БАКАЛАВРСЬКА РОБОТА

БР. ІП - 20.00.00.000 ПЗ

Група ІП-21-2

Бойчук Юрій

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Бойчук Юрій Олексійович

(прізвище, ім'я, по батькові)

УДК 004

(індекс)

БАКАЛАВРСЬКА РОБОТА

Інтеграція голосових команд в процес Андроїд Java-розробки

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

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

Здобувач освітнього рівня Бойчук Ю.О.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Мельник Віталій Дмитрович, к.т.н., доцент

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Бойчуку Юрію Олексійовичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) “ Інтеграція голосових команд в процес Андроїд Java-розробки”

керівник проекту (роботи) Мельник В.Д., доцент

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

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

1. Аналіз імплементації голосових команд і розпізнавання мови в процес андроїд java-розробки
2. Проблематика застосування систем розпізнавання мовлення для генерації програмного коду
3. Алгоритми та моделі розпізнавання мовлення для Android розробки
4. Проектування діаграми класів
5. Програмна реалізація системи інтеграція голосових команд в процес Андроїд java-розробки

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Архітектура системи обробки аудіофайлів з використанням Google Cloud Platform (рис. 1.1)
2. Архітектура Android (рис. 2.1)
3. Блок-схема системи розпізнавання мовлення (рис. 2.2)
4. Use Case діаграма (рис. 2.3)
5. Діаграма послідовності (рис. 2.4)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **28 квітня 2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз імплементації голосових команд і розпізнавання мови в процес андроїд java-розробки	06.05.2025	виконано
2	Проблематика застосування систем розпізнавання мовлення для генерації програмного коду	16.05.2025	виконано
3	Алгоритми та моделі розпізнавання мовлення для Android розробки	27.05.2025	виконано
4	Проектування діаграми класів	01.06.2025	виконано
5	Програмна реалізація системи інтеграція голосових команд в процес Андроїд java-розробки	05.06.2025	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	10.06.2025	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 77 сторінок, 18 рисунків, список використаних джерел із 34 найменуваннями.

Метою даної роботи є розробка прототипу системи, яка дозволяє інтегрувати голосові команди в процес Android Java-розробки з використанням сучасних систем розпізнавання мовлення.

Об'єкт дослідження - процес розробки мобільних додатків на платформі Android із використанням мови Java.

Предмет дослідження - методи та інструменти реалізації голосових команд і систем розпізнавання мовлення у процесі Android Java-програмування.

В першому розділі розглянуто основні аспекти та проблеми впровадження голосового керування в Android-додатки, а також обґрунтовано необхідність розробки нової архітектури інтеграції мовлення

В другому розділі проведено аналіз алгоритмів, бібліотек та моделей розпізнавання мовлення, спроєктовано архітектуру системи з використанням UML-діаграм.

В третьому розділі реалізовано прототип системи з підтримкою голосових команд, проведено тестування та оцінку працездатності, а також розроблено користувацький інтерфейс.

Висновок: запропоновано архітектуру системи, що дозволяє обробляти голосові команди розробника та трансформувати їх у програмні інструкції в межах середовища Android. Розроблено діаграми проектування, реалізовано основні компоненти програмного забезпечення та здійснено його тестування.

КЛЮЧОВІ СЛОВА: РОЗПІЗНАВАННЯ МОВЛЕННЯ, ANDROID, JAVA, ГОЛОСОВЕ УПРАВЛІННЯ, МОБІЛЬНА РОЗРОБКА, GOOGLE VOICE RECOGNITION, ІНТЕРФЕЙС КОРИСТУВАЧА, ІНТЕГРАЦІЯ API

ANNOTATION

The bachelor's thesis contains 77 pages, 18 figures, a list of used sources with 34 names.

The purpose of this work is to develop a prototype of a system that allows integrating voice commands into the Android Java development process using modern speech recognition systems.

The object of the study is the process of developing mobile applications on the Android platform using the Java language.

The subject of the study is methods and tools for implementing voice commands and speech recognition systems in the Android Java programming process.

The first section considers the main aspects and problems of implementing voice control in Android applications, and also justifies the need to develop a new speech integration architecture.

The second section analyzes speech recognition algorithms, libraries and models, designs the system architecture using UML diagrams.

In the third section, a prototype of a system with voice command support is implemented, testing and performance evaluation are carried out, and a user interface is developed.

Conclusion: a system architecture has been proposed that allows the developer's voice commands to be processed and transformed into program instructions within the Android environment. Design diagrams have been developed, the main software components have been implemented, and its testing has been carried out.

KEYWORDS: SPEECH RECOGNITION, ANDROID, JAVA, VOICE CONTROL, MOBILE DEVELOPMENT, GOOGLE VOICE RECOGNITION, USER INTERFACE, API INTEGRATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ASR – Automatic Speech Recognition – автоматичне розпізнавання мовлення

GVR – Google Voice Recognition – система розпізнавання мовлення Google

SDK – Software Development Kit – набір інструментів для розробника

TTS – Text-To-Speech – перетворення тексту в мовлення

DFD – Data Flow Diagram – діаграма потоку даних

USE CASE – Use Case Diagram – діаграма варіантів використання

CLD – Class Diagram – діаграма класів

AOSP – Android Open Source Project – проєкт відкритого коду Android

NLP – Natural Language Processing – обробка природної мови

ВСТУП

Актуальність роботи

У сучасному світі мобільні технології стрімко розвиваються, і важливою тенденцією стає природна взаємодія людини з пристроями, зокрема через голосове управління. Інтеграція систем розпізнавання мовлення у мобільні додатки дозволяє значно підвищити їхню доступність, зручність використання та інклюзивність. Особливої актуальності набуває використання голосових команд у середовищі Android Java, оскільки ця платформа є однією з найпоширеніших у світі мобільного програмування. Удосконалення механізмів розпізнавання мовлення, а також подолання обмежень стандартних рішень сприяє ефективнішій реалізації голосового введення та відкриває нові можливості для автоматизації програмної розробки.

У сучасну епоху стрімкого розвитку мобільних технологій та зростаючої популярності голосових інтерфейсів, інтеграція систем розпізнавання мовлення у процес Android Java-розробки набуває особливої важливості. Застосування голосових команд у мобільних додатках не лише підвищує рівень зручності користувача, а й відкриває нові можливості для розробки програмного забезпечення в умовах інклюзивності, автоматизації та оптимізації взаємодії з пристроєм. Однак впровадження таких систем супроводжується низкою технічних викликів, пов’язаних із точністю розпізнавання, взаємодією з апаратним забезпеченням та обмеженнями мобільних платформ. Тому дослідження алгоритмів, архітектурних рішень і практичних аспектів інтеграції мовленнєвих інтерфейсів в Android-додатки є актуальним та спрямованим на подолання цих викликів.

Мобільні технології за останнє десятиліття зазнали суттєвих трансформацій, що спричинило зміни у способах взаємодії користувача з пристроєм. Одним з основних напрямів удосконалення є розробка природних

інтерфейсів, зокрема голосових, які забезпечують більш інтуїтивну, гнучку та доступну форму керування додатками. Голосове введення набуло особливої актуальності завдяки його здатності зменшувати час взаємодії з пристроєм, підвищувати ефективність роботи та забезпечувати інклюзивний доступ до цифрових сервісів, зокрема для людей з обмеженими можливостями.

Платформа Android, як одна з наймасовіших мобільних операційних систем у світі, надає потужні інструменти для реалізації мовленнєвих інтерфейсів, зокрема за допомогою API Google Voice Recognition. Проте інтеграція функціональності розпізнавання мовлення у середовище Android Java-розробки, особливо з орієнтацією на генерацію або автоматизацію створення коду, залишається малодослідженою і технічно складною. Зокрема, виникають виклики, пов’язані з обмеженнями апаратних ресурсів мобільних пристроїв, нестабільністю мережевого з’єднання, багатомовністю користувачів, а також точністю розпізнавання термінологічно специфічної лексики, властивої програмуванню.

У зв’язку з цим зростає потреба у побудові системи, яка здатна ефективно сприймати голосові команди користувача-розробника, розпізнавати їх зміст та перетворювати у структуровані інструкції, що можуть бути використані у програмному коді. Такий підхід не лише відкриває нові горизонти в розробці Android-додатків, але й сприяє прискоренню процесу створення програмного забезпечення, підвищенню продуктивності та якості розробки.

Метою даної роботи є розробка прототипу системи, яка дозволяє інтегрувати голосові команди в процес Android Java-розробки з використанням сучасних систем розпізнавання мовлення.

Завдання дослідження

- Проаналізувати сучасний стан і перспективи використання голосового введення в Android-додатках.

- Дослідити алгоритми та бібліотеки для розпізнавання мовлення, які можливо адаптувати для Android Java-розробки.

- Розробити архітектуру системи з урахуванням апаратних і програмних вимог.

- Реалізувати функціональний прототип інтеграції голосових команд у середовищі Android.

- Провести тестування реалізованого рішення та оцінити його ефективність.

Об'єкт дослідження - процес розробки мобільних додатків на платформі Android із використанням мови Java.

Предмет дослідження - методи та інструменти реалізації голосових команд і систем розпізнавання мовлення у процесі Android Java-програмування.

Методи дослідження

- Аналіз і синтез літературних джерел та технічної документації.

- Метод порівняльного аналізу існуючих бібліотек розпізнавання мовлення.

- Моделювання архітектури системи за допомогою UML-діаграм.

- Експериментальна реалізація програмного забезпечення.

- Системне та модульне тестування.

Наукова новизна

Запропоновано та реалізовано концепцію інтеграції голосових команд у процес Android Java-програмування з урахуванням мінімальних вимог до системи, що дозволяє підвищити доступність мобільної розробки та автоматизувати частину процесів через голосове керування.

У рамках дослідження розглянуто еволюцію мобільної платформи Android, проведено огляд алгоритмів розпізнавання мовлення, описано архітектурні й алгоритмічні рішення, що дозволяють ефективно обробляти голосові запити в середовищі розробника.

Практичне застосування

Результати роботи мають прикладне значення для подальших досліджень у сфері голосових інтерфейсів та створення інтелектуальних систем підтримки мобільної розробки.

Бакалаврська робота містить 77 сторінок, 18 рисунків, 3 розділи список використаних джерел із 34 найменуваннями.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ІМПЛЕМЕНТАЦІЇ ГОЛОСОВИХ КОМАНД І РОЗПІЗНАВАННЯ МОВЛЕННЯ В ПРОЦЕС АНДРОЇД JAVA-РОЗРОБКИ

1.1. Передумови розробки системи голосового введення для Android Java-програмування

На сучасному етапі розвитку інформаційних технологій мобільні пристрої, що функціонують під управлінням операційної системи Android (ОС Android), посідають значне місце на ринку та мають багатомільйонну базу користувачів у всьому світі. Популярність даної операційної системи детермінована такими чинниками, як її багатозадачність, високий рівень доступності та широкий спектр сумісних апаратних засобів.

У контексті підвищення доступності інструментів розробки програмного забезпечення, особливо для осіб з обмеженими фізичними можливостями, виникає потреба в альтернативних методах взаємодії з комп'ютерними системами. "Додаток інтеграції голосових команд в процес Андроїд Java-розробки" є спеціалізованим програмним рішенням для платформи Android, розробленим з метою надання можливості користувачам, які мають труднощі з традиційним клавіатурним введенням, здійснювати написання програмного коду мовою Java шляхом використання голосових команд.

Функціональність зазначеного додатку ґрунтується на системі розпізнавання мовлення, яка конвертує голосовий ввід користувача в текстовий формат. Далі відбувається аналіз отриманого тексту для ідентифікації ключових елементів синтаксису мови Java (таких як boolean, break, if, else тощо), а також інших необхідних символів та команд, які є складовими частинами програмного коду. Таким чином, додаток дозволяє

формувати програмний код виключно за допомогою голосу, мінімізуючи або повністю усуваючи необхідність використання фізичної клавіатури.

Технічна реалізація "Додатку інтеграції голосових команд в процес Андроїд Java-розробки" передбачає інтеграцію зовнішніх програмних компонентів. Зокрема, використовуються спеціалізовані бібліотеки та попередньо розроблені модулі для реалізації функцій редактора програмного коду із синтаксичним підсвічуванням та, як вже згадувалося, системи розпізнавання мовлення. Ці компоненти підключаються до основної архітектури додатку під час його розробки, забезпечуючи необхідний функціонал для коректної обробки голосового вводу та його перетворення у структурований програмний код Java в межах інтегрованого середовища розробки на мобільній платформі. Така модульна структура спрощує процес розробки та оновлення, дозволяючи використовувати оптимізовані та перевірені рішення для ключових завдань.

1.2. Вступ до концепції мобільних додатків та платформи Android у контексті доступності розробки програмного забезпечення

Мобільні додатки є формою прикладного програмного забезпечення, спеціально розробленого для функціонування на портативних електронних пристроях, зокрема смартфонах та планшетних комп'ютерах. Імплементація мобільних додатків суттєво розширює первинну функціональність апаратних засобів, надаючи користувачам можливість ефективно реалізовувати широкий спектр завдань в інтерактивному та орієнтованому на користувача цифровому середовищі. Традиційно значна частка мобільних додатків, особливо для платформи Android, розроблялася з використанням мови програмування Java, доповненої специфічними для операційної системи фреймворками та інструментаріями, що забезпечують кросплатформну

сумісність та адаптацію функціоналу під різні операційні системи, такі як Android та iOS.

Операційна система Android являє собою домінуючу мобільну платформу, архітектура якої базується на модифікованому ядрі Linux. Розроблена корпорацією Google у співпраці з Open Handset Alliance, ОС Android була первинно спроєктована для оптимізації взаємодії з пристроями, оснащеними сенсорними екранами, що спричинило трансформаційні зміни у галузі мобільних технологій. На поточний момент Android є найпоширенішою мобільною операційною системою, що використовується в продуктах широкого кола виробників смартфонів, планшетів та інших типів обчислювальних пристроїв. Враховуючи її глобальне поширення, екосистема Android характеризується наявністю мільйонів різноманітних програмних додатків, розроблених для задоволення потреб користувачів у всьому світі.

У прагненні до підвищення інклюзивності та доступності процесів розробки програмного забезпечення, особливу актуальність набуває застосування альтернативних методів введення інформації. Зокрема, спеціалізовані додатки для платформи Android, що інтегрують функціонал розпізнавання мовлення, відкривають нові можливості. Такий додаток, який можна класифікувати як систему голосового введення програмного коду, призначений для конвертації усних команд, фраз та інструкцій користувача у відповідний текстовий формат, придатний для використання як елементи програмного коду. Це не лише потенційно прискорює процес кодування для досвідчених розробників, але й, що є критично важливим, надає можливість особам з обмеженими фізичними можливостями, які мають труднощі або не можуть здійснювати традиційне клавіатурне введення, брати участь у створенні програмного забезпечення. Ефективність такого додатка залежить від здатності системи розпізнавання мовлення точно ідентифікувати ключові слова мови Java, назви класів та бібліотек, оператори, символи та інші синтаксичні конструкції, необхідні для генерації синтаксично та семантично

коректного коду, готового до подальшої компіляції та виконання в інтегрованому середовищі розробки.

1.3. Системи розпізнавання мовлення та інтегровані середовища розробки для розробки на платформі Android

Операційна система Android надає доступ до широкого спектру програмних застосувань, зокрема тих, що забезпечують функціональність перетворення усного мовлення у текстовий формат, а також різноманітних інструментів для розробки програмного забезпечення, таких як редактори коду та інтегровані середовища розробки (IDE), що дозволяють користувачам здійснювати написання комп'ютерних програм в оптимізованому для користувача середовищі.

Системи розпізнавання мовлення (Speech-to-Text) являють собою технології, призначені для автоматичного перетворення аудіосигналу мовлення людини в текстовий формат. Процес зазвичай включає акустичний аналіз мовленнєвого сигналу, його сегментацію та зіставлення з фонетичними моделями, а потім використання лінгвістичних моделей та словників для визначення найбільш ймовірної послідовності слів. На платформі Android такі системи можуть бути реалізовані як хмарні сервіси (що вимагають підключення до Інтернету) або локальні компоненти, що функціонують без мережевого доступу (залежно від обсягу мовної моделі). Яскравим прикладом такої системи є сервіс Google Speech Recognizer, який доступний через стандартні прикладні програмні інтерфейси (API) Android і активно використовується в багатьох додатках, включаючи такі, як Google Voice, для реалізації функції голосового введення тексту.

На рисунку 1.1 зображено архітектуру системи обробки аудіофайлів з використанням сервісів Google Cloud Platform для перетворення мовлення у

текст. Ця архітектура ілюструє типовий серверний підхід до реалізації функціоналу розпізнавання мовлення.

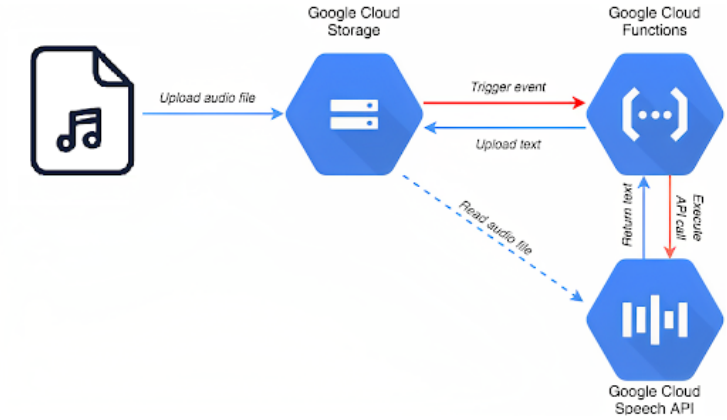

Рисунок 1.1 - Архітектура системи обробки аудіофайлів з використанням сервісів Google Cloud Platform

Ключові компоненти та їх взаємодія представлені наступним чином:

1. Вхідний аудіо файл позначає початкові дані – аудіофайл, що містить мовлення для розпізнавання.

2. Google Cloud Storage - компонент є масштабованим об'єктним сховищем даних. На рисунку показано, що аудіофайл завантажується (Upload audio file) саме сюди. Сховище виступає як централізоване місце для зберігання вхідних та вихідних даних.

3. Google Cloud Functions - це безсерверний обчислювальний сервіс, що дозволяє виконувати код у відповідь на події. На рисунку 1.1 показано, що завантаження аудіофайлу в Cloud Storage спричиняє подію (Trigger event), яка активує функцію у Google Cloud Functions. Ця функція є оркестратором процесу. Вона читає аудіофайл (Read audio file) з Cloud Storage (показано пунктирною стрілкою, що вказує на отримання даних) та виконує подальші дії.

4. Google Cloud Speech API - спеціалізований сервіс Google Cloud, призначений для розпізнавання мовлення та перетворення його у текст (Speech-to-Text). Функція з Google Cloud Functions здійснює виклик API

(Execute API call) до цього сервісу, передаючи йому аудіодані для обробки. У відповідь Google Cloud Speech API повертає текст (Return text) розпізнаного мовлення назад до функції в Google Cloud Functions.

Завершальним етапом процесу, зображеного на рисунку 1.1, є те, що функція в Google Cloud Functions завантажує отриманий текст (Upload text) назад у Google Cloud Storage. Це може бути окремий файл з текстом або метадані, пов'язані з оригінальним аудіофайлом.

Таким чином, представлена архітектура демонструє автоматизований workflow: завантаження аудіофайлу ініціює виконання безсерверної функції, яка взаємодіє з сервісом розпізнавання мовлення для отримання текстової транскрипції та її збереження. Цей підхід дозволяє ефективно обробляти аудіодані без необхідності керувати власною серверною інфраструктурою.

Паралельно з інструментами введення, для ефективної розробки програмного забезпечення на мобільних пристроях необхідні відповідні редактори програмування або інтегровані середовища розробки (IDE). Ці інструменти надають функціонал для написання, редагування та управління програмним кодом, часто включаючи синтаксичне підсвічування, автодоповнення коду, інструменти налагодження та управління проєктами. На операційній системі Android існує низка таких рішень, що ілюструють можливості платформи для мобільної розробки. Серед прикладів відомих редакторів та IDE можна назвати Deuter IDE, Code Peeker та AIDE (Android IDE), які надають розробникам необхідний інструментарій для створення та компіляції додатків безпосередньо на мобільному пристрої.

Наявність на платформі Android як розвинених систем розпізнавання мовлення, так і функціональних середовищ розробки створює передумови для інтеграції цих технологій з метою створення інноваційних інструментів, зокрема для голосового написання програмного коду, що є актуальним напрямом у контексті підвищення доступності процесів розробки програмного забезпечення.

1.4. Проблематика застосування стандартних систем розпізнавання мовлення для генерації програмного коду

Аналіз функціональності традиційних систем автоматичного розпізнавання мовлення (ASR) виявляє їхні суттєві обмеження при спробах застосування для специфічних завдань, таких як написання програмного коду. Типова імплементація ASR спрямована на транскрибування мовленнєвого потоку у текстовий формат без урахування його семантичного чи синтаксичного контексту, що робить її придатною для обробки природної мови, але неструктурованого або неспецифікованого контенту.

Відповідно, при спробі використання загальнопризначених ASR-систем для генерації програмного коду виникає фундаментальна проблема: мови програмування, зокрема Java, є формальними мовами зі строго визначеними синтаксичними правилами, ключовими словами, ідентифікаторами та структурними елементами. Користувач-програміст мусить дотримуватися цього жорсткого синтаксису та формату. Існуючі системи розпізнавання мовлення, що не орієнтовані на специфіку програмування, не можуть забезпечити таке точне та структуроване введення, оскільки вони оперують широким лексиконом природної мови та не мають вбудованого розуміння синтаксису коду. Це означає, що пряме диктування коду за допомогою стандартного ASR призведе до появи у тексті слів та фраз, що не мають відношення до синтаксису або структур даних мови програмування, роблячи згенерований текст синтаксично некоректним та непридатним для компіляції.

З метою подолання цих обмежень та забезпечення можливості голосового написання програм, необхідна розробка або адаптація ASR-системи, яка буде спеціалізована для домену програмування. Удосконалена імплементація такого додатку повинна не просто перетворювати мовлення на текст, але й активно розпізнавати та інтерпретувати синтаксичні конструкції мови Java. Це передбачає інтеграцію специфічної мовної моделі, навченої на

корпусах програмного коду Java, та, ймовірно, використання синтаксичного парсера, який буде валідувати та структурувати розпізнаний ввід відповідно до правил мови.

Така система має обмежити лексикон розпізнавання виключно тими словами та символами, що є валідними в контексті синтаксису Java та оголошених бібліотек/ідентифікаторів проєкту, відхиляючи або ігноруючи слова природної мови, що не відповідають цим правилам.

Крім того, функціональність повинна передбачати механізм для введення коментарів – елементів коду, які ігноруються компілятором і призначені для пояснення логіки програми природною мовою. Це може бути реалізовано за допомогою спеціального активаційного слова або команди (наприклад, диктування "коментар"), яка тимчасово перемикатиме систему в режим вільного текстового введення (стандартного ASR) для запису змісту коментаря, після чого система автоматично повертатиметься в режим розпізнавання коду.

Таким чином, покращена система розпізнавання мовлення для програмування вимагає інтеграції доменних знань (синтаксису та лексики мови програмування) у процес розпізнавання, що дозволить генерувати синтаксично коректний програмний код безпосередньо з голосового вводу.

1.5. Пропонована архітектура та функціональність системи голосового кодування

Представлена система являє собою спеціалізований програмний додаток, розроблений для платформи Android, що надає користувачам можливість здійснювати розробку програмного забезпечення мовою Java шляхом використання голосового введення. Ключова функціональність системи полягає у трансформації голосового вводу користувача у відповідний текстовий формат, що відповідає синтаксичним правилам мови

Java. Це досягається шляхом інтеграції вдосконаленої системи розпізнавання мовлення, адаптованої для розпізнавання специфічного лексикону та синтаксичних конструкцій, характерних для програмування на Java. По завершенні етапу голосового редагування коду, система забезпечує можливість збереження згенерованого файлу з вихідним кодом та його подальшу інтеграцію у компілятор та середовище виконання для верифікації та виконання програми.

Процес експлуатації системи передбачає структурований підхід до голосового введення коду. Користувач починає роботу з диктування назви класу, з наступним визначенням його елементів (полів, методів тощо). Система дозволяє реалізувати імпорт необхідних пакетів та бібліотек на будь-якій стадії розробки, забезпечуючи гнучкість у створенні структури програми.

Однією з переваг запропонованої системи, що оптимізує процес кодування, є використання преконфігурованих синтаксичних конструкцій та шаблонів. Замість необхідності диктувати кожен окремий базовий компонент стандартних блоків коду (наприклад, оголошення головного методу main), користувачеві достатньо активувати відповідну функцію за допомогою ключового слова (наприклад, "main"). У відповідь система автоматично генерує базову структуру даного блоку, включаючи необхідні дужки, ключові слова та маркери для аргументів або тіла методу, що суттєво прискорює написання типових фрагментів коду. По завершенні сеансу кодування та готовності програми до перевірки, користувач може активувати команду (наприклад, "commit"), яка ініціює процес збереження поточного стану вихідного коду та його подальшу компіляцію та виконання в інтегрованому середовищі.

Для коректного функціонування запропонованої системи голосового кодування на платформі Android необхідне дотримання наступних апаратних та програмних вимог.

1.5.1. Мінімальні апаратні вимоги

Мобільний пристрій: Пристрій, що працює під управлінням операційної системи Android версії 2.2 або новішої.

Процесор: Тактова частота центрального процесора має становити не менше 500 МГц.

Оперативна пам'ять (RAM): Мінімальний обсяг оперативної пам'яті – 170 МБ.

Пам'ять для зберігання даних: Обсяг зовнішньої (наприклад, карта пам'яті SD) або внутрішньої пам'яті має бути не менше 512 МБ для зберігання додатку, його компонентів та файлів проєктів.

Режим налагодження: На пристрої має бути активований режим налагодження через USB (USB debugging), що може бути необхідним для встановлення, тестування або взаємодії з середовищем розробки.

1.5.2. Мінімальні програмні вимоги

Операційна система: Мобільна операційна система Android версії 2.2 (Froyo) або вище.

Середовище розробки (для створення додатку): Використання інтегрованих середовищ розробки, таких як Eclipse з ADT (Android Development Tools) або Android Studio, є необхідним для розробки та компіляції самого додатку.

Реалізація користувацького інтерфейсу: Інтерфейс користувача системи реалізується за допомогою мови розмітки XML.

Кодова база: Основна логіка та функціональність додатку імплементується на мовах програмування JAVA та XML (для UI).

Мережеве з'єднання: Наявність стабільного доступу до мережі Інтернет є обов'язковою, що, ймовірно, пов'язано з використанням хмарних сервісів розпізнавання мовлення або для завантаження компонентів/бібліотек.

Висновки до першого розділу

У цьому розділі проведено ґрунтовний аналіз передумов і контексту впровадження систем голосового введення у сфері Android-розробки. Здійснено огляд платформи Android як середовища, що підтримує мобільне програмування та забезпечує доступ до різноманітних API для роботи з мовленням. Проаналізовано існуючі системи розпізнавання мовлення та визначено їхні обмеження, зокрема у випадку спроби генерації програмного коду на основі голосових команд. Було запропоновано концепцію та архітектуру системи голосового кодування, окреслено її функціональні вимоги, а також визначено мінімальні апаратні та програмні умови для її реалізації. Розділ створює теоретичну базу для подальшої розробки інструменту, що забезпечує голосову інтеракцію в межах Android Java-програмування.

РОЗДІЛ 2. АЛГОРИТМИ ТА МОДЕЛІ РОЗПІЗНАВАННЯ МОВЛЕННЯ ДЛЯ ANDROID РОЗРОБКИ

2.1. Еволюція та характеристики мобільної операційної системи Android

Платформа Android була вперше заснована як компанія в жовтні 2003 року в Пало-Альто, Каліфорнія, чотирма співзасновниками: Енді Рубіним, Річем Майнером, Ніком Сірсом та Крісом Вайтом. Початковою метою компанії була розробка більш розумних мобільних пристроїв, які б усвідомлювали місцезнаходження та вподобання користувача. Придбання Android корпорацією Google, яке відбулося 17 серпня 2005 року, стало знаковою подією, що викликала припущення на ринку щодо стратегічного наміру Google вийти на ринок мобільної телефонії.

Під егідою Google, команда під керівництвом Рубіна зосередилася на розробці мобільної платформи, що базується на ядрі операційної системи Linux. Вибір Linux як основи був зумовлений його відкритим кодом, стабільністю та гнучкістю, що дозволяло створити надійну та адаптовану систему для мобільних пристроїв. Google позиціонував Android як гнучку та оновлювану системну платформу, що надає широкі можливості як користувачам, так і виробникам пристроїв. Ця фундаментальна характеристика, реалізована через модульну архітектуру та механізми оновлення "по повітрю" (Over-The-Air, OTA), сприяла подальшій швидкій еволюції та досягненню поточного стану розвитку платформи Android.

Android є мобільною операційною системою, спочатку розробленою переважно для смартфонів, планшетів та пристроїв із сенсорним інтерфейсом. Проте, завдяки своїй адаптивності та відкритому характеру, її застосування значно розширилося. На сьогоднішній день Android використовується у широкому спектрі інших електронних пристроїв,

включаючи ігрові консолі, інформаційно-розважальні системи автомобілів (Android Automotive), телевізори (Android TV), пристрої Інтернету речей (IoT) та певні типи персональних комп'ютерів (наприклад, через емулятори або спеціалізовані дистрибутиви).

За короткий термін Android досяг значної ринкової домінантності. Згідно з даними про продажі за 2013 рік, пристрої під управлінням Android демонстрували вищі обсяги реалізації порівняно з пристроями, що використовували такі операційні системи як Microsoft Windows, iOS та Mac OS. Ця тенденція домінування продовжилася і в наступні роки, закріпивши за Android статус найпоширенішої мобільної операційної системи у світі.

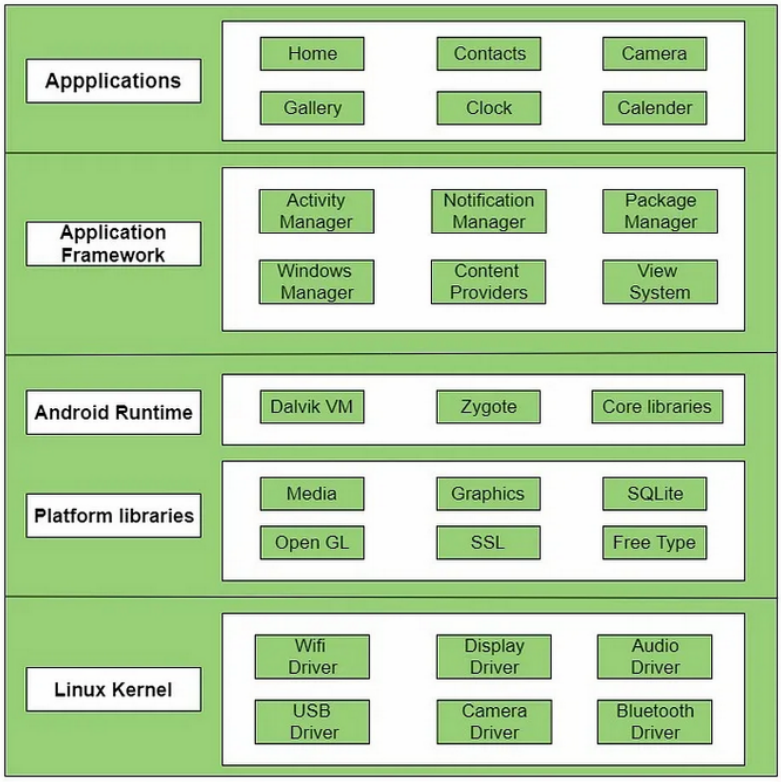

Рисунок 2.1 – Архітектура Android

Центральним елементом екосистеми Android є магазин додатків Google Play. Станом на початок 2025 року, кількість додатків, доступних у магазині Google Play, оцінюється в діапазоні приблизно від 2 до 3.95 мільйонів додатків для платформи Android, а загальна кількість завантажень додатків

перевищила 200 мільярдів. Ця масштабна база додатків є ключовим фактором привабливості платформи для кінцевих користувачів. Високий рівень проникнення Android на ринок також стимулював активність розробників. На сьогоднішній день кількість активних користувачів пристроїв на базі Android перевищує три мільярди.

Інструментарій програмного забезпечення, що використовується для розробки під Android, включає як компоненти з відкритим кодом, так і власницьке програмне забезпечення, розроблене Google. Популярність Android значною мірою зумовлена його "нульовою вартістю" ліцензування для виробників обладнання та високим ступенем налаштовуваності операційної системи, що дозволяє адаптувати її для широкого спектру пристроїв, включаючи високотехнологічні моделі. Відкритий характер платформи Android (через проект Android Open Source Project, AOSP) є фундаментальною перевагою. Він дозволяє розробникам та спільнотам використовувати вихідний код як основу для власних проектів, створювати користувацькі версії (custom ROMs) та впроваджувати інноваційні функції. Це сприяє швидшому розвитку, більшій прозорості та можливості кастомізації як на рівні системи, так і на рівні окремих пристроїв, надаючи просунутим користувачам та виробникам обладнання значну гнучкість.

2.2. Принципи та імплементація в Android автоматичного розпізнавання мовлення

Автоматичне розпізнавання мовлення (Automatic Speech Recognition, ASR), також відоме як комп'ютерне розпізнавання мовлення, розпізнавання мовлення (Speech Recognition, SR) або перетворення мовлення на текст (Speech-to-Text, STT), є галуззю досліджень, що спрямована на трансформацію усного мовлення людини у текстову форму за допомогою машини. Ця галузь має історію досліджень, що триває понад шість

десятиліть, і є об'єктом інтенсивного наукового інтересу. Дослідження в сфері машинного розпізнавання мовлення є міждисциплінарними і охоплюють значний спектр галузей, включаючи, але не обмежуючись, обробку сигналів, акустику, розпізнавання образів, теорію комунікацій та інформації, лінгвістику, фізіологію, комп'ютерні науки та психологію.

2.2.1. Алгоритм системи розпізнавання мовлення

На рисунку 2.2 зображено блок-схему системи розпізнавання мовлення, яка, може представляє модель для ізольованого розпізнавання мовлення на основі прихованих марковських моделей (Hidden Markov Model, HMM), хоча сама схема має загальніші компоненти систем розпізнавання мовлення.

Розпізнавання мовлення на основі прихованих марковських моделей є одним із найпоширеніших і найефективніших підходів, особливо в класичних системах ASR (Automatic Speech Recognition). Цей метод базується на імовірнісній моделі, яка дозволяє моделювати часові процеси — зокрема, мовлення, що є послідовністю звуків, які змінюються з часом.

HMM — це статистична модель, яка описує систему, що перебуває в одному з кількох прихованих станів. У контексті мовлення ці стани зазвичай відповідають фонемам або їх частинам (наприклад, "begin", "middle", "end" фонеми). У кожен момент часу система знаходиться в певному стані, але цей стан не спостерігається безпосередньо. Замість цього, для кожного стану спостерігається вихід (output) — наприклад, спектральний вектор, який моделює звук.

Розглянемо етапи розпізнавання мовлення за допомогою HMM.

1. Акустичне моделювання.

Голосовий сигнал перетворюється в послідовність векторів ознак (наприклад, MFCC), які відповідають коротким фреймам мовлення.

2. Лексичне моделювання.

Кожне слово описується як послідовність фонем, а кожна фонема — як HMM.

3. Пошук (декодування)

За допомогою алгоритмів (наприклад, алгоритму Вітербі) визначається найбільш ймовірна послідовність станів (фонем/слів), яка пояснює отримані акустичні спостереження.

Переваги використання HMM:

- Добре моделює часову природу мовлення.

- Має зрозумалу математичну інтерпретацію.

- Широко підтримується існуючими бібліотеками та інструментами (CMU Sphinx, HTK тощо).

- Довгий час був стандартом у промислових ASR-системах.

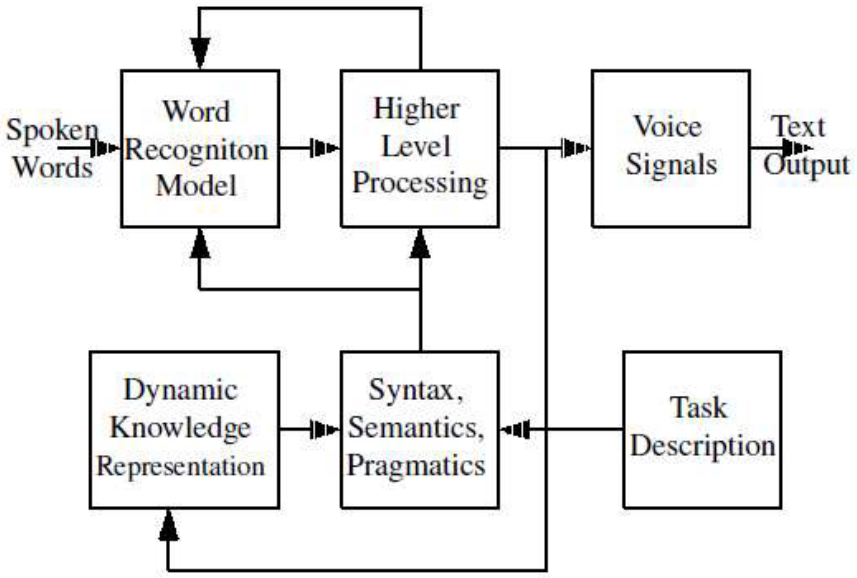

Рисунок 2.2 – Блок-схема системи розпізнавання мовлення

Система складається з наступних ключових блоків та зв'язків:

1. Spoken Words (Вимовлені Слова): Це вхідний сигнал системи – звукова форма мовлення, що вимовляється користувачем.

2. Word Recognition Model (Модель Розпізнавання Слів): Цей блок, імовірно, відповідає за початкову обробку вхідного звукового сигналу та

його сегментацію на потенційні слова або фонетичні одиниці. Він порівнює акустичні патерни вхідного сигналу з попередньо навченими акустичними моделями слів. Напрямок стрілок вказує, що вхідні вимовлені слова надходять до цього блоку. Також є зворотний зв'язок від блоку "Dynamic Knowledge Representation", що свідчить про можливість адаптації або використання контекстуальних знань на етапі розпізнавання слів.

3. Dynamic Knowledge Representation (Динамічне Представлення Знань): Цей блок, ймовірно, містить інформацію про динамічні аспекти мовлення, контекст або модель користувача/сесії, яка може впливати на процес розпізнавання. Він пов'язаний з "Word Recognition Model" (зворотний зв'язок) та "Syntax, Semantics, Pragmatics".

4. Syntax, Semantics, Pragmatics (Синтаксис, Семантика, Прагматика): Цей блок представляє лінгвістичні моделі, які використовуються для аналізу послідовності розпізнаних слів на вищому рівні. Він перевіряє граматичну правильність (синтаксис), змістове наповнення (семантика) та контекстуальне значення (прагматика) для уточнення та корекції результатів початкового розпізнавання слів. Стрілки вказують, що він отримує інформацію від "Dynamic Knowledge Representation" та "Higher Level Processing". Він також надсилає зворотний зв'язок до "Higher Level Processing" та "Word Recognition Model", що вказує на ітеративний процес уточнення результатів.

5. Higher Level Processing (Обробка Вищого Рівня): Цей блок, ймовірно, інтегрує результати початкового акустичного розпізнавання від "Word Recognition Model" з лінгвістичним аналізом від "Syntax, Semantics, Pragmatics". Він може виконувати подальшу обробку, таку як вирішення неоднозначностей та формування остаточної послідовності слів. Він отримує вхід від "Word Recognition Model" та "Syntax, Semantics, Pragmatics" і надсилає вихід до "Voice Signals" та "Syntax, Semantics, Pragmatics" (зворотний зв'язок).

6. Task Description (Опис Завдання): Цей блок містить інформацію про специфіку завдання, для якого використовується система розпізнавання (наприклад, диктування тексту, голосове керування певною програмою). Ця інформація може впливати на лінгвістичні моделі ("Syntax, Semantics, Pragmatics") та/або обробку вищого рівня ("Higher Level Processing").

7. Voice Signals (Голосові Сигнали): Цей блок, розміщений перед текстовим виходом, може представляти етап перетворення внутрішнього представлення розпізнаного мовлення назад у форму, придатну для виведення тексту. Його зв'язок з "Higher Level Processing" та пряма стрілка до "Text Output" підтверджують його роль у фіналізації результату. Можливо, назва блоку не зовсім точна і мала б стосуватися обробки результатів розпізнавання для виводу тексту.

8. Text Output (Текстовий Вихід): Це кінцевий результат роботи системи – розпізнаний текст, отриманий з вихідного усного мовлення.

Система отримує на вхід усне мовлення. "Word Recognition Model" здійснює початкове акустичне розпізнавання. Результати цього розпізнавання надходять на "Higher Level Processing", де вони інтегруються з лінгвістичним аналізом від "Syntax, Semantics, Pragmatics". Лінгвістичний аналіз використовує знання про синтаксис, семантику та прагматику, а також може враховувати динамічний контекст ("Dynamic Knowledge Representation") та специфіку завдання ("Task Description"). Існує значна кількість зворотних зв'язків між блоками "Word Recognition Model", "Higher Level Processing", "Syntax, Semantics, Pragmatics" та "Dynamic Knowledge Representation", що вказує на ітеративний характер процесу розпізнавання, де результати аналізу вищих рівнів можуть впливати на переоцінку та уточнення результатів нижчих рівнів. Фіналізований результат передається через блок "Voice Signals" до "Text Output" як розпізнаний текст.

Ця схема, що представлена на рисунку 2.2 відображає типовий конвеєр обробки в системах розпізнавання мовлення, який поєднує акустичне

моделювання з мовними моделями для досягнення високої точності розпізнавання.

2.2.2. Мобільні системи розпізнавання мовлення та Google Voice Recognition (GVR)

На сучасному етапі мобільні імплементації систем розпізнавання мовлення (SR) є широко поширеними. Існує значна кількість сторонніх SR-додатків, сумісних з платформою Android. Як приклад та механізм розпізнавання для подальшого розгляду був обраний Google Voice Recognition (GVR), який часто постачається попередньо встановленим на багатьох пристроях Android [11].

GVR використовує алгоритми, що базуються на архітектурі нейронних мереж, для перетворення аудіоданих людського мовлення у текст. Система розроблена для роботи з рядом основних мов, хоча специфічне застосування може вимагати використання лише однієї мови (наприклад, англійської). Нейронна мережа GVR складається з численних взаємопов'язаних обчислювальних елементів (імітуючих нейрони), які функціонують паралельно. Така паралельна обробка забезпечує необхідну обчислювальну потужність для ефективного розпізнавання мовлення в реальному часі.

Ключовою характеристикою нейронних мереж, що використовується в GVR, є їхня здатність до адаптивного навчання та набуття знань. На відміну від систем, що виконують строго визначені, попередньо запрограмовані алгоритми, нейронна мережа навчається ідентифікувати патерни та взаємозв'язки в даних на основі великих обсягів навчальних прикладів. Хоча GVR може підтримувати обмежений офлайн-режим на деяких моделях пристроїв Android, його повноцінне функціонування, як правило, вимагає підключення до Інтернету, зазвичай через Wi-Fi. Це зумовлено необхідністю взаємодії з масштабною хмарною базою даних Google, яка містить агреговані дані мовлення та контекстуальну інформацію, отриману, зокрема, з

попередніх пошукових запитів користувачів. Аналіз такої інформації дозволяє системі GVR оцінювати статистичну ймовірність використання певних фраз, що оптимізує процес розпізнавання.

Загалом, навчання нейронних мереж класифікується за двома основними парадигмами: кероване (supervised learning) та самоорганізоване (unsupervised/self-organizing learning). У парадигмі керованого навчання системі надаються позначені вхідні дані разом із відповідними очікуваними вихідними результатами, і мережа навчається зіставляти входи з виходами. Натомість, самоорганізована мережа отримує лише вхідні дані і самостійно виявляє приховані структури, кластери або патерни в цих даних без попередньо визначених міток. Згідно з джерелом [11], GVR навчається, використовуючи метод самоорганізації на основі своєї внутрішньої бази даних.

2.2.3. Взаємодія клієнта Android з GVR та комунікаційні аспекти

Клієнтська частина застосунку на платформі Android, що інтегрує функціональність GVR, виконує ряд завдань. Ці завдання включають управління графічним інтерфейсом користувача, прийом та обробку текстової інформації, що надходить від механізму GVR, а також забезпечення комунікації із зовнішнім сервером (у разі потреби обміну даними). Як уже згадувалося, сам механізм GVR вимагає стабільного підключення до Інтернету (бажано через Wi-Fi) для ефективної взаємодії з віддаленою хмарною інфраструктурою Google.

Слід підкреслити, що час виконання кожного з цих завдань є варіабельним. Зокрема, значні флуктуації можуть спостерігатися в часі, необхідному для мережевої комунікації, що прямо залежить від таких чинників, як пропускна здатність каналу зв'язку, затримка (latency) та стабільність з'єднання.

2.3. Опис бібліотек для розпізнавання мовлення

Google Voice Recognizer — це додаток, розроблений компанією Google. Цей додаток широко використовується в продуктах Google для ПК та операційних систем Android. Цей додаток дозволяє нам друкувати за допомогою голосу, тобто, диктуючи слова, додаток отримує вхідні дані, аналізує їх та перетворює на текст.

Google Voice Recognizer був революційним додатком свого часу та приніс багато нових функцій у системи Android. Наприклад, користувач пристрою може продиктувати речення для пошуку в Google, і додаток отримає вхідні дані, перетворить їх на текст, виконає пошук у Google та покаже результат користувачеві.

Сьогодні багато розробників додатків використовують розпізнавання мовлення у своїх додатках для більшої функціональності. Щоб полегшити роботу розробникам додатків, Android додав попередньо визначений API для розпізнавання мовлення до своєї бібліотеки. Таким чином, розробникам потрібно лише додати бібліотеку до свого додатку та викликати правильну функцію та метод у своєму класі Java.

Щоб написати автономний розпізнавач мовлення, який розпізнає продиктоване слово та перетворює його на текст, рекомендується імпортувати наступний JAR-файл у клас Java [2].

Лістинг 2.1. Імпорт бібліотек

```
import android.speech.RecognitionListener;
import android.speech.RecognizerIntent;
import android.speech.SpeechRecognizer;
import android.app.Activity;
import android.content.Intent;
import android.view.View;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.ToggleButton;
```

Пакет android.speech. Цей пакет містить один інтерфейс (RecognitionListener) та п'ять класів (RecognitionService, RecognitionService.Callback, RecognizerIntent, RecognizerResultsIntent та SpeechRecognizer).

Таблиця 2.1 – Перелік класів

RecognitionService	Цей клас надає базовий клас для реалізацій служби розпізнавання.
RecognitionService.Callback	Цей клас отримує зворотні виклики від служби розпізнавання мовлення та передає їх користувачеві.
RecognizerIntent	Константи для підтримки розпізнавання мовлення через запуск Intent
RecognizerResultsIntent	Константи для намірів, пов'язаних із показанням результатів розпізнавання мовлення.
SpeechRecognizer	Цей клас надає доступ до служби розпізнавання мовлення.

Пакет android.app. "Цей пакет містить класи високого рівня, які інкапсулюють загальну модель додатку Android. Додаток може бути визначений за допомогою одного або кількох з чотирьох основних компонентів додатку Android. Компоненти Activity та Service визначені в цьому пакеті. Activity може запускати інші активності, включаючи активності, які живуть у окремих додатках. Service — це компонент додатку, який надає екран, з яким користувач може взаємодіяти для виконання дії. Наприклад, служба може обробляти відтворення музики, мережеве з'єднання або роботу з браузером без відома користувача про роботу, що відбувається" [2].

Пакет android.widget. "Щоб створити власний віджет, розширте View або підклас. Щоб використовувати свій віджет у XML-макеті, є два додаткові файли, які вам потрібно створити. Ось список файлів, які вам потрібно створити для реалізації користувацького віджета" [4].

Файл реалізації Java. "Це файл, який реалізує поведінку віджета. Якщо ви можете створити об'єкт з XML-макету, вам також доведеться закодувати конструктор, який отримує всі значення атрибутів з файлу XML-макету" [2].

Файл визначення XML. "XML-файл у res/values/, який визначає XML-елемент, що використовується для створення вашого віджета, та атрибути, які він підтримує. Інші додатки використовуватимуть цей елемент та атрибути в іншому XML-макеті".

XML-макет. "Необов'язковий XML-файл у res/layout/, який описує макет вашого віджета. Ви також можете зробити це в коді у вашому Java-файлі. Приклад додатка API Demos демонструє створення користувацького тегу XML-макету, LabelView. Дивіться наступні файли, які демонструють реалізацію та використання користувацького віджета".

Клас LabelView Java: файл реалізації.

Res_values_attrs.xml — файл визначення.

Res/layout/custom_view_1.xml — файл макету.

Пакет android.view. "Надає класи, які відкривають базові класи інтерфейсу користувача, які обробляють макет екрану та взаємодію з користувачем".

2.4. Алгоритмічні основи сучасних систем розпізнавання мовлення

Сучасні системи автоматичного розпізнавання мовлення (ASR) функціонують на основі двох фундаментальних компонентів: акустичного моделювання та мовного моделювання. Акустичне моделювання зосереджується на зв'язку між акустичними сигналами (звуками мовлення) та фонемами або іншими елементарними одиницями мови, тоді як мовне моделювання аналізує послідовності слів для передбачення ймовірності тієї чи іншої послідовності, враховуючи синтаксичні, семантичні та прагматичні правила.

У контексті реалізації цих компонентів використовуються різноманітні алгоритмічні підходи. Нижче наведені базові алгоритми, що історично або актуально застосовуються в системах розпізнавання мовлення.

Приховані марковські моделі (Hidden Markov Models, HMM) - є одними з ключових статистичних моделей, що застосовуються в сучасних системах ASR, зокрема, в акустичному моделюванні. HMM являють собою генеративні моделі, здатні моделювати послідовності спостережень (наприклад, акустичних ознак), що походять від прихованих станів (наприклад, фонем або субфонемних одиниць). Їх ефективність у розпізнаванні мовлення пояснюється можливістю апроксимувати мовленнєвий сигнал як послідовність короткочасних (типово близько 10 мілісекунд), шматково-стаціонарних сегментів, трактуючи кожен сегмент як стаціонарний процес протягом цього короткого інтервалу. Важливими перевагами HMM є їхня здатність до автоматичного навчання на основі великих обсягів даних та відносна легкість імплементації [7].

Динамічне часове вирівнювання (Dynamic Time Warping, DTW) являє собою алгоритм, призначений для кількісної оцінки схожості між двома часовими рядами, які можуть мати відмінності у часі або швидкості протікання. DTW дозволяє знайти оптимальне "вирівнювання" або відповідність між точками двох послідовностей. Цей алгоритм історично застосовувалося у системах розпізнавання мовлення, особливо у системах розпізнавання ізольованих слів, однак у новітніх, більш гнучких архітектурах, що працюють з неперервним мовленням, він був значною мірою витіснений більш успішними підходами на основі HMM [7].

Нейронні мережі (Neural Networks, NN) є класом обчислювальних моделей, архітектура яких імітує структуру та функціонування біологічних нейронних мереж. В контексті ASR, нейронні мережі використовуються, зокрема, для моделювання або апроксимації складних нелінійних функцій, які встановлюють відповідність між вхідними акустичними ознаками та

ймовірностями фонем або інших лінгвістичних одиниць. Їх застосування особливо ефективне для обробки високорозмірних вхідних даних [7].

Глибокі нейронні мережі (Deep Neural Networks, DNN) та моделі глибокого навчання представляють собою підклас штучних нейронних мереж (Artificial Neural Networks, ANN) з багатошаровою архітектурою, що включає численні приховані шари між вхідним та вихідним рівнями. Моделі глибокого навчання (Deep Learning) охоплюють ширший спектр архітектур з численними рівнями обробки. Ця багатошаровість надає DNN та іншим моделям глибокого навчання здатність автоматично вивчати ієрархічні представлення даних та моделювати високоскладні, неочевидні нелінійні взаємозв'язки між вхідними акустичними даними та лінгвістичним змістом. Вони продемонстрували значне покращення точності в сучасних системах ASR.

2.5. Проектування системи

2.5.1. Діаграми потоку даних

Діаграма потоку даних (DFD) — це графічний метод представлення потоку даних у програмі. DFD представляє загальний огляд системи та аналіз структури. Цей метод дає детальний опис компонентів системи та також може надати загальний огляд системи на високому рівні. Раніше архітектури програмування використовували блок-схеми та псевдокод для представлення системи. DFD пояснює функції, які повинні бути виконані в програмі, та дані, які їм потрібні.

2.5.2. Діаграма випадків використання

Ця діаграма показує чотири найважливіші кроки в цьому додатку для розробки Java. У цьому короткому сценарії користувач отримає деякі підказки мовлення від додатку, ці підказки є результатом перевірки даних у

системі запису голосу в Android. Якщо система запису голосу ще не має результату, додаток почне надавати деякі підказки мовлення.

Наступний крок — коли користувач почне давати команди. Починаючи говорити, система запису голосу активується автоматично та почне отримувати вхідні дані з мікрофона.

Після отримання деяких результатів від перетворювача мовлення в текст (перетворювач мовлення в текст — це попередньо визначена та закодована програма Google Speech Recognition. Поточний додаток використовує бібліотеки та розширення для використання цього API) програма почне шукати відповідне слово у своїй бібліотеці Java та веб-пошуку. Якщо буде знайдено відповідність, ми побачимо отриманий результат на екрані.

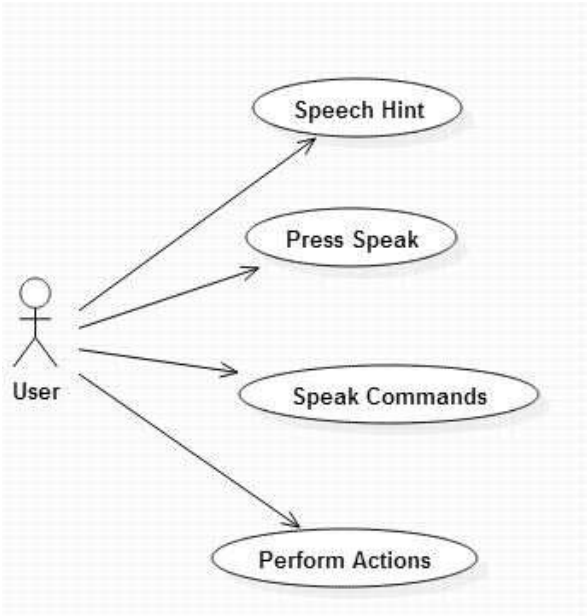

Рисунок 2.3 - Use Case діаграма

На даній діаграмі (рис. 2.3) представлені наступні елементи:

1. Актор (Actor). Зображений у вигляді чоловічка і позначений як "User" (Користувач). Актор представляє роль, яку зовнішня сутність (в даному випадку, людина-користувач) відіграє при взаємодії з системою.

2. Варіанти Використання (Use Cases). Зображені у вигляді овалів. Кожен овал представляє окрему функціональність системи, яка має значення для актора. На діаграмі представлені наступні варіанти використання:

- Speech Hint (Мовленнєва підказка): Функція системи, яка надає користувачеві підказки або інструкції у звуковому вигляді. Стрілка від "User" до цього овалу може означати, що користувач ініціює запит на підказку або система автоматично надає її у відповідь на дії користувача. Більш типово стрілка йшла б від системи до користувача, якщо це підказка від системи, але в контексті варіантів використання, це означає, що користувач отримує підказку як результат взаємодії. Однак, якщо розглядати стрілку від актора до варіанта використання, це частіше означає, що актор ініціює цей варіант використання. У даному випадку це є ініціація запиту на підказку.

- Press Speak (натиснути говорити) - представляє функцію активації режиму запису голосу користувача, яка ініціюється натисканням кнопки або іншого елемента інтерфейсу користувачем. Стрілка від "User" до цього овалу чітко вказує на ініціацію дії користувачем.

- Speak Commands (вимовляти команди) - цей варіант використання стосується функціональності системи, пов'язаної з розпізнаванням голосових команд, які вимовляє користувач. Стрілка від "User" вказує, що користувач надає голосові команди як вхідні дані.

- Perform Actions (виконувати дії) - представляє функціональність системи, яка полягає у виконанні певних дій у відповідь на розпізнані команди або іншу взаємодію з користувачем через мовлення. Стрілка від "User" до цього овалу означає, що дії системи є результатом взаємодії, ініційованої користувачем (наприклад, через вимову команд).

3. Зв'язки (Associations). Прямі лінії зі стрілками, що з'єднують актора з варіантами використання. Стрілка від актора до варіанта використання, як правило, вказує на те, що актор ініціює цей варіант використання.

Діаграма ілюструє основні способи взаємодії користувача з системою, яка, має функціонал, пов'язаний з обробкою мовлення. Користувач може ініціювати отримання мовленнєвої підказки, активувати режим запису мовлення, натиснувши "Говорити", вимовляти команди для системи, а система у відповідь буде виконувати певні дії. Діаграма показує ці функціональності як окремі варіанти використання, доступні для користувача.

2.5.3. Діаграма послідовності

Ця діаграма послідовності показує активність у класі MainActivity, де розміщений весь потік процесу. Для виконання команди додаток повинен завершити 6 кроків та перейти від одного методу до наступного.

Метод onCreate() — це крок, де створюється адаптер, і якщо адаптер був створений у попередньому циклі, він буде очищений. Наступний етап — коли додаток перевіряє, чи розпізнавач голосу був запущений та працює правильно.

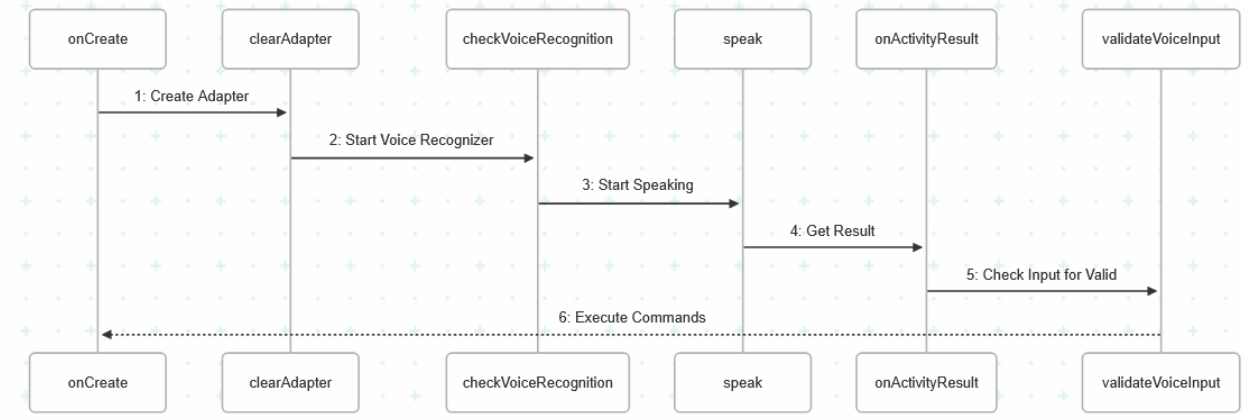

Рисунок 2.4 – Діаграма послідовності

На цій діаграмі (рис. 2.4) "Interaction MainActivity" зображено послідовність викликів методів (або функцій) та взаємодій, що відбуваються,

ймовірно, в межах головної активності Android-застосунку (MainActivity), пов'язаної з використанням функціоналу розпізнавання мовлення.

Елементи діаграми:

1. Назва Взаємодії (Interaction). Діаграма названа "Interaction MainActivity" моделює сценарій взаємодії, який відбувається в контексті класу MainActivity (типового для Android-розробки).

2. Лінії Життя (Lifelines). Вертикальні пунктирні лінії, що опускаються від заголовків, представляють лінії життя об'єктів. У даному випадку заголовки представляють методи або логічні блоки в межах MainActivity:

- onCreate: Метод, який викликається при створенні активності (стандартний життєвий цикл Android).

- clearAdapter: Метод або компонент для очищення адаптера (пов'язаного з відображенням результатів розпізнавання).

- checkVoiceRecognition: Метод або компонент для перевірки наявності та готовності функції розпізнавання голосу.

- speak: Метод або компонент, що ініціює процес "слухання" мовлення користувача.

- onActivityResult: Метод, який обробляє результат діяльності, що була запущена (стандартний життєвий цикл Android для отримання результату від іншої активності, наприклад, вбудованого інтерфейсу розпізнавання мовлення).

- validateVoiceInput: Метод або компонент для валідації (перевірки правильності або придатності) отриманого голосового вводу.

3. Повідомлення (Messages). Горизонтальні стрілки, що йдуть від однієї лінії життя до іншої, представляють повідомлення, що надсилаються (виклики методів, події). Повідомлення пронумеровані для відображення послідовності виконання. Прямокутники на лініях життя (execution specifications) показують період часу, протягом якого об'єкт виконує отримане повідомлення.

Діаграма ілюструє наступний сценарій:

1. Create Adapter. Взаємодія починається (в рамках методу onCreate або після нього), де ініціюється створення Адаптера (для списку результатів).

2. Start Voice Recognizer. Після створення адаптера викликається метод або виконується логіка для запуску (ініціалізації) механізму розпізнавання голосу (наприклад, викликається інтент для системного розпізнавача мовлення). Ця дія виходить з невідомого компонента і спрямована на компонент/метод checkVoiceRecognition. Це означає, що система ініціює перевірку готовності розпізнавача. Виходячи зі стрілки від невідомого до checkVoiceRecognition, що checkVoiceRecognition отримує повідомлення на перевірку стану розпізнавача, а не його запуск.

3. Start Speaking. Після перевірки (або в результаті успішної перевірки) викликається метод speak, який ініціює процес очікування голосового вводу від користувача. Це повідомлення від checkVoiceRecognition до speak.

4. Get Result. Після того, як користувач припиняє говорити і система розпізнавання обробляє мовлення, результат розпізнавання отримується. Це повідомлення надсилається до методу onActivityResult (системний інтерфейс розпізнавання мовлення повертає результат саме через цей метод активності). Стрілка йде до onActivityResult.

5. Check Input for Valid. Отриманий результат голосового вводу передається для валідації до методу validateVoiceInput. Стрілка від onActivityResult до validateVoiceInput.

6. Execute Commands. Якщо ввід визнано дійсним (за результатами валідації), виконуються відповідні команди. Це повідомлення надсилається від validateVoiceInput назад до початкового компонента (до головної логіки MainActivity або іншого компонента, який ініціював процес).

Діаграма послідовності демонструє потік управління в Android-додатку, який використовує функціонал розпізнавання мовлення. Вона починається з ініціалізації (створення адаптера), включає етапи перевірки та

запуску розпізнавання, очікування голосового вводу від користувача, отримання результату через системний callback (onActivityResult), валідацію отриманих даних та, нарешті, виконання команд на основі успішно розпізнаного та валідованого вводу. Діаграма наочно показує порядок виконання операцій та взаємодію між різними логічними частинами системи в рамках даного сценарію.

2.5.4. Діаграма активності

Діаграма активності — це проста діаграма, яка показує шлях користувача з моменту, коли він/вона запускає додаток, до моменту завершення повного успішного циклу. Цей шлях дає загальну картину того, як клас MainActivity організовує процес.

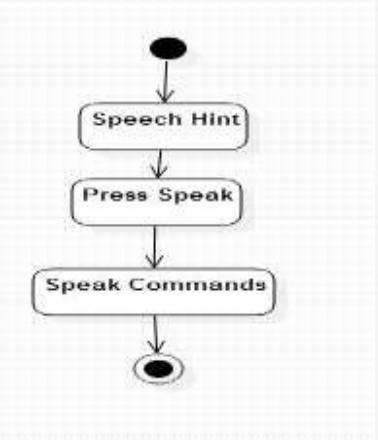

Рисунок 2.5 – Діаграма активності

На цій діаграмі (рис. 2.5) представлені наступні елементи:

1. Початковий вузол (Initial Node) - позначає початкову точку потоку активності.

- Вузли активності (Activity Nodes). Кожен вузол представляє окрему дію або крок у процесі. На діаграмі представлені наступні вузли активності:

- Speech Hint (мовленнєва підказка) - представляє дію надання або отримання мовленнєвої підказки.

- Press Speak (натиснути говорити) - представляє дію активації функції запису мовлення шляхом натискання кнопки.

- Speak Commands (вимовляти команди) - представляє дію вимовляння голосових команд користувачем.

3. Переходи (Transitions) вказують напрямок потоку управління від однієї дії до іншої.

4. Кінцевий вузол (Final Node) позначає кінцеву точку потоку активності.

Діаграма ілюструє послідовний потік дій:

1. Потік починається з Початкового Вузла.

2. Першою виконуваною дією є Speech Hint (мовленнєва підказка). Це може бути автоматичне надання підказки або дія користувача з її отримання.

3. Після виконання дії "Speech Hint" потік переходить до дії Press Speak (Натиснути Говорити). Це дія, яку виконує користувач для активації мікрофона.

4. Після "Press Speak" потік переходить до дії Speak Commands (вимовляти команди). Це дія, коли користувач вимовляє голосові команди.

5. Після виконання дії "Speak Commands" потік завершується у Кінцевому Вузлі.

Ця діаграма активності моделює сценарій взаємодії користувача з системою, що використовує розпізнавання мовлення. Вона показує послідовність дій: спочатку надається підказка, потім користувач активує функцію запису мовлення, вимовляє команди і процес завершується.

2.6. Проектування діаграми класів

Діаграма класів показує кількість класів, що існують у цьому додатку, та їхні методи. Вона також показує зв'язки між різними класами в різних пакетах.

MainActivity та JavaConstants — це два основні класи в цьому додатку, де ми зберігаємо константи програмування Java (для вгадування слів) та повний процес потоку додатку.

MainActivity містить основні змінні, методи та об'єкти. Він використовує дані в класі JavaConstants для перевірки правильності отриманих даних від Voice Recorder.

Database Helper — це інший клас в іншому пакеті, який підключається до різних бібліотек та передає результат до класу JavaConstants.

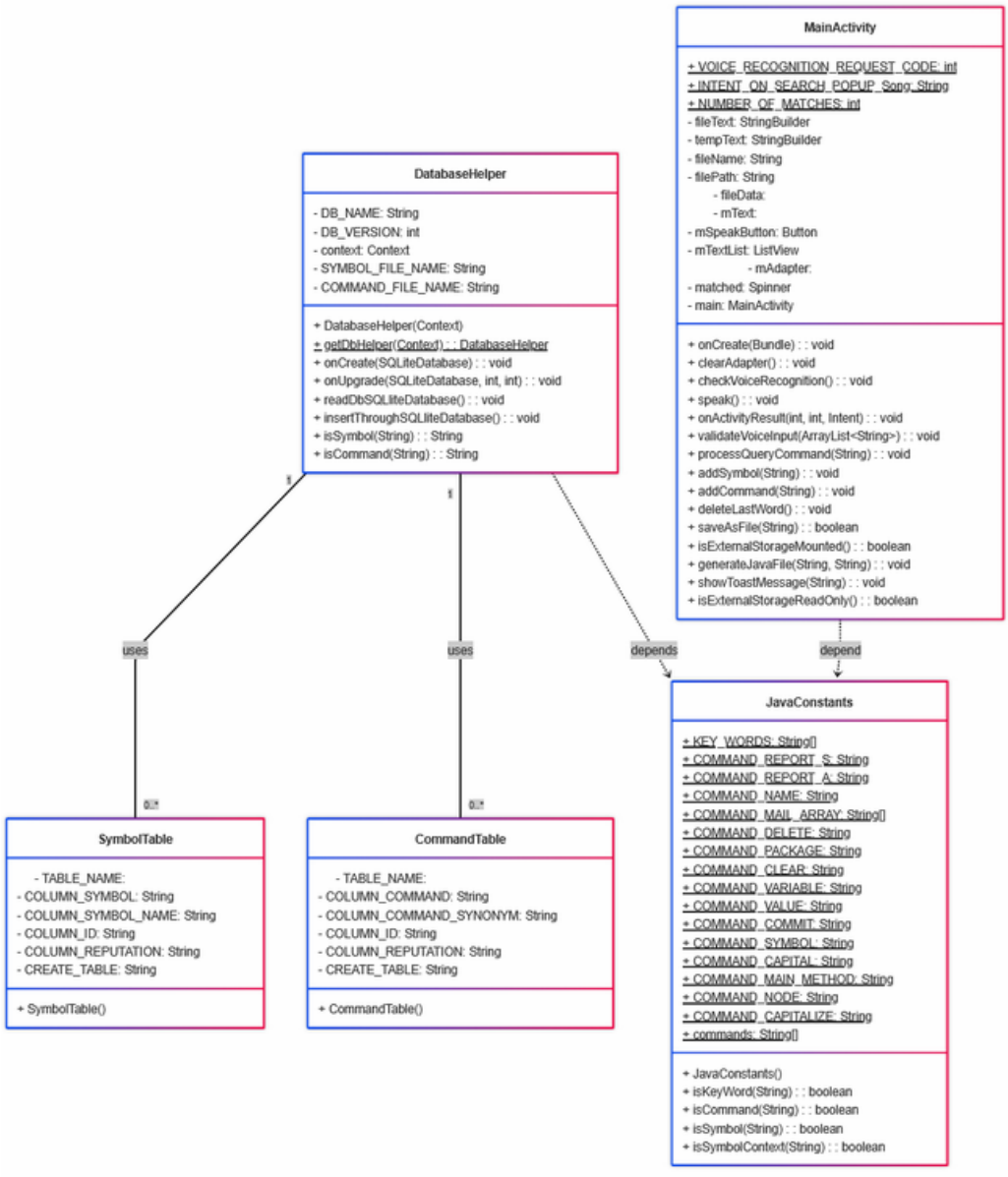

Рисунок 2.6 – Діаграма класів

Діаграма представляє частину архітектури Android-додатку, що використовує базу даних SQLite та функціонал розпізнавання мовлення.

Діаграма включає наступні класи:

1. DatabaseHelper (Пакет: com.example.voicerecognitiondb)

- Цей клас є допоміжним класом для роботи з базою даних SQLite в Android (успадковується від SQLiteOpenHelper).

- Атрибути:

- DB_NAME: String (Приватний, назва бази даних)

- DB_VERSION: int (Приватний, версія бази даних)

- context: Context (Приватний, контекст застосунку)

- SYMBOL_FILE_NAME: String (Приватний, назва файлу символів)

- COMMAND_FILE_NAME: String (Приватний, назва файлу команд)

- Методи:

+ DatabaseHelper(Context) (Публічний, конструктор)

+ getDbHelper(Context): DatabaseHelper (Публічний, статичний, для отримання синглтон-екземпляра)

+ onCreate(SQLiteDatabase): void (Публічний, викликається при створенні бази даних)

+ onUpgrade(SQLiteDatabase, int, int): void (Публічний, викликається при оновленні схеми бази даних)

+ readDbSQLliteDatabase(): void (Публічний для читання даних з бази)

+ insertThroughSQLliteDatabase(): void (Публічний для вставки даних у базу)

+ isSymbol(String): String (Публічний, перевіряє, чи є рядок символом)

+ isCommand(String): String (Публічний, перевіряє, чи є рядок командою)

2. SymbolTable (Пакет: com.example.voicerecognitiondb)

Цей клас представляє структуру таблиці для зберігання символів у базі даних.

- Атрибути:

- TABLE_NAME: String (Приватний, назва таблиці символів)

- COLUMN_SYMBOL: String (Приватний, назва стовпця для символу)

- COLUMN_SYMBOL_NAME: String (Приватний, назва стовпця для імені символу)

- COLUMN_ID: String (Приватний, назва стовпця для ідентифікатора)

- COLUMN_REPUTATION: String (Приватний, назва стовпця для репутації/ваги символу)

- CREATE_TABLE: String (Приватний, SQL-запит для створення таблиці)

- Методи:

+ SymbolTable() (Публічний, конструктор)

3. CommandTable (Пакет: com.example.voicerecognitiondb)

- Цей клас представляє структуру таблиці для зберігання команд у базі даних.

- Атрибути:

- TABLE_NAME: String (Приватний, назва таблиці команд)

- COLUMN_COMMAND: String (Приватний, назва стовпця для команди)

- COLUMN_COMMAND_SYNONYM: String (Приватний, назва стовпця для синоніма команди)

- COLUMN_ID: String (Приватний, назва стовпця для ідентифікатора)

- COLUMN_REPUTATION: String (Приватний, назва стовпця для репутації/ваги команди)

- CREATE_TABLE: String (Приватний, SQL-запит для створення таблиці)

- Методи:

+ CommandTable() (Публічний, конструктор)

4. JavaConstants (Пакет: com.example.voicerecognitiondb)

Цей клас містить набір констант, що використовуються в додатку, зокрема, пов'язані з командами та ключовими словами.

- Атрибути: (Всі публічні статичні константи типу String або масиви String[])

+ KEY_WORDS: String[]

+ COMMAND_REPORT_S: String

+ COMMAND_REPORT_A: String

+ COMMAND_NAME: String

+ COMMAND_MAIL_ARRAY: String[]

+ COMMAND_DELETE: String

+ COMMAND_PACKAGE: String

+ COMMAND_CLEAR: String

+ COMMAND_VARIABLE: String

+ COMMAND_VALUE: String

+ COMMAND_COMMIT: String

+ COMMAND_SYMBOL: String

+ COMMAND_CAPITAL: String

+ COMMAND_MAIN_METHOD: String

+ COMMAND_NODE: String

+ COMMAND_CAPITALIZE: String

+ commands: String[]

- Методи:

+ JavaConstants() (Публічний, конструктор)

+ isKeyWord(String): boolean (Публічний, перевіряє, чи є рядок ключовим словом)

+ isCommand(String): boolean (Публічний, перевіряє, чи є рядок командою)

+ isSymbol(String): boolean (Публічний, перевіряє, чи є рядок символом)

+ isSymbolContext(String): boolean (Публічний, перевіряє контекст символу)

5. MainActivity (Пакет: com.example.voicerecognitiondb)

Цей клас представляє головну активність Android-додатку, яка відповідає за користувацький інтерфейс, взаємодію з розпізнаванням мовлення та обробку вводу.

- Атрибути:

+ VOICE_RECOGNITION_REQUEST_CODE: int (Публічний, код запиту для розпізнавання мовлення)

+ INTENT_ON_SEARCH_POPUP_Song: String (Публічний, константа інтенту)

+ NUMBER_OF_MATCHES: int (Публічний, кількість знайдених відповідностей)

- fileText: StringBuilder (Приватний, для зберігання тексту файлу)

- tempText: StringBuilder (Приватний, тимчасовий текст)

- fileName: String (Приватний, назва файлу)

- filePath: String (Приватний, шлях до файлу)

- fileData: TextView (Приватний, елемент UI для відображення даних файлу)

- mText: TextView (Приватний, елемент UI для тексту)

- mSpeakButton: Button (Приватний, кнопка "Говорити")

- mTextList: ListView (Приватний, список для відображення тексту)

- mAdapter: ArrayAdapter<String> (Приватний, адаптер для списку)

- matched: Spinner (Приватний, випадаючий список для відповідностей)

- main: MainActivity (Приватний, посилання на екземпляр активності)

- Методи:

+ onCreate(Bundle): void (Публічний, метод життєвого циклу активності)

+ clearAdapter(): void (Публічний, очищення адаптера)

+ checkVoiceRecognition(): void (Публічний, перевірка можливості розпізнавання голосу)

+ speak(): void (Публічний, ініціація процесу розпізнавання мовлення)

+ onActivityResult(int, int, Intent): void (Публічний, обробка результату інтенту)

+ validateVoiceInput(ArrayList<String>): void (Публічний, валідація голосового вводу)

+ processQueryCommand(String): void (Публічний, обробка розпізнаної команди)

+ addSymbol(String): void (Публічний, додавання символу)

+ addCommand(String): void (Публічний, додавання команди)

+ deleteLastWord(): void (Публічний, видалення останнього слова)

+ saveAsFile(String): boolean (Публічний, збереження як файлу)

+ isExternalStorageMounted(): boolean (Публічний, перевірка монтування зовнішнього сховища)

+ generateJavaFile(String, String): void (Публічний, генерація Java-файлу)

+ showToastMessage(String): void (Публічний, показ Toast-повідомлення)

+ isExternalStorageReadOnly(): boolean (Публічний, перевірка зовнішнього сховища на читання)

Розглянемо зв'язки між класами.

MainActivity залежить від JavaConstants. На діаграмі є пунктирна стрілка від MainActivity до JavaConstants. Це зв'язок залежності (Dependency), що вказує на те, що клас MainActivity використовує функціональність або константи, визначені в класі JavaConstants (наприклад, методи isKeyWord, isCommand або самі константи команд/ключових слів).

DatabaseHelper асоційований з SymbolTable та CommandTable. Від класу DatabaseHelper йдуть стрілки до SymbolTable та CommandTable. Це зв'язки асоціації (Association), що вказують на те, що DatabaseHelper взаємодіє з концепціями, представленими цими таблицями. Зв'язок між DatabaseHelper та SymbolTable має множинність "1" з боку DatabaseHelper та "0.." (нуль або багато) з боку SymbolTable, що може означати, що один DatabaseHelper може працювати з багатьма записами в SymbolTable. Аналогічно, зв'язок між DatabaseHelper та CommandTable також має множинність "1" до "0..", що вказує на взаємодію з багатьма записами команд.

DatabaseHelper залежить від JavaConstants. Є пунктирна стрілка від DatabaseHelper до JavaConstants, що вказує на залежність. Це означає, що DatabaseHelper використовує константи або методи з JavaConstants, ймовірно, для визначення назв таблиць, стовпців або інших параметрів бази даних, які зберігаються як константи.

Отже, діаграма класів демонструє модульну структуру додатку. Класи SymbolTable та CommandTable визначають схему даних для зберігання символів та команд. DatabaseHelper надає функціональність для керування базою даних (створення, оновлення, читання, запис), використовуючи схеми, визначені в класах таблиць, та константи з JavaConstants. JavaConstants централізує визначення констант, що використовуються в інших частинах програми. MainActivity є основним інтерфейсом користувача, який ініціює процеси розпізнавання мовлення, обробляє отриманий текст (можливо, використовуючи логіку, яка взаємодіє з базою даних через DatabaseHelper та перевіряє ввід за допомогою JavaConstants), та виконує відповідні дії.

Ця діаграма ілюструє, як різні компоненти системи взаємодіють для реалізації функціоналу, що включає розпізнавання мовлення, обробку команд/символів та роботу з локальною базою даних.

Висновки до другого розділу

У другому розділі детально проаналізовано еволюцію мобільної операційної системи Android та її можливості у сфері автоматичного розпізнавання мовлення. Представлено принципи побудови та імплементації систем ASR, зокрема Google Voice Recognition, що є базовим інструментом для інтеграції голосового введення в Android-додатки. Розглянуто взаємодію клієнтської частини додатка з серверною частиною мовленнєвого сервісу, а також проаналізовано бібліотеки, що можуть бути застосовані для розпізнавання мовлення. Окрема увага приділена алгоритмічним основам функціонування таких систем. Запропоновано архітектурне рішення з використанням UML-діаграм (DFD, Use Case, Sequence, Activity, Class), що ілюструють логіку обробки голосових команд у мобільному середовищі. Отримані результати створюють основу для технічної реалізації голосового інтерфейсу у процесі розробки.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ІНТЕГРАЦІЯ ГОЛОСОВИХ КОМАНД В ПРОЦЕС АНДРОЇД JAVA-РОЗРОБКИ

3.1. Аналіз компонентів структура проекту Android із функціоналом інтеграції розпізнавання мовлення

Типова структура директорій Android-проекту, що інтегрує функціональність розпізнавання мовлення, включає низку ключових компонентів, необхідних для компіляції, розгортання та виконання застосунку. Основна директорія проекту, яку можна умовно назвати "Код Java з голосом" (або відповідно до назви проекту), містить стандартизований набір піддиректорій, що організовують вихідний код, ресурси, бібліотеки та згенеровані файли. До цих піддиректорій зазвичай належать:

- Src - містить вихідний код програми, написаний мовою Java або Kotlin.

- Gen - директорія для автоматично згенерованих файлів, таких як R.java (файл ресурсів) та файли, створені інструментами AIDL (Android Interface Definition Language).

- Assets - призначена для зберігання довільних файлів, які можуть бути доступні застосунком у режимі читання (наприклад, файли конфігурації, бази даних).

- Bin - містить згенеровані бінарні файли, включаючи скомпільований код та упакований APK-файл.

- Res - містить ресурси застосунку, такі як макети інтерфейсу користувача (layouts), зображення (drawables), рядки тексту (values/strings), анімації (anim) тощо.

- Libs - призначена для розміщення сторонніх бібліотек у форматі JAR-файлів.

Окрім цих стандартних директорій, структура проекту включає посилання на бібліотеки, критично важливі для розробки та забезпечення сумісності.

Бібліотека Android (Android SDK). Цей компонент представляє собою набір програмних інтерфейсів (API) та інструментів, що надаються платформою Android для розробників. Він є невід'ємною частиною середовища розробки Android (Android SDK) і необхідний для компіляції застосунку під конкретну версію платформи. Бібліотека містить скомпільований код фреймворку Android, доступний у вигляді JAR-файлів, що дозволяє розробникам використовувати стандартні класи та методи Android API.

У контексті застосунків, що використовують функціонал розпізнавання мовлення, особливе значення має пакет android.speech. Цей пакет надає класи та інтерфейси для взаємодії зі службою розпізнавання мовлення, що надається операційною системою Android. Він містить константи (наприклад, CONFIDENCE_SCORES, ERROR_AUDIO, ERROR_CLIENT), статичні методи (наприклад, для отримання екземпляра SpeechRecognizer) та інші елементи, успадковані від базових класів Java (java.lang.Object). Використання API з пакету android.speech дозволяє застосунку ініціювати процес розпізнавання, отримувати результати та обробляти помилки.

Бібліотеки підтримки Android (нині значною мірою замінені на AndroidX) є набором бібліотек, що надають зворотну сумісність для нових функцій фреймворку Android на старіших версіях операційної системи, а також включають додаткові компоненти UI та утиліти. Вони поставляються у вигляді JAR-файлів або AAR-файлів (Android Archive) і додаються як залежності проекту.

Залежності Android (Android Dependencies). Цей термін у контексті середовищ розробки та систем збірки (наприклад, Gradle в Android Studio) позначає механізм управління зовнішніми бібліотеками, від яких залежить

проект. Це віртуальне представлення, яке автоматично вирішує залежності, завантажує необхідні JAR/AAR файли та управляє потенційними конфліктами версій між різними бібліотеками. Система збірки ідентифікує кожну залежність за унікальним ідентифікатором (група, артефакт, версія) і визначає оптимальну версію для використання, якщо одна й та сама бібліотека потрібна різним компонентам проекту.

Приватна Бібліотека Android (Private Android Library) цей термін відноситься до специфічних бібліотек підтримки, які безпосередньо додані до проекту або використовуються ним. Прикладами є android-support-v7-appcompat.jar та android-support-v4.jar. Ці бібліотеки забезпечують сумісність застосунку з більш ранніми версіями Android, ніж та, що використовувалася для компіляції (цільова версія SDK). Бібліотека android-support-v4 надає сумісність з Android 2.0 (API рівня 5) та вище, тоді як android-support-v7-appcompat забезпечує сумісність з Android 3.0 (API рівня 11) та вище, надаючи, зокрема, компоненти ActionBar та Material Design.

Важливо розрізняти концепції "Залежності Android" та "Приватна Бібліотека Android" у контексті інструментів розробки (наприклад, Eclipse або старіші версії Android Studio). "Приватні бібліотеки" можуть відображати безпосередньо додані JAR-файли бібліотек підтримки, тоді як "Залежності" представляють більш високорівневий механізм управління залежностями, який автоматично визначає та включає необхідні бібліотеки (включаючи відповідні версії appcompat та support-v4) на основі конфігурації проекту.

Обидва ці представлення є віртуальними в тому сенсі, що вони управляються середовищем розробки/системою збірки і не завжди відповідають фізичним директоріям у файловій системі проекту. Включення відповідних бібліотек підтримки є критично важливим для забезпечення широкої сумісності застосунку з різними версіями операційної системи Android.

3.2. Реалізація основних методів

Директорія src є стандартним компонентом структури Android-проекту і призначена для розміщення вихідного коду застосунку, розробленого програмістом. Ієрархія цієї директорії містить основні класи, що реалізують логіку та інтерфейс користувача програми. У розглянутому проекті, що інтегрує функціональність розпізнавання мовлення, ключовими класами, розташованими в src, є JavaConstants.java та MainActivity.java.

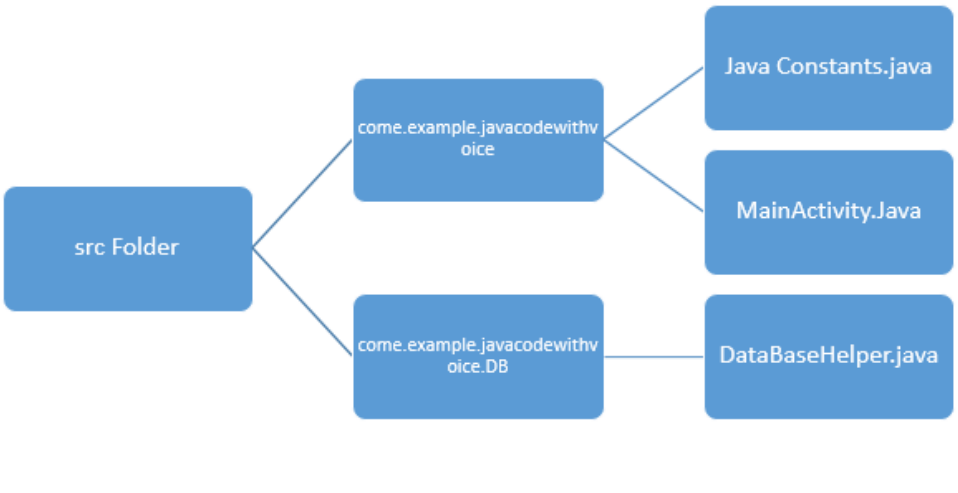

Рисунок 3.1 – Структура директорії src

Клас JavaConstants слугує репозиторієм для визначення програмних констант, які використовуються в межах застосунку. Окрім стандартних ключових слів мови програмування Java (таких як abstract, assert, Boolean, break тощо), цей клас містить константи та структури даних, специфічні для обробки результатів розпізнавання мовлення. Зокрема, він визначає відповідності між очікуваними голосовими командами або термінами та їхніми потенційними варіантами розпізнавання, які можуть виникати через акустичні варіації, акцент користувача, умови запису або неточності алгоритмів розпізнавання.

Наприклад, для команди "main" визначено константу COMMAND_MAIN та масив COMMAND_MAIN_ARRAY, що містить очікуваний варіант ("main") та типові помилкові розпізнавання ("maine", "mean", "meen", "mein")

```
public static final String COMMAND_MAIN = "main";
public static final String COMMAND_MAIN_ARRAY[] = { "main", "maine", "mean", "meen", "mein" };
```

Такий підхід дозволяє системі ідентифікувати команду "main" навіть за наявності незначних відхилень у розпізнаному тексті.

Клас JavaConstants також містить допоміжні методи, такі як isKeyword, isSymbol та isCommand. Ці методи взаємодіють з базою даних (через DatabaseHelper) для перевірки, чи належить розпізнане слово до множини визначених ключових слів, символів або команд, і повертають відповідний ідентифікатор або об'єкт до основного обробника (наприклад, у MainActivity). Приклад методу isCommand:

```
public static String isCommand(Context context, String commandI) {
    DatabaseHelper dbHelper = DatabaseHelper.getDBHelper(context);
    String command = dbHelper.isCommand(commandI);
    dbHelper.close();
    dbHelper = null;
    return command;
}
```

Цей метод отримує контекст застосунку та рядок (commandI), використовує DatabaseHelper для запиту до бази даних і повертає стандартизоване представлення команди, якщо вона знайдена.

Клас MainActivity є основним компонентом застосунку, що успадковується від базового класу Android Activity. Він відповідає за ініціалізацію користувацького інтерфейсу, управління життєвим циклом

активності та координацію взаємодії між різними частинами програми, включаючи функціонал розпізнавання мовлення.

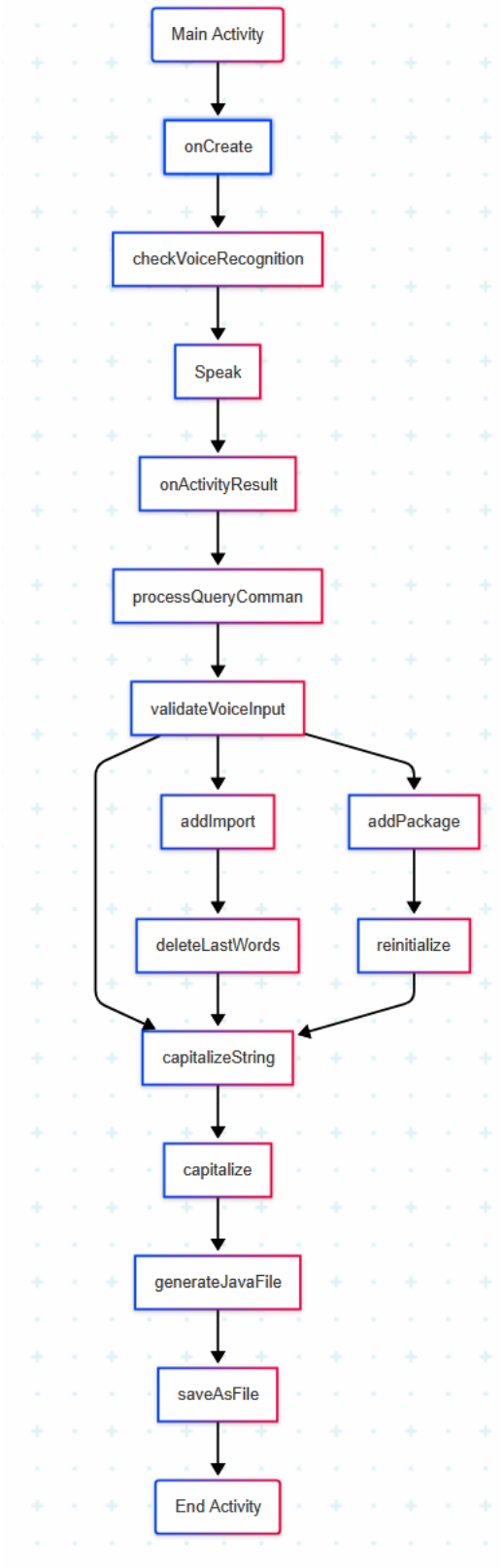

Рисунок 3.2 - Методи класу MainActivity

Ключові методи класу MainActivity включають:

- onCreate(Bundle) - цей метод є частиною життєвого циклу активності та викликається при її створенні. Він використовується для ініціалізації компонентів інтерфейсу користувача, встановлення початкового стану застосунку та відновлення попереднього стану активності, якщо він був збережений у об'єкті Bundle.

- checkVoiceRecognition() - цей метод виконує перевірку наявності та доступності служби розпізнавання мовлення на поточному пристрої. Він використовує PackageManager (клас, що надає інформацію про встановлені пакети застосунків) для запиту активностей, здатних обробляти інтент RecognizerIntent.ACTION_RECOGNIZE_SPEECH. Якщо список знайдених активностей порожній (activities.size() == 0), це означає відсутність сумісного розпізнавача мовлення, і відповідні елементи інтерфейсу (наприклад, кнопка "Говорити") можуть бути деактивовані, а користувачеві відображається повідомлення про помилку.

```
public void checkVoiceRecognition() {
    PackageManager pm = getPackageManager();
    List<ResolveInfo> activities = pm.queryIntentActivities(new
    Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH), 0);
    if (activities.size() == 0) {
        mbtSpeak.setEnabled(false);
        fileDataTv.setText(filePackage.toString() + "\n"+ fileImports.toString() + "\n"
        + "Voice recognizer not present");
    }
}
```

- speak(View view) - цей метод ініціює процес розпізнавання мовлення шляхом запуску відповідної системної активності. Створюється інтент з дією RecognizerIntent.ACTION_RECOGNIZE_SPEECH. До інтенту додаються додаткові параметри (putExtra), такі як пакет застосунку, текстова підказка для користувача (EXTRA_PROMPT), мовна модель (EXTRA_LANGUAGE_MODEL, наприклад, LANGUAGE_MODEL_WEB_

SEARCH для коротких фраз або LANGUAGE_MODEL_FREE_FORM для вільного мовлення) та максимальна кількість результатів розпізнавання (EXTRA_MAX_RESULTS). Запуск активності здійснюється методом startActivityForResult, що дозволяє отримати результат розпізнавання у методі onActivityResult.

```
public void speak(View view) {
    System.out.println(getFilesDir().getPath());
    Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_CALLING_PACKAGE, getClass()
    .getPackage().getName());
    intent.putExtra
    (RecognizerIntent.EXTRA_PROMPT,HINT_ON_SEARCH_POPUP);
    intent.putExtra
    (RecognizerIntent.EXTRA_LANGUAGE_MODEL,RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra
    (RecognizerIntent.EXTRA_MAX_RESULTS,NUMBER_OF_MATCHES);
    startActivityForResult(intent, VOICE_RECOGNITION_REQUEST_CODE);
}
```

- onActivityResult(int requestCode, int resultCode, Intent data): Цей захищений метод викликається системою для обробки результату активності, запущеної за допомогою startActivityForResult. Він перевіряє requestCode (для ідентифікації запиту розпізнавання мовлення) та resultCode (для визначення статусу операції). Якщо resultCode дорівнює RESULT_OK, це свідчить про успішне розпізнавання, і отримані результати (список рядків EXTRA_RESULTS) передаються для подальшої обробки. У випадку, якщо перший результат розпізнавання містить слово "search", застосунок ініціює веб-пошук за рештою фрази. В іншому випадку, результати передаються методу validateVoiceInput для валідації та обробки як команди або символу. Метод також обробляє різні коди помилок розпізнавання (наприклад, RESULT_AUDIO_ERROR, RESULT_CLIENT_ERROR, RESULT_NETWORK_ERROR, RESULT_NO_MATCH, RESULT_SERVER_ERROR), відображаючи відповідні повідомлення користувачеві.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == VOICE_RECOGNITION_REQUEST_CODE)
    if (resultCode == RESULT_OK) {
        ArrayList<String> textMatchList =
        data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
        if (!textMatchList.isEmpty()) {
            if(textMatchList.get(0).contains("search")) {
                String searchQuery = textMatchList.get(0);
                searchQuery = searchQuery.replace("search", "");
                Intent search = new Intent(Intent.ACTION_WEB_SEARCH);
                search.putExtra(SearchManager.QUERY, searchQuery);
                startActivity(search);
            } else {
                validateVoiceInput(textMatchList);
            }
        }
    } else if (resultCode == RecognizerIntent.RESULT_AUDIO_ERROR) {
        showToastMessage("Audio Error");
    } else if (resultCode == RecognizerIntent.RESULT_CLIENT_ERROR) {
        showToastMessage("Client Error");
    } else if (resultCode == RecognizerIntent.RESULT_NETWORK_ERROR) {
        showToastMessage("Network Error");
    } else if (resultCode ==RecognizerIntent.RESULT_NO_MATCH) {
        showToastMessage("No Match");
    } else if (resultCode == RecognizerIntent.RESULT_SERVER_ERROR) {
        showToastMessage("Server Error");
    }
}
```

- validateVoiceInput(ArrayList<String> textMatchList) - цей метод приймає список потенційних результатів розпізнавання мовлення (textMatchList) і виконує їх аналіз для визначення відповідних програмних конструкцій (наприклад, назв класів, символів, команд).

Логіка цього методу обробляє елементи списку, ймовірно, порівнюючи їх з визначеними константами та правилами для генерації або модифікації тексту програми на основі голосового вводу.

Таким чином, директорія src містить основний програмний код застосунку, де клас JavaConstants визначає ключові терміни та їхні варіації для розпізнавання, а клас MainActivity управляє життєвим циклом активності, взаємодією з користувачем, ініціацією та обробкою результатів розпізнавання мовлення.

3.3. Опис виконання системного тестування програмного забезпечення

Тестування програмного забезпечення є критично важливим етапом життєвого циклу розробки програмного забезпечення (Software Development Life Cycle, SDLC), що забезпечує верифікацію та валідацію продукту з метою гарантування його якості, надійності та відповідності встановленим специфікаціям та вимогам. Тестування надає комплексну оцінку архітектури, дизайну та реалізації програмного забезпечення, інтегруючись як невід'ємна частина процесу програмної інженерії.

Розробка програмного забезпечення часто моделюється за допомогою ітеративних або спіральних моделей, де тестування інтегрується на різних етапах. Стратегічний підхід до тестування передбачає систематичне планування та виконання тестових випадків, що охоплюють різні аспекти функціональності та продуктивності системи. Цей процес, як правило, прогресує через кілька рівнів, починаючи з тестування окремих компонентів і завершуючи комплексною перевіркою всієї системи.

Тестування програмного забезпечення класифікується за рівнями, кожен з яких має специфічні цілі та область застосування:

- Юніт-тестування (Unit Testing) - рівень тестування є першим і фокусується на перевірці найменших ізольованих компонентів програмного забезпечення (юнітів), якими зазвичай є окремі методи або функції. Метою юніт-тестування є верифікація коректності логіки кожного юніта та його здатності обробляти всі можливі вхідні дані та генерувати очікувані вихідні результати. Ефективне юніт-тестування допомагає ідентифікувати та усунути дефекти на ранніх стадіях розробки, підвищуючи загальну стабільність кодової бази.

Модульне тестування (Module Testing) - рівень, іноді синонімічний або тісно пов'язаний з юніт-тестуванням, зосереджується на тестуванні окремих

модулів або груп пов'язаних юнітів. Модульне тестування перевіряє функціональність кожного модуля в ізоляції, переконуючись, що кожен компонент виконує свої визначені функції належним чином.

Інтеграційне тестування (Integration Testing). Після юніт- та модульного тестування проводиться інтеграційне тестування, яке перевіряє взаємодію між інтегрованими модулями або компонентами. Метою є виявлення дефектів, що виникають на межах між компонентами, таких як помилки інтерфейсів, передачі даних або протоколів взаємодії. Цей рівень тестування допомагає переконатися, що всі частини ПЗ коректно функціонують разом.

Системне тестування (System Testing) є комплексним тестуванням повністю інтегрованої системи. Воно перевіряє відповідність системи функціональним та нефункціональним вимогам, визначеним у специфікації. Системне тестування оцінює поведінку системи як єдиного цілого, включаючи її продуктивність, безпеку, надійність та сумісність. На цьому рівні не вимагається глибоке знання внутрішньої структури коду.

Тестування прийняття користувачем (User Acceptance Testing, UAT) - фінальний рівень тестування проводиться кінцевими користувачами або зацікавленими сторонами в реалістичному середовищі. UAT спрямоване на перевірку того, чи відповідає система бізнес-вимогам користувачів та чи є вона прийнятною для використання. Результати UAT є вирішальними для фінального схвалення продукту перед його розгортанням.

Для забезпечення якості та коректності функціонування розробленого застосунку, що інтегрує розпізнавання мовлення, було проведено тестування на всіх вищезазначених рівнях.

Юніт-тестування було виконано для верифікації логіки окремих методів та компонентів, зокрема, для перевірки коректності обробки вхідних даних, включаючи різні варіанти розпізнавання слів та спеціальних символів. Була перевірена плавність потоку управління між методами в межах основної активності (MainActivity).

Модульне тестування передбачало ізольовану перевірку функціональності кожного модуля, наприклад, модуля взаємодії з базою даних (DatabaseHelper) або модуля обробки результатів розпізнавання.

Інтеграційне тестування було спрямоване на перевірку коректної взаємодії між модулями, зокрема, між компонентами розпізнавання мовлення та логікою обробки команд/символів, а також взаємодії з базою даних. Були протестовані сценарії збою та реакція системи на них.

Системне тестування підтвердило відповідність повністю інтегрованого застосунку системним вимогам, включаючи його здатність коректно розпізнавати мовлення, обробляти команди та генерувати відповідний вивід. Була перевірена відповідність функціоналу очікуванням користувача щодо розпізнавання та генерації коду Java.

Тестування прийняття користувачем було проведено для оцінки зручності використання застосунку кінцевими користувачами та підтвердження відповідності реалізованого функціоналу їхнім потребам та вимогам.

Під час тестування були розглянуті різні тестові випадки, включаючи введення некоректних або нульових даних. Зокрема, було перевірено реакцію системи на спробу оголошення методу main двічі, і система коректно відхилила цю дію, відобразивши відповідне повідомлення про помилку користувачеві. За результатами всебічного тестування на всіх рівнях, дефектів у функціональності застосунку виявлено не було, що підтверджує його працездатність та відповідність вимогам.

3.4. Розробка інтерфейсу користувача

Функціональність даного програмного забезпечення була верифікована шляхом проведення багатоетапного тестування, що охоплювало юніт-тестування, модульне тестування, інтеграційне тестування, системне

тестування та тестування прийняття користувачем. За результатами проведених випробувань, дефектів виявлено не було, що свідчить про коректну роботу реалізованого функціоналу. Результати тестування задокументовані на наступних скріншотах.

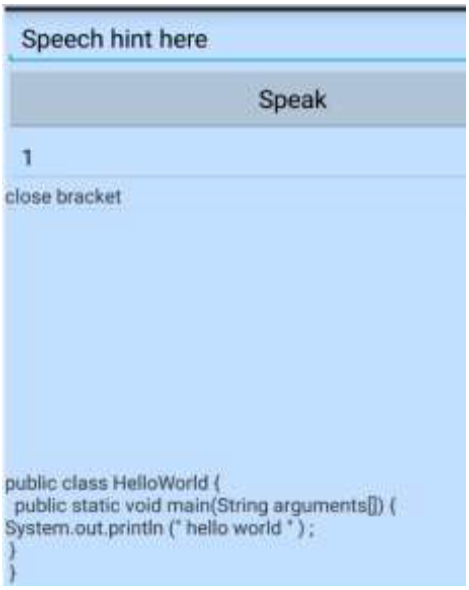

Рисунок 3.3 - Представлення спеціальних символів та базової функціональності.

Даний скріншот (рис. 3.3) ілюструє коректне розпізнавання та відображення спеціального символу "крапка з комою" (;) у текстовому виводі системи при голосовому вводі відповідної фрази. Додатково, демонструється приклад генерації програмного коду простої програми "Hello World" за допомогою функціоналу застосунку.

Скріншот (рис. 3.4) ілюструє синтаксис оголошення змінних цілочисельного типу та операцію присвоєння їм початкових значень у згенерованому коді.

На рисунку 3.5 демонструється структура та застосування циклу while для організації повторюваних обчислень у програмному коді, згенерованому системою.

Рисунок 3.4 - Оголошення та ініціалізація цілочисельних змінних

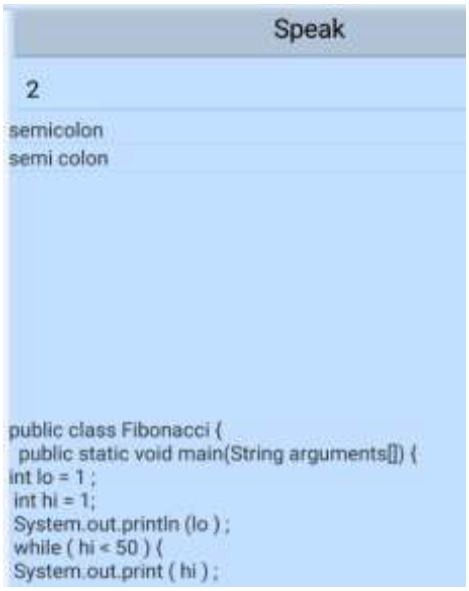

Рисунок 3.5 - Реалізація циклу while

На рисунку 3.6 ілюструються приклади використання різних типів операторів (арифметичних, логічних тощо) у програмному коді, отриманому як вивід системи.

На рисунку 3.7 представлено програмний код, згенерований застосунком, що реалізує обчислення послідовності Фібоначчі.

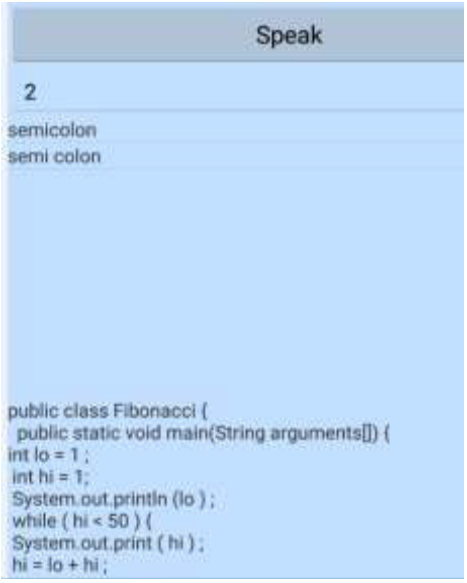

Рисунок 3.6 - Застосування операторів

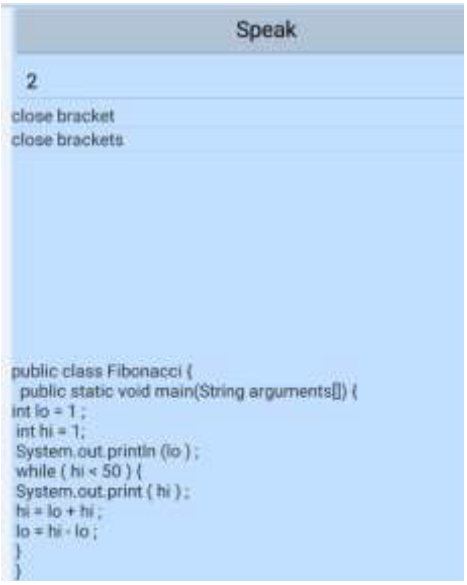

Рисунок 3.7 - Реалізація алгоритму Фібоначчі

На рисунку 3.8 демонструється синтаксис оголошення класу та приклади його застосування у згенерованому коді.

На рисунку 3.9 ілюструються додаткові приклади інтеграції спеціальних символів у програмний код, що демонструє можливості системи з їх коректного розпізнавання та вставки.

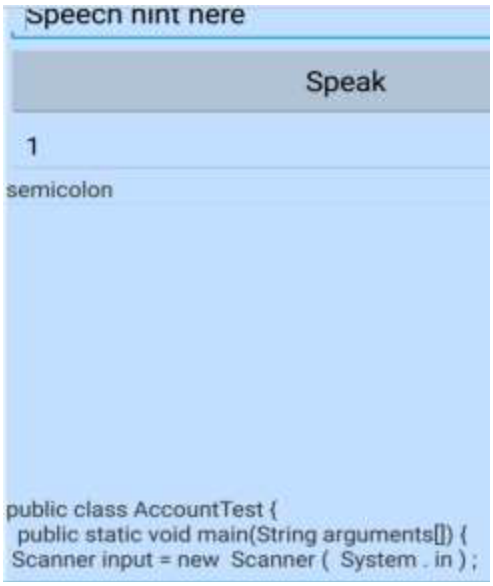

Рисунок 3.8 - Визначення та використання класу

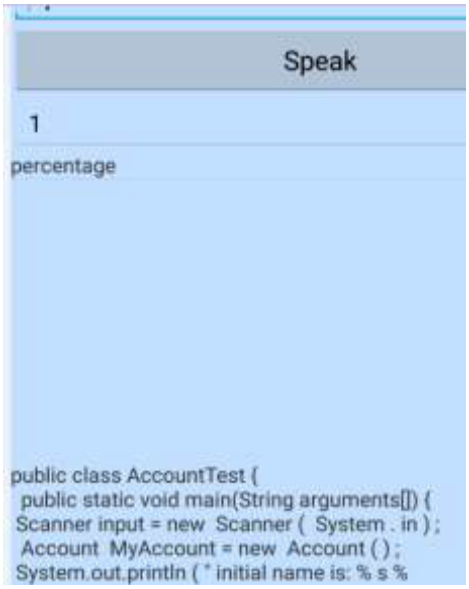

Рисунок 3.9 - Використання спеціальних символів

На рисунку 3.10 ілюструється процес інстанціювання об'єкта класу AccountTest та виклик його методу getName(). Демонструється комплексний приклад застосування даного класу в згенерованому коді.

На рисунку 3.11 представлено приклади оголошення змінних цілочисельного типу у згенерованому програмному коді, що є базовою операцією в програмуванні.

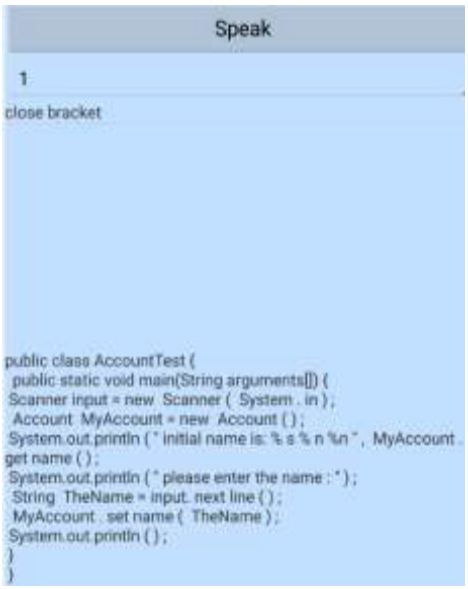

Рисунок 3.10 - Створення об'єктів та виклик методів

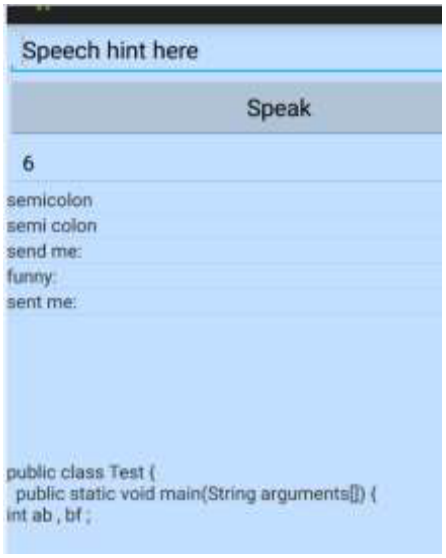

Рисунок 3.11 - Оголошення цілочисельних змінних

Подальший розвиток та оптимізація функціональних можливостей реалізованої системи інтеграції розпізнавання мовлення можуть бути реалізовані шляхом імплементації низки вдосконалень. Одним з потенційних напрямків є розширення програмної функціональності через інтеграцію додаткових компонентів. Це може включати:

- Інкапсуляцію зовнішніх бібліотек, тобто включення спеціалізованих сторонніх бібліотек, що надають розширені можливості обробки тексту,

синтаксичного аналізу, або взаємодії з іншими системами, може значно підвищити гнучкість та потужність застосунку.

- Інтеграцію внутрішніх модулів. Структурна реорганізація та інтеграція додаткових внутрішніх класів або модулів, що реалізують специфічну логіку (наприклад, для підтримки інших мов програмування, складніших синтаксичних конструкцій або кастомізованих команд), сприятиме масштабованості та розширюваності системи.

Іншим критично важливим аспектом для покращення є оптимізація механізму розпізнавання мовлення. Поточна імплементація покладається на службу Google Voice Recognition (GVR). Покращення в базовій технології GVR, зокрема, підвищення її точності та стійкості до варіацій мовлення (таких як акценти, діалекти, швидкість мовлення) та умов запису, безпосередньо позитивно впливатиме на продуктивність даного застосунку. Чутливість до акценту може бути значним бар'єром для користувачів-неносіїв англійської мови. Розширення діапазону акустичних моделей та вдосконалення алгоритмів GVR для кращого врахування цих варіацій є ключовим для підвищення інклюзивності та зручності використання системи.

Основне призначення розробленого застосунку полягає у функціонуванні як допоміжної технології для надавання користувачам можливість здійснювати розробку програмного забезпечення мовою Java шляхом голосового диктування. Система трансформує вимовлені користувачем програмні терміни та команди у відповідний текстовий код за допомогою механізму розпізнавання мовлення. Така функціональність може суттєво підвищити продуктивність та розширити можливості користувачів які прагнуть займатися програмуванням.

Реалізована система розроблена спеціально для функціонування на пристроях під управлінням операційної системи Android, використовуючи мову програмування Java як основний інструмент розробки.

Висновки до третього розділу

Третій розділ присвячено безпосередній реалізації розробленої системи голосового управління. Проаналізовано структуру Android-проєкту, що включає функціональність розпізнавання мовлення, реалізовано ключові методи обробки голосових команд та їх трансформації у програмні дії. Проведено системне тестування реалізованого програмного забезпечення, що підтвердило коректність роботи основних функцій. Описано підхід до створення інтерфейсу користувача, який забезпечує зручність, інтуїтивність та ефективність взаємодії з голосовими функціями. Таким чином, завершено повний цикл розробки — від теоретичної концепції до технічного впровадження та тестування, що підтверджує життєздатність запропонованого рішення.

ВИСНОВКИ

В дипломній роботі було здійснено всебічний аналіз теоретичних і практичних засад імплементації голосових команд і технологій розпізнавання мовлення в середовищі Android Java-розробки. Проведене дослідження охоплює широкий спектр питань — від обґрунтування передумов використання голосового введення до практичної реалізації функціональної системи, що дозволяє інтегрувати розпізнавання мовлення в програмні продукти мобільного спрямування.

У першому розділі було розглянуто технологічні й концептуальні аспекти розробки мобільних застосунків для платформи Android, що є однією з найбільш поширених і динамічних екосистем мобільного програмування. Детально проаналізовано існуючі системи розпізнавання мовлення, їх переваги й обмеження, а також визначено технічні вимоги до реалізації голосового інтерфейсу. Особливу увагу приділено проблематиці адаптації стандартних систем розпізнавання мовлення до задач генерації та керування програмним кодом, що потребує не лише точності, але й контекстної чутливості алгоритмів.

Другий розділ присвячено розгляду алгоритмічної основи роботи систем розпізнавання мовлення, включно з аналізом алгоритмів, що лежать в основі Google Voice Recognition (GVR), та способів їхньої інтеграції в Android-додатки. Деталізовано архітектурні та функціональні аспекти, що дозволяють забезпечити ефективну комунікацію між мобільним клієнтом і голосовим сервісом. Описано програмні бібліотеки та фреймворки, які надають інструменти для реалізації голосового введення, а також побудовано UML-діаграми, які формалізують структуру, логіку взаємодії та послідовність виконання основних сценаріїв.

У третьому розділі реалізовано практичну частину дослідження — створено прототип мобільного додатку з підтримкою голосових команд на

базі Android Java. Проведено розробку основної структури проєкту, реалізовано ключові методи обробки голосового введення, виконано інтеграцію з сервісами розпізнавання мовлення, зокрема GVR, та забезпечено базову візуалізацію користувацького інтерфейсу. Здійснено системне тестування програмного забезпечення, в результаті якого підтверджено відповідність реалізованого функціоналу заявленим вимогам та стабільність роботи додатку в типових умовах експлуатації.

Таким чином, результати дослідження підтверджують ефективність використання технологій розпізнавання мовлення в мобільній Java-розробці. Розроблений підхід може бути застосований у широкому колі задач, пов’язаних з підвищенням доступності, інклюзивності та інтуїтивності програмних продуктів. Запропонована архітектура є масштабованою і може бути адаптована для реалізації голосових інтерфейсів у складніших середовищах. Подальші дослідження доцільно спрямувати на удосконалення механізмів контекстної обробки мовлення, інтеграцію з інтелектуальними системами аналізу команд користувача, а також на дослідження можливостей офлайн-розпізнавання для покращення автономності програмних продуктів.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., ... & Ng, A. Y. (2014). Deep Speech: Scaling up end-to-end speech recognition. arXiv preprint arXiv:1412.5567.
2. Watanabe, S., Hori, T., Karita, S., Hayashi, T., Nishitoba, J., Unno, Y., ... & Ochiai, T. (2018). ESPnet: End-to-End Speech Processing Toolkit. arXiv preprint arXiv:1804.00015.
3. Pratap, V., Hannun, A., Xu, Q., Cai, J., Kahn, J., Synnaeve, G., ... & Collobert, R. (2018). wav2letter++: The Fastest Open-source Speech Recognition System. arXiv preprint arXiv:1812.07625.
4. Kim, C., Gowda, D., Lee, D., Kim, J., Kumar, A., Kim, S., ... & Han, C. (2020). A review of on-device fully neural end-to-end automatic speech recognition algorithms. arXiv preprint arXiv:2012.07974.
5. Lim, M. J., Lee, E. S., & Kwon, Y. M. (2012). Implementing Mobile Interface Based Voice Recognition System. In Computer Applications for Database, Education, and Ubiquitous Computing (pp. 189-195). Springer, Berlin, Heidelberg.
6. Ghita, C. (2020). Process voice input in Android using StanfordNLP. ProAndroidDev. Retrieved from https://proandroiddev.com/how-to-process-voice-input-in-android-with-stanfordnlp-84be2d6bb67e
7. Khare, A. (2021). Add Voice Commands to Android Apps. Medium. Retrieved from https://medium.com/geekculture/add-voice-commands-to-android-apps-80157c0d5bcc
8. CodingTechRoom. (n.d.). Integrating Speech Recognition with Android Apps using Java. Retrieved from https://codingtechroom.com/tutorial/java-integrating-speech-recognition-android-apps-java
9. CodeProject. (2013). Developing Android Applications with Voice Recognition Features. Retrieved from

https://www.codeproject.com/Articles/689451/Developing-Android-Applications-with-Voice-Recogni

10. Alan AI. (n.d.). Building a voice agent for an Android Java or Kotlin app. Retrieved from https://alan.app/docs/tutorials/android/integrating-java-kotlin/
11. Wired. (2013). How Google Retooled Android With Help From Your Brain. Retrieved from https://www.wired.com/2013/02/android-neural-network
12. Wired. (2009). Web Semantics: Sphinx4 speech recognition. Retrieved from https://www.wired.com/2009/07/web-semantics-sphinx4-speech-recognition
13. MDPI. (2021). Development of an Android-Based, Voice-Controlled Autonomous Robotic Vehicle. Proceedings, 58(1), 48.
14. Google Developers. (n.d.). SpeechRecognizer | Android Developers. Retrieved from https://developer.android.com/reference/android/speech/SpeechRecognizer
15. Google Cloud. (n.d.). Speech-to-Text Documentation. Retrieved from https://cloud.google.com/speech-to-text/docs
16. Mozilla. (n.d.). DeepSpeech. Retrieved from https://github.com/mozilla/DeepSpeech
17. CMU Sphinx. (n.d.). CMU Sphinx Open Source Speech Recognition. Retrieved from https://cmusphinx.github.io/
18. Kumar, A., & Han, C. (2020). On-device end-to-end speech recognition: Challenges and opportunities. IEEE Signal Processing Magazine, 37(6), 140-151.
19. Zhang, Y., Chen, J., & Yu, D. (2017). Speech recognition with deep recurrent neural networks. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 25(1), 1-12.
20. Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., ... & Zhu, Z. (2016). Deep Speech 2: End-to-End Speech

Recognition in English and Mandarin. In International Conference on Machine Learning (pp. 173-182).

21. Li, J., Deng, L., Gong, Y., & Haeb-Umbach, R. (2014). An overview of noise-robust automatic speech recognition. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 22(4), 745-777.
22. Yu, D., & Deng, L. (2014). Automatic Speech Recognition: A Deep Learning Approach. Springer.
23. Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. R., Jaitly, N., ... & Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition. IEEE Signal Processing Magazine, 29(6), 82-97.
24. Graves, A., Mohamed, A. R., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In IEEE International Conference on Acoustics, Speech and Signal Processing (pp. 6645-6649).
25. Chiu, C. C., Sainath, T. N., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., ... & Weiss, R. J. (2018). State-of-the-art speech recognition with sequence-to-sequence models. In IEEE International Conference on Acoustics, Speech and Signal Processing (pp. 4774-4778).
26. Zhang, Y., Wu, Y., Jaitly, N., & Le, Q. V. (2017). Very deep convolutional networks for end-to-end speech recognition. In IEEE International Conference on Acoustics, Speech and Signal Processing (pp. 4845-4849).
27. Chan, W., Jaitly, N., Le, Q. V., & Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In IEEE International Conference on Acoustics, Speech and Signal Processing (pp. 4960-4964).
28. Sainath, T. N., Weiss, R. J., Senior, A., Wilson, K. W., & Vinyals, O. (2015). Learning the speech front-end with raw waveform CLDNNs. In Sixteenth Annual Conference of the International Speech Communication Association.

29. Park, D. S., Chan, W., Zhang, Y., Chiu, C. C., Zoph, B., Cubuk, E. D., & Le, Q. V. (2019). SpecAugment: A simple data augmentation method for automatic speech recognition. In Interspeech (pp. 2613-2617).
30. Panayotov, V., Chen, G., Povey, D., & Khudanpur, S. (2015). Librispeech: An ASR corpus based on public domain audio books. In IEEE International Conference on Acoustics, Speech and Signal Processing (pp. 5206-5210).
31. Kudo, T. (2018). Subword regularization: Improving neural network translation models with multiple subword candidates. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (pp. 66-75).
32. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 770-778).
33. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems (pp. 5998-6008).
34. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “ Інтеграція голосових команд в процес Андроїд Java-розробки ”

Обсяг пояснювальної записки: 77 аркушів.

Дата закінчення роботи: 10 червня 2025 р.

Підпис студента _____