_phrases = CONFIG["find_video"]["pattern_phrases"]
    answer_to_say = choice(pattern_phrases)
    link = "https://www.youtube.com/results?search_query=" + "+".join(params.split())
    say(answer_to_say, save_audio_path)
    return answer_to_say + f'<br><a href="{link}">Відео</a>'

def weather(command, save_audio_path):
    try:
        keywords = CONFIG["weather"]["keywords"]
        ban_words = CONFIG["weather"]["ban_words"]
        params = extract_params(command, keywords, ban_words)
        if params is None:
            return no_results_handle(save_audio_path)

        city = normilaze_city(params)
        base_url = CONFIG["weather"]["base_url"]
        api_key = CONFIG["weather"]["api_key"]
        params = {"q": city, "appid": api_key, "units": "metric", "lang": "uk"}
        response = requests.get(base_url, params=params)

        if response.status_code == 200:
            data = response.json()
            weather_description = data["weather"][0]["description"]
            temperature = data["main"]["temp"]
```

```
            feels_like = data["main"]["feels_like"]
            humidity = data["main"]["humidity"]

            answer_to_say = f"Погода в {city}. Опис: {weather_description}.
Температура: {temperature} градусів Цельсія. Відчувається як: {feels_like}
градусів Цельсія. Вологість: {humidity} відсотків."

            answer_to_print = f"Погода в {city}.<br> Опис: {weather_description}.<br>
Температура: {temperature} °C.<br> Відчувається як: {feels_like} °C.<br>
Вологість: {humidity}%."

        else:
            answer_to_say = answer_to_print = (
                "Нажаль, я зараз не можу відповісти на цей запит"
            )
        say(answer_to_say, save_audio_path)
        return answer_to_print
    except Exception:
        say(CONFIG["weather"]["error_message"], save_audio_path)
        return CONFIG["weather"]["error_message"]
```

3.9 Виділення параметрів команди

При розробці голосового асистента я зіткнувся з наступною проблемою: коли користувач відправляє команду, наприклад, "знайди в інтернеті котів", не зрозуміло як дізнатися, що саме потрібно шукати, адже сама команда може бути по різному сказана. Для людини відразу зрозуміло, що потрібно шукати в інтернеті, адже ми бачимо цей зв'язок між словами "знайди" і "котів", а комп'ютер не може. Але потім я звернувся до NLP і зрозумів, що насправді може. Була розроблена функція extract_params, яка приймає текст команди, список слів, відносно яких потрібно

шукати зв'язок (ключові слова), а також список слів, які відносяться до команди, а не до її параметрів (заборонені слова). Працює ця функція наступним чином:

1. Виділяються леми для ключових та заборонених слів.
2. У тексті команди шукається ключове слово.
3. Відносно нього шукаються залежні від нього слова, яких немає у списку заборонених слів (для команди "знайди в інтернеті штучний інтелект" це буде слово "інтелект").
4. Викликається функція find_phrase, яка приймає слова, відносно яких потрібно шукати фразу, тобто залежні від них слова (для команди "знайди в інтернеті штучний інтелект", у цю функцію буде передано слово "інтелект", а поверне вона "штучний інтелект"), результат якої повертає функція extract_params.

Код функцій extract_params і find_phrase можна побачити на лістингу 3.13.

Лістинг 3.13. Код Python функцій для пошуку параметрів у запиті і пошуку фрази до якої відноситься слово

```
import spacy

nlp = spacy.load("uk_core_news_sm")

def find_phrase(text_doc, start_words):
    phrase = []
    for token in text_doc:
        if token.head in start_words or token in start_words:
            phrase.append(token)

    if sorted(phrase) == sorted(start_words):
        return " ".join([token.text for token in text_doc if token in start_words])
    else:
        return find_phrase(text_doc, phrase)

def extract_params(text: str, keywords: list[str], ban_words: list[str]):
```

```
delete_words = []
text = " ".join(word for word in text.split() if word.lower() not in delete_words)
keywords_doc_lemmas = [token.lemma_ for token in nlp(" ".join(keywords))]
ban_words_doc_lemmas = [token.lemma_ for token in nlp(" ".join(ban_words))]
doc = nlp(text)

# пошук ключового слова
index = -1
for i, token in enumerate(doc):
    if token.lemma_ in keywords_doc_lemmas:
        index = i
        break

# пошук залежних слів
start_words = []
for token in doc:
    if (
        token.lemma_ not in ban_words_doc_lemmas
        and token.head == doc[index]
        and token != doc[index]
    ):
        start_words.append(token)
if len(start_words) < 1:
    return None

return find_phrase(doc, start_words)
```

3.10 Класифікація команд

Для того, щоб зрозуміти, яка команда була сказана голосовому асистенту було використано бібліотеку scikit-learn, метод логістичної регресії. Штучний інтелект було натреновано на дата сеті, який можна побачити на лістингу 3.14.

Лістинг 3.14. Код Python дата сету для тренування логістичної регресії

```
DATA_SET = {
    'знайди в інтернеті' : 'find_in_google',
    'пошукай в інтернеті' : 'find_in_google',
    'можеш знайти в інтернеті' : 'find_in_google',
    'можеш пошукати в інтернеті' : 'find_in_google',
    'чи можеш знайти в інтернеті' : 'find_in_google',
    'знайди будь ласка в інтернеті' : 'find_in_google',
    'пошукай будь ласка в інтернеті' : 'find_in_google',
    'можеш будь ласка знайти в інтернеті' : 'find_in_google',
    'можеш будь ласка пошукати в інтернеті' : 'find_in_google',
    'знайди відео' : 'find_video',
    'пошукай відео' : 'find_video',
    'можеш знайти відео' : 'find_video',
    'можеш пошукати відео' : 'find_video',
    'чи можеш знайти відео' : 'find_video',
    'знайди будь ласка відео' : 'find_video',
    'пошукай будь ласка відео' : 'find_video',
    'можеш будь ласка знайти відео' : 'find_video',
    'можеш пошукати будь ласка відео' : 'find_video',
    'знайди на ютубі' : 'find_video',
    'пошукай на ютубі' : 'find_video',
    'можеш знайти на ютубі' : 'find_video',
    'можеш пошукати на ютубі' : 'find_video',
    'чи можеш знайти на ютубі' : 'find_video',
```

```
    'знайди будь ласка на ютубі' : 'find_video',
    'пошукай будь ласка на ютубі' : 'find_video',
    'можеш будь ласка знайти на ютубі' : 'find_video',
    'можеш пошукати будь ласка на ютубі' : 'find_video',
    'знайди в ютубі' : 'find_video',
    'пошукай в ютубі' : 'find_video',
    'можеш знайти в ютубі' : 'find_video',
    'можеш пошукати в ютубі' : 'find_video',
    'чи можеш знайти в ютубі' : 'find_video',
    'знайди будь ласка в ютубі' : 'find_video',
    'пошукай будь ласка в ютубі' : 'find_video',
    'можеш будь ласка знайти в ютубі' : 'find_video',
    'можеш пошукати будь ласка в ютубі' : 'find_video',
    'яка погода зараз в' : 'weather',
    'яка погода в' : 'weather',
    'яка температура зараз в' : 'weather',
    'що по погоді в' : 'weather',
    'погода' : 'weather',
}
```

Сама функція по визначенню потрібної функції називається execute_command, примає вона команду, передану від користувача, шлях для зберігання аудіо відповіді, а також API ключ, для взаємодії з мовною моделлю Код функції можна побачити на лістингу 3.15.

Лістинг 3.15. Код Python для класифікації запиту користувача і визначення потрібної функції

```
from sklearn.linear_model  import LogisticRegression
from sklearn.feature_extraction.text  import CountVectorizer

from .data_set import DATA_SET, TO_FUNC
```

```
from .assistant import lang_model_answer

vectorize = CountVectorizer()
vectors = vectorize.fit_transform(list(DATA_SET.keys()))

clf = LogisticRegression()
clf.fit(vectors, list(DATA_SET.values()))

def execute_command(command, save_audio_path, api_key):
    user_command_vector = vectorize.transform([command])
    predicted = clf.predict_proba(user_command_vector)
    max_probability = max(predicted[0])

    if max_probability > 0.5:
        func = TO_FUNC[clf.classes_[predicted[0].argmax()]]
        return func(command, save_audio_path)
    else:
        return lang_model_answer(command, save_audio_path, api_key)
```

3.11 Тестування веб-застосунку

Після того, як користувач вперше заходить на сайт, від повинен зареєструватися або авторизуватися. Я, як новий користувач веб-застосунку, почну з реєстрації (рисунок 3.2).

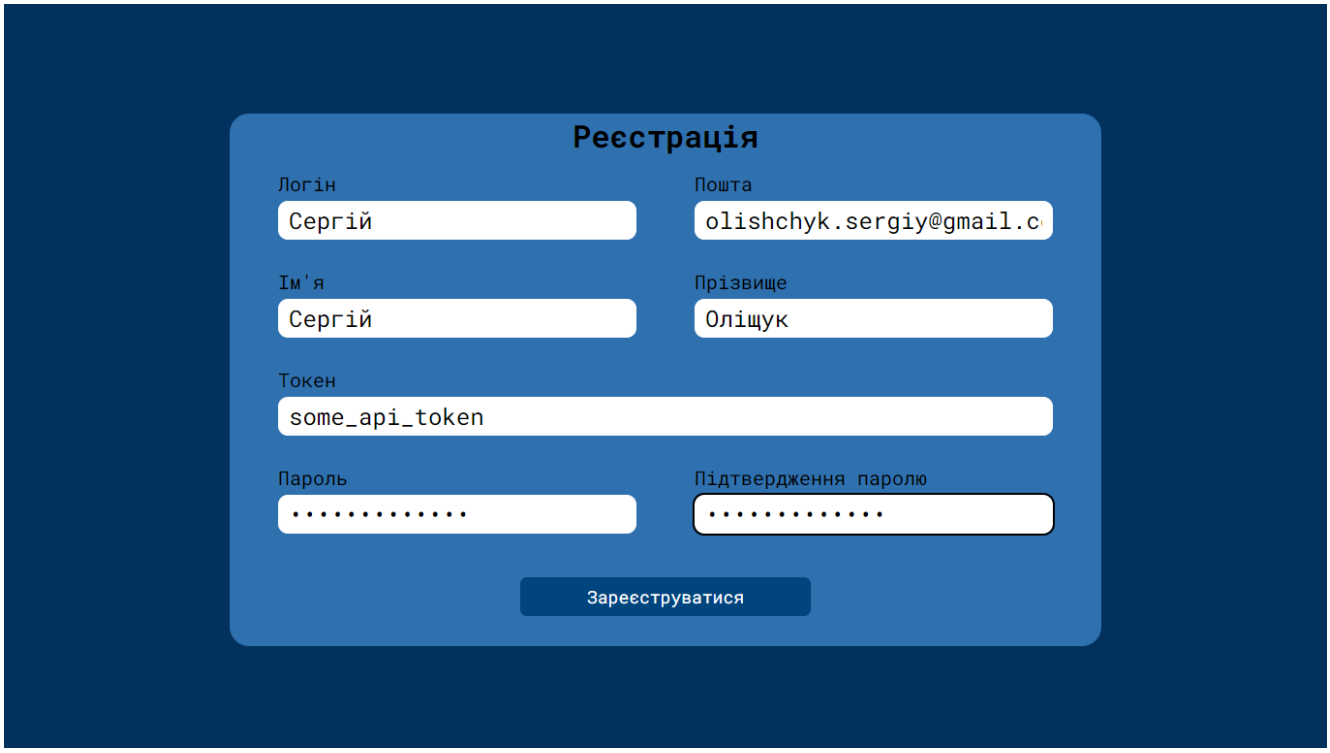

Рис. 3.2 Сторінка реєстрації

Після реєстрації автоматично відбувається авторизація в профіль, але спробуємо вийти та спробувати авторизуватися заново (рисунок 3.3).

Рис. 3.3 Сторінка авторизації.

Спробуємо змінити свій профіль та завантажити фотографію профілю (рисунок 3.4-3.5).

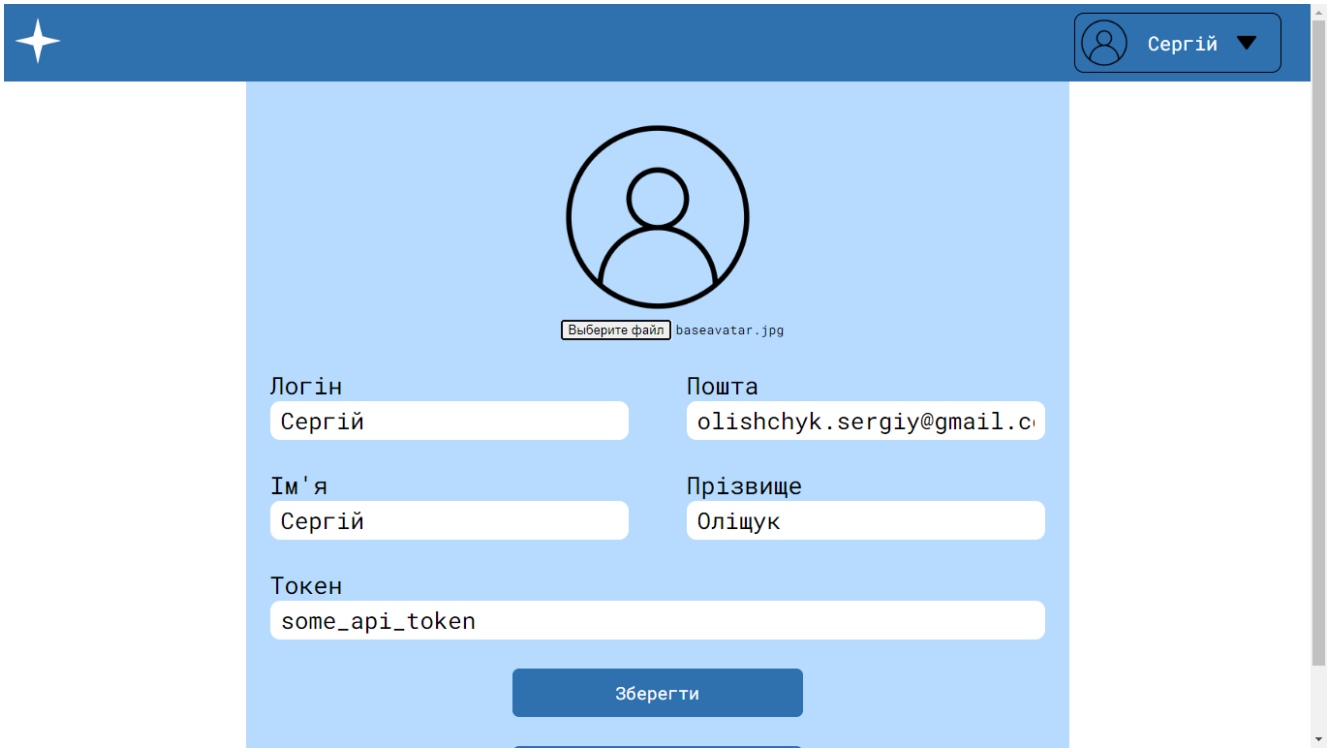

Рис. 3.4 Сторінка зміни профілю

Рис. 3.5 Сторінка зміни профілю після завантаження нової фотографії профілю.

Перейдемо до сторінки з функціоналом голосового асистента (рисунок 3.6).

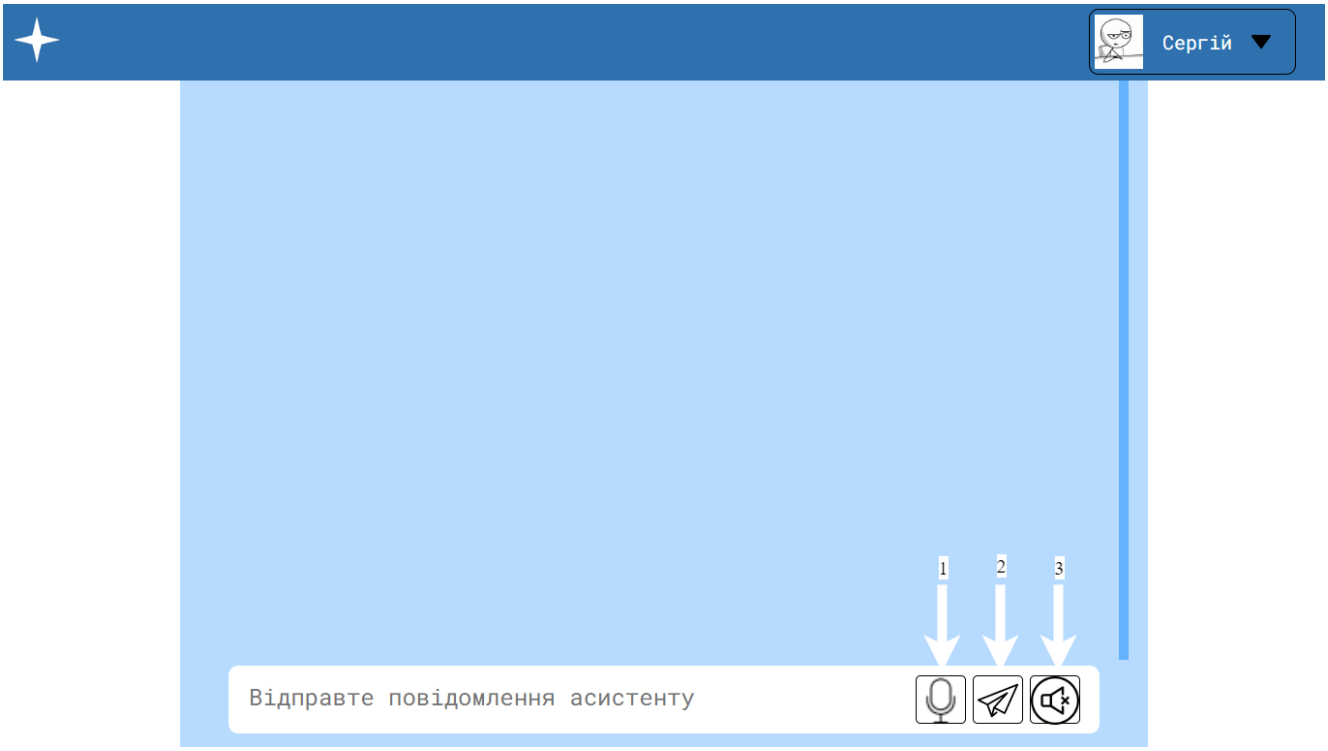

Рис. 3.6 Сторінка з функціоналом голосового асистента

Знизу сторінки знаходиться поле, за допомогою якого з асистентом можна спілкуватися текстом, щоб відправити текст потрібно натиснути на кнопку, позначену стрілкою 1. Для того, щоб відправити запит голосом, потрібно натиснути кнопку позначену стрілкою 2. Для того, щоб зупинити відтворення аудіо відповіді асистента, потрібно натиснути кнопку, позначену стрілкою 3.

Спробуємо відправити асистенту запит "знайди в інтернеті інформацію про штучний інтелект" текстом і подивимося на відповідь (рисунок 3.7).

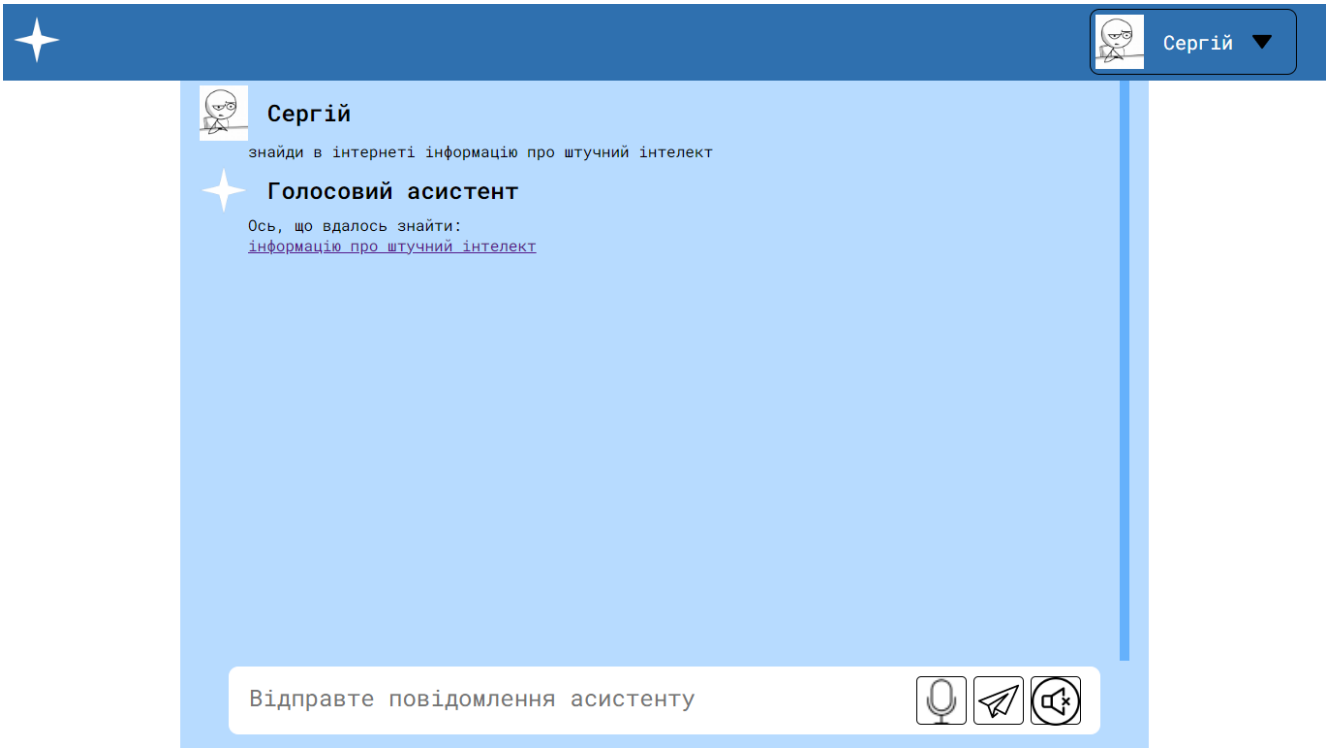

Рис. 3.7 Відповідь голосового асистента на текстовий запит.

Спробуємо тепер попросити знайти інформацію про нейронні мережі, але уже голосом (рисунок 3.8).

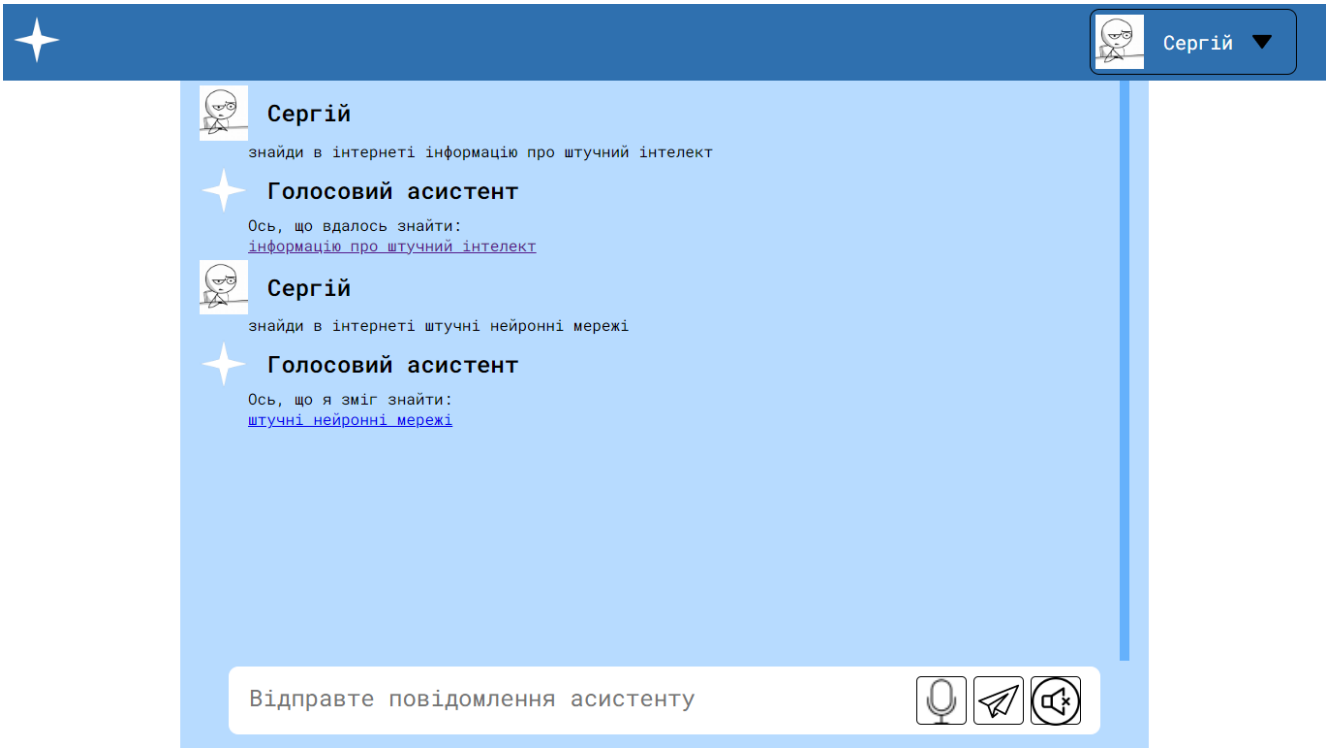

Рис. 3.8 Відповідь голосового асистента на голосовий запит

Після проведення 20 текстів, було виявлено, що при відсутності шуму і якщо команда була сказана точно, то розпізнавання тексту відбувається правильно у 95% випадків. Розпізнавання самої команди відбувається з точністю приблизно 95%, а розпізнавання параметрів команди відбувається коректно у 85% випадків.

ВИСНОВКИ

Розробка веб-застосунку голосового асистента з використанням технологій штучного інтелекту є актуальним та перспективним напрямком, який сприяє підвищенню ефективності взаємодії користувачів з інформаційними системами та інтелектуальною моделлю. У ході виконання дослідження була досягнута основна мета – підвищення ефективності використання технологій штучного інтелекту, таких як обробка природної мови, логістична регресія, синтез та розпізнавання мови, а також було розроблено веб-застосунок голосового асистента.

Було вивчено та проаналізовано історія появи та розвитку розпізнавання та синтезу мови, NLP, логістичної регресії. Було проведено дослідження еволюції цих технологій, що дозволило зрозуміти їхню значущість та внесок у сучасні рішення. Це дослідження створило міцну основу для подальшого використання цих технологій у практичній реалізації проекту.

Було детально проаналізовано принципи функціонування технологій розпізнавання та синтезу мови, NLP та логістичної регресії, що дозволило ефективно використовувати їх під час розробки голосового асистента. Це забезпечило точне розуміння механізмів їх роботи та можливостей інтеграції.

Було досліджено можливості та переваги використання Django для розробки веб-застосунків. Вивчення цього фреймворку дало змогу вибрати оптимальні методи і засоби створення стабільного та масштабованого веб-застосунку.

Успішно розроблено всі компоненти системи голосового асистента, зокрема, модулі розпізнавання мови, синтезу мови та обробки природної мови. Це дозволило створити функціональний прототип, здатний виконувати команди користувачів і надавати відповідну інформацію.

Було завершено розробку веб-застосунку, який включає інтерактивний інтерфейс та інтеграцію з системою голосового асистента. Проведено тестування веб-застосунку, що підтвердило його працездатність та зручність використання.

Розроблений веб-застосунок та система голосового асистента демонструє значний потенціал для подальшого розвитку і впровадження у різних сферах. Він може бути використаний як основа для створення спеціалізованих рішень у таких

галузях, як електронна комерція, охорона здоров’я, освіта, обслуговування клієнтів тощо. Веб-застосунок може бути адаптований для різних потреб, забезпечуючи інтерактивний і зручний інтерфейс для користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Оліщук С.О., Фесенко М.А., Використання бібліотеки speech recognition для розпізнавання природньої мови сторінки / Науково-практична конференція «Проблеми комп'ютерної інженерії». К.: ДУІКТ, 2023 р. с. 31-32

2. Фесенко М.А., Оліщук С.О. Використання бібліотеки scikit-learn для логічної регресії/ Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. – К.: ДУІКТ, 2024р.,

3. A short history of text to speech | Speechify. *Speechify*. URL: https://speechify.com/blog/history-of-text-to-speech/ (дата звернення: 01.03.2024).

4. Black A. W., Tokuda K. The blizzard challenge - 2005: evaluating corpus-based speech synthesis on common datasets. *Interspeech 2005*. ISCA, 2005.

5. Django. *Django Project*. URL: https://www.djangoproject.com/ (дата звернення: 18.04.2024).

6. FFmpeg. *FFmpeg*. URL: https://ffmpeg.org/ (дата звернення: 03.04.2024).

7. Flanagan D. JavaScript: the definitive Guide. 7-ме вид. O'Reilly Media, 2020. 706 с.

8. Freeman E., Robson E. Head First HTML and CSS. O'Reilly Media, Incorporated, 2012.

9. GitHub : Speech recognition module for Python. *GitHub*. URL: https://github.com/Uberi/speech_recognition (дата звернення: 01.04.2024).

10. GTTS. *gTTS – gTTS documentation*. URL: https://gtts.readthedocs.io/ (дата звернення: 02.04.2024).

11. H M. J., Jurafsky D. Speech and language processing: pearson new international edition. Pearson Education, Limited, 2013. 944 с.

12. Haverbeke M. Eloquent JavaScript: a modern introduction to programming. 3-тє вид. San Francisco, California, United States of America : No Starch Press, 2018. 472 с.

13. Hirschberg J., Manning C. D. Advances in natural language processing. *Science*. 2015. Т. 349, № 6245. С. 261–266.

14. Juang B. H., Rabiner L. R. Speech recognition, automatic: history. *Encyclopedia of language & linguistics*. 2006. С. 806–819.

15. Kleinbaum D. G. Logistic regression: a self-learning text. New York, NY : Springer Science+Business Media, LLC, 2010.

16. Postgresql server programming - second edition. Packt Publishing, 2015. 312 с.

17. PostgreSQL. *PostgreSQL*. URL: https://www.postgresql.org/ (дата звернення: 21.04.2024).

18. Rabiner L. R., Schafer R. W. Introduction to digital speech processing. *Foundations and trends in signal processing*. 2007. Т. 1, № 1–2. С. 1–194.

19. Scikit-learn: machine learning in Python – scikit-learn 0.16.1 documentation. *scikit-learn*. URL: https://scikit-learn.org/ (дата звернення: 11.04.2024).

20. SpaCy · industrial-strength natural language processing in python. *spaCy*. URL: https://spacy.io/ (дата звернення: 13.04.2024).

21. Zen H., Tokuda K., Black A. W. Statistical parametric speech synthesis. *Speech communication*. 2009. Т. 51, № 11. С. 1039–1064.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

Кваліфікаційна робота на тему : «Розроблення веб-застосунку голосового асистента з використанням технології штучного інтелекту»

Виконав: здобувач вищої освіти гр. ШІД-41
Сергій ОЛІЩУК
Керівник: кандидат технічних наук, доцент
Максим ФЕСЕНКО

2

Мета роботи: підвищення ефективності використання технологій штучного інтелекту, таких як NLP, логістична регресія, синтез та розпізнавання мови.

Об'єкт дослідження: процес розробки веб-додатку голосового асистента.

Предмет дослідження: система голосового асистента.

Основні завдання кваліфікаційної роботи:

1. Вивчення історії появи технологій розпізнавання та синтезу мови, NLP, логістичної регресії.
2. Вивчення принципу роботи технологій розпізнавання та синтезу мови, NLP, логістичної регресії для правильного їх використання при розробці голосового асистента.
3. Аналіз роботи фреймворку Django для Python для розробки веб-застосунку.
4. Реалізація системи голосового асистента.
5. Створення веб-застосунку голосового асистента.

3

Використані технології

- Розпізнавання людської мови.
- Синтез мови.
- Логістична регресія.
- NLP.
- Django Python.

4

Демонстрація проекту

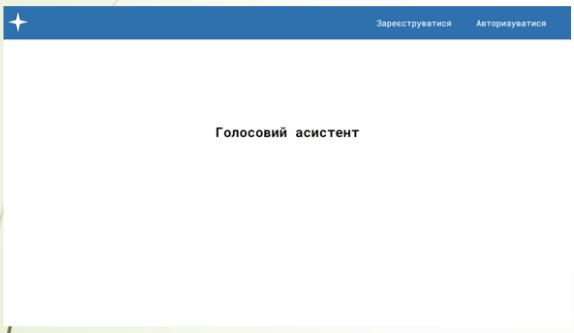

Рис. 1. Головна сторінка не авторизованого користувача

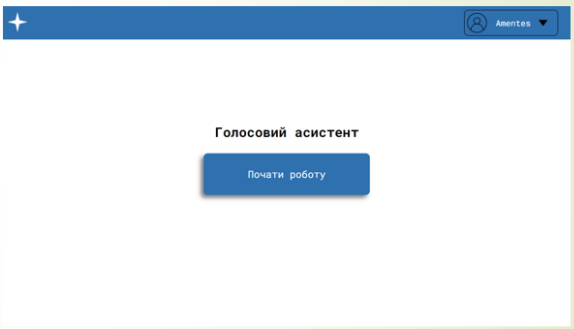

Рис. 2. Головна сторінка авторизованого користувача

5

Демонстрація роботи

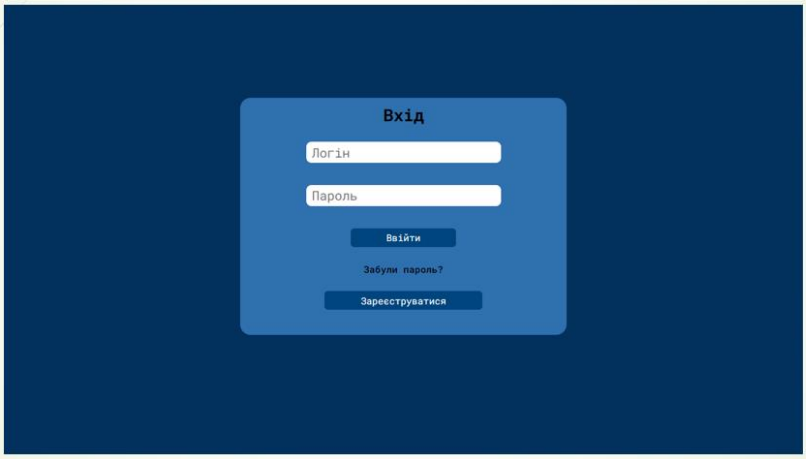

Рис. 3. Сторінка авторизації

6

Демонстрація роботи

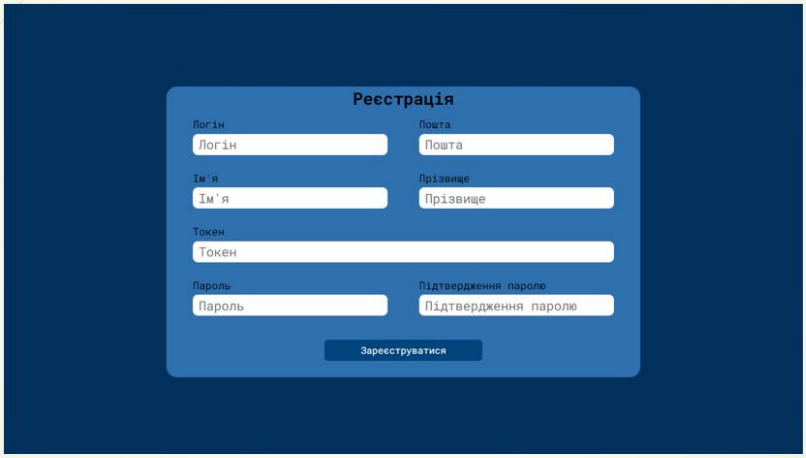

Рис. 4. Сторінка реєстрації

7

Демонстрація роботи

Рис. 5. Сторінка профілю користувача

8

Демонстрація роботи

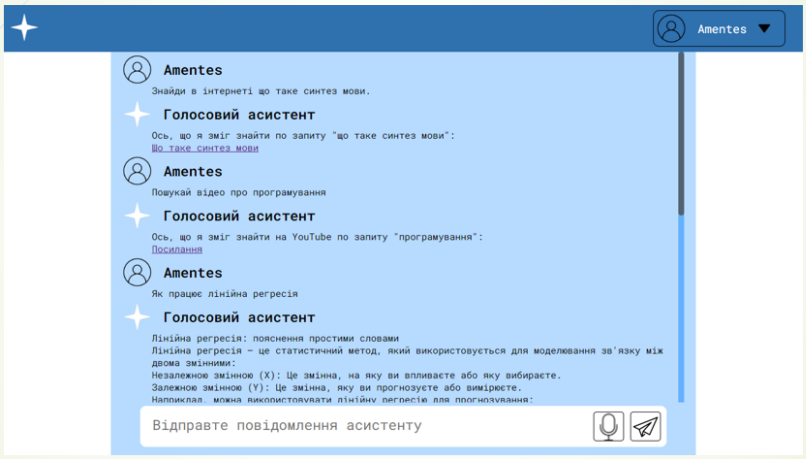

Рис. 6. Робота голосового асистента

Висновки

- У кваліфікаційній роботі проведено аналіз історії розвитку наступних технологій: розпізнавання та синтез мови, NLP.
- Проведено аналіз технологій розпізнавання та синтезу мови, NLP, логістичної регресії, Django Python.
- Розроблено систему голосового асистента з використанням технології штучного інтелекту.
- Створено веб-застосунок голосового асистента на основі розробленої системи.