Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента	*Яковлєвої Владислави Олексіївни*
	(ПІБ)
академічної групи	*121М-21-1*
	(шифр)
спеціальності	*121 Інженерія програмного забезпечення*
	(код і назва спеціальності)
освітньої програми	*«Інженерія програмного забезпечення»*
	(назва освітньої програми)
на тему:	*Підвищення точності розпізнавання голосу на клієнтській стороні на підставі створення автономного механізму ASR*

В.О. Яковлєва

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний				
економічний				
Рецензент				
Нормоконтролер				

Дніпро
2022

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

(підпис) (прізвище, ініціали)

« » 20 22 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

студенту *121м-21-1* *Яковлєвій Владиславі Олексіївні*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Підвищення точності розпізнавання голосу на клієнтській стороні на підставі створення автономного механізму ASR*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від____.__.2022 р. № _____

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процеси автоматичного розпізнавання голосу.

Предмет досліджень – методи представлення голосу у вигляді тексту.

Методи дослідження: нейромережеві методи створення акустичної та мовної моделі, використання існуючих програмних пакетів для обробки мовного сигналу, розробка програмного забезпечення автоматичного розпізнавання голосу.

Мета роботи – розробити алгоритми розпізнавання голосу на основі створення україномовної моделі.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Новизна запропонованих рішень визначається тим, що вперше було створено мобільний застосунок з функцією розпізнавання голосу і перетворення в текст на основі створення україномовної моделі і використання архітектури DeepSpeech.

Практична цінність результатів полягає у тому, що в результаті проведеного дослідження було розроблено алгоритм для автоматичного розпізнавання голосу.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Розробка методів автоматичного збору транскрипцій українською мовою, які підходять для навчання нейронних мереж. Створення системи розпізнавання українською мовою на основі архітектури DeepSpeech за допомогою навчання рекуретної нейроної мережі на готовиї аудіо-сигналах та транскрипцій з використанням метода CTC.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз джерел та постановка задачі	08.10.2022 - 30.10.2022
Побудова математичних моделей та розробка алгоритму	01.11.2022 - 15.11.2022
Розробка та тестування програмного забезпечення для автономного механізму ASR для підвищення точності розпізнавання голосі на клієнтській стороні	15.11.2022 - 01.12.2022

Завдання видав _____ *Алексєєв М.О.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Яковлєва В.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *08.10.2022 р.*

Термін подання кваліфікаційної роботи до ЕК *20.01.2022*

РЕФЕРАТ

Пояснювальна записка:____стор.,____рис.,____таблиці,____додатка, ____ джерел.

Об’єкт досліджень – процеси автоматичного розпізнавання голосу.

Предмет досліджень – методи представлення голосу у вигляді тексту.

Методи дослідження: нейромережеві методи створення акустичної та мовної моделі, використання існуючих програмних пакетів для обробки мовного сигналу, розробка програмного забезпечення автоматичного розпізнавання голосу.

Мета роботи – розробити алгоритми розпізнавання голосу на основі створення україномовної моделі.

Новизна запропонованих рішень визначається тим, що вперше було створено мобільний застосунок з функцією розпізнавання голосу і перетворення в текст на основі створення україномовної моделі і використання архітектури DeepSpeech.

Практична цінність результатів полягає у тому, що в результаті проведеного дослідження було розроблено алгоритм для автоматичного розпізнавання голосу.

Розробка методів автоматичного збору транскрипцій українською мовою, які підходять для навчання нейронних мереж. Створення системи розпізнавання українською мовою на основі архітектури DeepSpeech за допомогою навчання рекуретної нейроної мережі на готовиї аудіо-сигналах та транскрипцій з використанням метода CTC.

Список ключових слів: АВТОМАТИЧНЕ РОЗПІЗНАВАННЯ ГОЛОСУ, ЯКІСТЬ РОЗПІЗНАВАННЯ, ОБРОБКА МОВНОГО СИГНАЛУ, CTC, АКУСТИЧНА МОДЕЛЬ, DEEPSPEECH.

ABSTRACT

Explanatory note: ___ pages, ___ figures, ___ tables, ___ appendices, sources.

The object of research is processes of automatic voice recognition.

The subject of research is the methods of presenting voice in the form of text.

Research methods: neural network methods of creating an acoustic and speech model, use of existing software packages for speech signal processing, development of automatic voice recognition software.

The aim of the work is to develop voice recognition algorithms based on the creation of a Ukrainian-language model.

The novelty of the proposed solutions is determined by the fact that for the first time, a mobile application with the function of voice recognition and conversion into text was created based on the creation of a Ukrainian-language model and the use of the DeepSpeech architecture.

The practical value of the results is that an algorithm for automatic voice recognition was developed as a result of the conducted research.

Development of methods of automatic collection of transcriptions in the Ukrainian language, which are suitable for training neural networks. Creation of a Ukrainian language recognition system based on the DeepSpeech architecture by training a recurrent neural network on ready-made audio signals and transcriptions using the CTC method.

List of key words: AUTOMATIC VOICE RECOGNITION, RECOGNITION QUALITY, SPEECH PROCESSING, CTC, ACOUSTIC MODEL, DEEPSPEECH.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ASR – automatic speech recognition;

IVR – interactive voice response;

MFCC – Mel-frequency cepstral coefficients;

RNN – recurrent neural networks;

LSTM - Long short-term memory;

WER - word error rate.

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ ТА АЛГОРИТМІВ АВТОМАТИЧНОГО РОПІЗНАВАННЯ ГОЛОСУ 10
1.1. Актуальність теми дослідження. 10
1.2. Алгоритми розпізнавання мови 13
1.3. Постановка задачі 20
РОЗДІЛ 2. РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ ДЛЯ РОЗПІЗНАВАННЯ МОВЛЕННЯ 21
2.1. Структура акустичної моделі 21
2.2. Налаштування RNN тренування 26
2.3. Мовна модель 28
2.4. Паралелізм даних 29
2.5. Синтез порівнянням сигналів 30
РОЗДІЛ 3.РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 32
3.1. Технології розробки програмного забезпечення 32
3.2. Збір даних для навчання акустичної моделі 36
ВИСНОВКИ 50
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 51
Додаток А. ЛІСТИНГ ПРОГРАМИ 55
Додаток Б. ВІДГУК КЕРІВНИКА 80
Додаток В. РЕЦЕНЗІЯ 87
Додаток Г. АКТ ВПРОВАДЖЕННЯ 87
Додаток Д. ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ 87

ВСТУП

Автоматичне розпізнавання мовлення – ціль останніх семи десятиліть. При перших комп'ютерах зв'язок між людиною і комп'ютером досягався механічними клавіатурами, а потім в допомогу прийшли і комп'ютерні мишки. Але останньою тенденцією, яка допомагає керувати комп'ютерними системами є розпізнавання мовлення і в подальшому перетворення цих команд в зрозумілу для комп'ютерних систем мову. Ця технологія стала широко розповсюджуватись майже у всіх сферах зв'язаних з інформаційними технологіями. Не так швидко крокує розвиток, але дослідники багато вклали в можливість інтегрування в наш побут і продовжують це робити в даний момент часу.

Ключовою проблемою залишається точність розпізнавання мовлення. Точність порушується багатьма факторами, які нас оточують нас і переслідують кожну хвилину. Людина адаптувалася до більшості цих факторів і ігнорує їх в процесі розпізнавання тексту іншої людини. Комп'ютерній системі важко на перших часах розділити корисну інформацію від залишкової, тому її треба навчити це розрізняти і так само як людина ігнорувати цю інформацію або інтерпретувати нові діалектні слова.

Серед основних факторів можна виділити шум, неоднотипне мовлення диктора, різні акценти і розставляння наголосів диктора, різні діалектичні слова або словосполучення, величина словникового запасу, продуктивність даної системи та багато інших. До шуму або акустичного шуму можна віднести будь-які коливання частинок довкілля, які не дають корисної інформації. Усі вони заважають коректній роботі системі розпізнавання мовлення, так само коли ви розмовляєте по телефону з другом, а попри вас проїжджає автомобіль і привносить в ваш діалог фізичне забруднення навколишнього середовища, таким чином також можна описати визначення шуму.

До неоднотипного мовлення диктора можна віднести складність розпізнавання при ситуаціях, які викликали в диктора різні емоційні збудження. Не тільки емоції можуть стати бар'єром для розпізнавання, але й швидкість

подачі інформації диктором комп’ютерній системі. Вони можуть, так мовити, «з’їдати» букви при вимовлені, а система отримає тільки інформацію у вигляді незрозумілих словоформ. Проблема розпізнавання акцентів і різного розставлення наголосів у словах є одною з важких завдань на даний момент для систем розпізнавання мовлення. Ця мовна варіативність породжує великі складнощі, які важко передбачити або спробувати навчити і адаптуватися. Різниця і проблема в акцентах в технічному розумінні полягає в тоні диктора, його наголосів та довжині, які важко передбачити та змоделювати. Не можна виключати такий параметр як схожість деяких видів акцентів та відмінності на фонологічному рівні.

Так, вирішення цієї проблеми розглядається у створенні кросплатформеної бібліотеки для розпізнавання голосу на базі двигуна DeepSpeech з попередньо навченою мовною моделлю.

Кваліфікаційна робота складається з трьох розділів. Перший розділ присвячено загальному опису потреби автономного механізму ASR. У другому розділі описано алгоритмізація розпізнавання голосу на основі побудови мовної моделі. У третьому розділі представлено обґрунтування вибору використаних технологій, характерні особливості, архітектура розробленого програмного забезпечення, аналіз якості роботи систем та аналіз отриманих результатів.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ ТА АЛГОРИТМІВ АВТОМАТИЧНОГО РОЗПІНАВАННЯ ГОЛОСУ

1.1. Актуальність теми дослідження.

Розпізнавання мовлення – це технологія, яка дозволяє комп'ютеру чи програмі ідентифікувати окремі слова або фрази, сказані людиною, та перетворити їх у текст. Вона включає в себе знання та дослідження у галузях інформатики, лінгвістики та електротехніки.

На сьогоднішній день суспільство вносить велику кількість коштів на розвиток науково-дослідних розробок для вирішення проблем автоматичного розпізнавання і розуміння мови. Це стимулюється практичними потребами, пов'язаними із створенням систем комерційного призначення.

Проблема розпізнавання мови вважається надзвичайно серйозною і відіграє велику роль у спілкуванні людини з машиною. Управління об'єктами за допомогою мови може відкрити великі перспективи перед автоматизацією у багатьох людської діяльності, відкрило б можливість спілкування з машинами, особливо користувачам персональних комп'ютерів, які не володіють базовими навичками програмування. Мовний контакт полегшує запис даних у машину, допомагає людині і комп'ютеру працювати в реальному масштабі часу.

Системи розпізнавання мовлення поступово займають місце посередником між людиною та девайсом, і таким чином створюють альтернативу звичним засобам обміну інформацією. Поряд з програмним забезпеченням для диктування на персональним комп'ютерах, розвиваються більш прогресивні системи, такі як голосові асистенти, які, окрім виконання команд, імітують живий діалог та розв'язують прикладні задачі. Ці системи можуть бути реалізовані, як програма на комп'ютері чи смартфоні, так і як окремий девайс.

Комерційні програми або застосунки з розпізнавання мови з'явились ще на початку дев'яностих років. Найчастіше вони використовуються серед людей, у

яких є травми або захворювання рук, через що вони не можуть друкувати великий об’єм тексту. Такі програми (наприклад, Dragon Naturally Speaking, VoiceNavigator) допомагають користувачу не навантажувати руки, перекладаючи голос у текстовий вигляд.

У комп’ютерах Apple Macintosh є вбудована функція для аналізу команд користувача при натисканні певної кнопки, або якщо система розпізнала ключове слово або фразу для включення функції Speech.

Ще однією популярною програмою є Speereo Voice Translator, відомий як голосовий перекладач. Ця програма здатна розпізнавати слова або фрази англійською мовою і «відповідати» перекладом на обрану користувачем мову. Мовні прикладні програми, які можуть автоматично розпізнавати усну мову, є наступним кроком у розвитку інтерактивних голосових систем IVR.

Використання інтерактивного телефонного програмного забезпечення в сучасному світі є більше необхідністю, ніж розвагою. З деяких плюсів можна назвати зниження навантаження на операторів контакт-центрів, скорочення витрат на оплату праці, підвищення продуктивності багатьох систем обслуговування і збільшення швидкості обробки інформації.

Проте, прогрес не стоїть на місці і все частіше у телефонних застосунках з’являються системи автоматичного розпізнавання голосу та синтезу мовлення. Це використовується для спрощення спілкування користувачем з системою, бо голосові команди є більш природніми і зрозумілими для людини. При цьому системи розпізнавання є незалежними від тону, вони розпізнають голос будь-якої людини. Необхідність у розвитку голосових систем є в основній перевазі для звичайного користувача – в такому випадку можливо повністю відмовитися від вивчення налаштувань і складних лабіринтів різних меню, застосовуючи для цього голосові команди. Тепер, наприклад, можна просто вимовити бажаний пункт застосунку, після чого програма самостійно відкриє бажану сторінку додатка.

З кожним днем необхідність використання автоматичного розпізнавання мовлення на ринку зростає. Застосування цієї технології знаходить своє місце в

галузі охорони здоров’я, комерційних проєктах, будівельній та технологічній сферах. Разом з цим потреба в розвитку розпізнавання мови як технології є підставою для розробки кращих засобів ASR.

На одному рівні з необхідністю в розробці механізму автоматичного розпізнавання голосу, випливають багато недоліків, які значно впливають на досвід використання ASR. Наприклад, Siri – віртуальний голосовий помічник – може використовуватися тільки на пристроях від компанії Apple. Окрім цього, з 2016 року 46% користувачів продукції подали скарги на точність розпізнавання голосу і перетворення у текст – з них близько 20% люди з вадами мови.

З цим можна виділити основні недоліки існуючих ASR сервісів:

1) Точність.

Для більшості проектів перетворення мови в текст точність є найважливішою змінною. Звичайне програмне забезпечення для розпізнавання мовлення не завжди дає бажані результати. Перевірки вручну є дорогим та тривалим. Альтернативою є розширена транскрипція, яка забезпечує точність, коли базове програмне забезпечення для перетворення тексту в мовлення цього не робить. Професійна транскрипція справляється з надмірним фоновим шумом або діалогами з акцентами. Транскрипція за допомогою людини настільки ефективна, що деякі постачальними пропонують 99-відсоткову гарантію точності.

2) Гнучкість.

Спеціалізовані служби пропонують більш повний перелік послуг. Розширені платформи перетворення мовлення в текст можуть пропонувати програми та безкоштовні інструменти, які можна використовувати в своїх проектах. Проте проблемою таких програм є не широкий перелік можливого застосування – проект необхідно налаштувати під конкретну функцію.

3) Інтеграції.

Звичайне програмне забезпечення для розпізнавання голосу не завжди інтегрується з усіма операційними системами.

4) Залежність від інтернету.

Незважаючи на те, зо використання хмарних технологій для технології розпізнавання мови стає більш популярним, в нього є недолік у вигляді неможливості використання функції перетворення голосу в текст при від'єднанні від мережі.

1.2. Алгоритми розпізнавання мови

Вперше для розпізнавання мови використовували алгоритм динамічної трансформації часової шкали (DTW-алгоритм), який дозволяв знайти оптимальну відповідність між часовими послідовностями. Приймаючи два звукових сигналів, які представляли собою одну й ту саму фразу, за допомогою деформації осі двух послідовностей, знаходились спільні форми в сигналах. Незважаючи на те, що такий метод пошуку слів в звукових сигналах використовувався досить довго, він мав суттєві недоліки у вигляді неправильних результатів через невірну деформацію одного сигналу до другого, а також часто виникають випадки, коли вирівнювання в принципі неможливе через знаходження плато першого ряду набагато нижче або вище плато нижчого ряду.

На заміну прийшли методи дискримінантного аналізу, побудовані на Байесовській дискримінації, а також приховані марківські моделі (HMM). Традиційна архітектура систем автоматичного розпізнавання мови мала велику кількість недоліків, через які було необхідно шукати інші методи перетворення голосу в текст без втрати точності. Тоді за допомогою використання нейронних мереж стався прорив у технології, який дозволив більше використовувати потенціал методу ASR.

Поєднання рекуретної нейронної мережі і прихованої марківської моделі створили новий, так названий, гібридний підхід, який відмітився найбільш ефективним з усіх запропонованих алгоритмів. Проте гібридна архітектура успадковувала всі недоліки марківської моделі, такі як заміна оригінальних слів на найбільш популярні в акустичній моделі.

Пізніше були створені end-to-end ASR, які повністю створювались на нейронних мережах і завдяки алгоритму навчання передбачали текст, отримуючи

на вхід набір акустичних ознак певного висловлювання. Саме такі моделі використовуються останнім часом.

Таким чином, можна сказати, що нейронні мережі витіснили всі інші алгоритми і методи в розробці, і стали невід'ємною частиною ASR.

End-to-end ASR – це система, яка безпосередньо відображає послідовність вхідних акустичних характеристик у послідовність слів. Система , яка навчена оптимізувати критеріях, пов'язана з фінальною метрикою оцінки (тобто частотою помилок у словах). Для звичайної архітектури ASR, більшість систем включають окремо навчену акустику, вимови та компоненти мовної моделі, які навчаються окремо. Опрацювання вимовної лексики, визначення наборів фонем для конкретної мови вимагає експертних знань і займає багато часу на навчання.

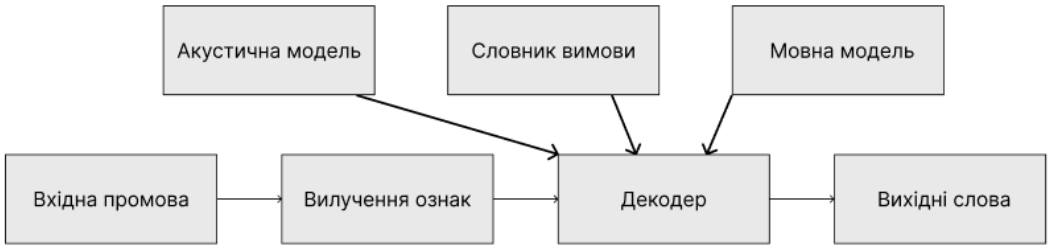

Рис. 1.1 Традиційна архітектура ASR

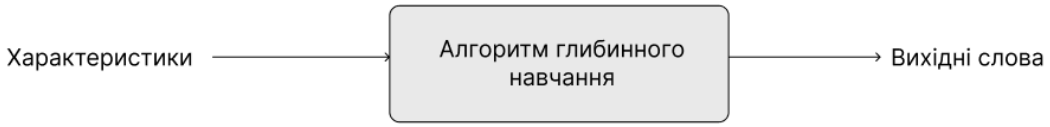

Рис. 1.2 End-to-end архітектура ASR

Можна побачити, що end-to-end архітектура значно простіша за традиційний метод розпізнавання мови. Немає необхідності вручну виставляти точки інформації, в нейронній мережі можливо автоматично визначати мову або інформацію про вимову, як показано на малюнку (1.2). Існує всього дві основні структури для end-to-end ASR: attention model та connectionist temporal classification (CTC).

Connectionist temporal classification модель приймає акустичні ознаки на вхід і навчається передбачати символи для кожного фрейму. Символи зазвичай представляють собою літери або звуки. Потім алгоритм проходить по всіх можливих послідовностях символів, які підходять до транскрипції, та обирає найбільш імовірну.

Алгоритм, що використовує виходи RNN для розпізнавання мови отримав назву connectionist temporal classification (CTC). А процес незалежного маркування послідовності нейронною мережею на кожному часовому кроці, або фреймі, вхідної послідовності називається пофреймова класифікація (framewise classification).

Важливий крок для забезпечення сумісного використання RNN та CTC складається у перетворенні виходів нейронної мережі у розподіл умовної ймовірності послідовностей маркерів. Таким чином, виходи нейронної мережі стають вхідними даними для CTC алгоритму. Отримана CTC мережа є класифікатором, що дозволяє обирати найбільш ймовірне маркування для поданої вхідної послідовності.

При огляді існуючих алгоритмів, необхідно розглянути вже існуючі рішення для розпізавання голосу. Одним із найвідоміших сервісів сьогодення для розпізнавання мовлення є Google Cloud Speech Api. Основною сферою застосування цього рішення є перетворення мови з аудіо в текст, але також є інструменти для проведення діаризації. Розпізнавання відбувається віддалено на серверах компанії Google, тому для роботи програми потрібне підключення до Інтернету.

Крім того, це рішення використовує закриті розробки компанії, що накладає додаткові обмеження можливості його використання. При цьому команда розробників регулярно ділиться своїми основними досягненнями на різних конференціях, а також реалізацією окремих методів та алгоритмів.

Доступ до API надається через web-інтерфейс за допомогою HTTP запитів або через бібліотеки для мов Java, Python, JavaScript (Node.JS).

Компанія IBM пропонує своє end-to-end рішення для розпізнавання мови в текст, у тому числі з використанням діаризації дикторів на аудіозаписі. Доступ до сервісу, що використовує потужності суперкомп'ютера IBM Watson, здійснюється за допомогою web-інтерфейсу.

Неважко помітити, що це рішення багато в чому схоже на аналогічне API від компанії Google і виявляє самі недоліки: неможливість автономної роботи алгоритму і закритий вихідний код. Незважаючи на високу репутацію обох компаній, до подібних реалізацій завжди є претензії щодо захищеності даних користувача.

Одна з найбільш продуктивних бібліотек для аналізу мови, анонсована наприкінці 2019 року та опублікована у вільному доступі. На даний момент практично немає документації, що значно ускладнює роботу з даним продуктом. Для Linux можна використовувати бібліотеку Python, продуктивності якої достатньо для роботи на платах Raspberry Pi. Бібліотека працює на вдосконаленому движку Kaldi, який раніше вважався одним з кращих рішень для мовного аналізу. Мовна модель займає лише 50Мб і працює досить точно (з моделями розміром більше 1Гб). Підтримуються мови: російська, англійська, німецька, французька, китайська. Очікується підтримка іспанської, хінді, арабської та португальської).

CMUS Sphinx поставляється з групою розширених функцій з кількома попередньо створеними пакетами, що належать до розпізнавання мови. Це програма з відкритим кодом, розроблений в Університеті Карнегі-Меллона. Ви отримаєте цей незалежний від розмови інструмент розпізнавання кількома мовами, включаючи французьку, англійську, німецьку, голландську та інші. Примітні особливості CMU Sphinx:

1) постачається з гнучким дизайном та ефективною системою навіть на платформах з низьким рівнем ресурсів;
2) надає інструменти для навчання акустичної моделі пакет.
3) допомагає виконувати різні типи завдань за допомогою своїх корисних

пакетів, включаючи визначення ключових слів, оцінку вимови, вирівнювання та багато іншого.

4) Це кросплатформовий інструмент, що підтримує як Windows, так і Linux.

WavLetter ++ – це сучасний та популярний інструмент розпізнавання мови, розроблений дослідницькою групою Facebook AI Research. Це ще одна програма з відкритим кодом під ліцензією BCD. Це надшвидке програмне забезпечення для розпізнавання голосу було створено C ++ і містить безліч функцій. Він надає своїм користувачам можливість моделювання мов, машинного перекладу, синтезу мови та багато іншого в гнучкому середовищі.

Він містить активну спільноту на популярних платформах, таких як Facebook і Google, щоб допомогти своїм користувачам по всьому світу. WavLetter ++ - це швидкий і гнучкий інструментарій, який для максимальної ефективності використовує тензорну бібліотеку ArrayFire. як wav2letter ++, який допомагає проводити успішні дослідження та налаштування моделі.

Julius - це порівняно старіша програма для розпізнавання голосу з відкритим вихідним кодом, розроблена Лі Акінобу. Цей інструмент написаний мовою програмування з розробниками Kawahara Lab, Університет Кіото. Це високопродуктивний додаток для розпізнавання мовлення з великим словниковим запасом. Ви можете використовувати його як англійською, так і японською мовами. Це може бути відмінним вибором, якщо ви хочете використовувати його в академічних та дослідних цілях.

Julius - це програма з широкими можливостями налаштування, яка може встановлювати різні параметри пошуку для налаштування своєї продуктивності. Це кросплатформовий проект, що працює в системах Linux, BSD, Windows.

Саймон поставляється із сучасним та простим у використанні програмним забезпеченням для розпізнавання мови, розробленим Пітером Грашем. Це ще одна програма з відкритим вихідним кодом під Стандартною громадською ліцензією GNU. Ви можете використовувати Simon як у Linux, так і Windows.

Крім того, він забезпечує гнучкість для роботи з будь-якою мовою.

Використовуючи свій калькулятор з голосовим керуванням, Саймон надає можливість виконувати різні арифметичні операції. Сумісний зі Skype та іншими популярними програмами VOIP встановити легкий система зв'язку з друзями та родичами. Це дозволяє користувачам дивитися слайд-шоу та відео, слухає музику і багато іншого за допомогою декількох простих голосових команд. Крім того, це важливий інструмент для читання газет та роботи в Інтернеті.

Mycroft поставляється із простим у використанні голосовим помічником з відкритим вихідним кодом для перетворення голосу на текст. Він вважається одним із найпопулярніших інструментів розпізнавання мови Linux у наш час, написаним на Python. Це дозволяє користувачам якнайкраще використовувати цей інструмент у науковому проекті або корпоративному програмному додатку. Крім того, його можна використовувати як практичний помічник, який може сказати вам час, дату, погоду та багато іншого.

Інтегрований з найпопулярнішими соціальними мережами та професійними платформами, включаючи Facebook, Github, LinkedIn та ін. Крім того, що він є розумним голосовим помічником, він надає можливість запису звуку, машинного навчання, бібліотеки програмного забезпечення та багато іншого. Він дозволяє користувачам перетворювати природну мову на машиночитані дані за допомогою Adapt, аналізатора намірів Mycroft.

Open Mind Speech - один із основних інструментів розпізнавання мови Linux, призначений для безкоштовного перетворення вашої мови на текст. Він є частиною Open Mind Initiative, керує своєю роботою, особливо розробників. Ця програма була представлена під різними іменами, такими як VoiceControl, SpeechInput та FreeSpeech, до того, як отримала справжнє ім'я.

Speech Control - це безкоштовний додаток для розпізнавання мови, який підходить для будь-якого дистрибутива Ubuntu. Він поставляється з графічним інтерфейсом користувача на основі Qt. Хоча вона все ще знаходиться на ранній стадії розробки, ви можете використовувати його для свого простого проекту.

Speech Control - це програма з відкритим вихідним кодом під Стандартною громадською ліцензією (GPL). Він націлений на роботу як віртуальний помічник, який надає повторювані інструкції з виконання завдання для плавного виконання процесу.

1.3. Постановка задачі

Задача роботи полягає у дослідженні методів обробки вхідного голосового сигналу для створення кросплатформеної бібліотеки з найвищим показником точності на клієнтській стороні. Після чого необхідно створити мобільний застосунок з функціоналом перетворення мови, для чого треба вирішити наступні завдання:

1) провести аналіз предметної області, здійснити огляд існуючих підходів до покращення якості розпізнавання мовного сигналу;

2) розробити мовну модель для розпізнавання української мови на клієнтській стороні;

3) розробити та реалізувати алгоритм покращення якості розпізнавання мови;

4) провести аналіз існуючих двигунів задля забезпечення автономного розпізнавання голосу;

5) розробити бібліотеку для використання на різних платформах;

6) створити мобільний застосунок для зручної взаємодії користувача з системою.

РОЗДІЛ 2
РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ ДЛЯ РОЗПІЗНАВАННЯ МОВЛЕННЯ

2.1. Структура акустичної моделі

Для початку необхідно визначити структуру акустичної моделі, передбачаючої послідовність символів алфавіту для вхідних аудіо-сигналів.

Введемо множину символів (файл алфавіту) A, довжини [A], який складається з букв української мови та пробілу.

Послідовність довжини T векторів x складаєтсья з векторів, кожен з яких отримано вилученням з аудіо-сигналу короткої довжини (зазвичай 30 мілісекунд) мел-часткового кепстру (MFCCs). Ознаки, що виділяються MFCC чисельно описують відрізок аудіо і являються набором даних, на основі яких передбачатимуться символи.

При вилученні алгоритмом MFCC можна задати кількість ознак – m. Тоді створюється рекурентна нейрона мережа з m входам та n виходами і з вектором ω, що переводить вхідну послідовність x в послідовність векторів довжини n.

$$\mathrm{N}_{\omega} : (\mathrm{R}^{\mathrm{m}})^{\mathrm{T}} \rightarrow (\mathrm{B}^{\mathrm{n}})^{\mathrm{T}} \qquad (2.1)$$

Нейронна мережа складається з 5 шарів. Вхідний шар має m входів, що відповідають m MFCC частотним ознакам, які вибираються з фрагменту аудіо. Далі йдуть подібні два шари. У перших 3 шарах нейрони між сусідними шарами з'єднані всіма можливими способами і використовують функцію активації ReLU (Rectifier Linear Unit). Четвертий шар є двонаправленим рекурентним шаром (Bidirectional Recurrent layer) з LSTM ячейками і тангенсом як функція активації. Далі результат подається до п'ятого шару з ReLU активацією. Після цього вихідний шар розміром n = [A'], де значення на виході кожного з нейронів пропорційне ймовірності відповідної літери алфавіту. Функція активації ReLU виглядає наступним чином:

$$\mathrm{Re}LU(x) = \max(0, \mathrm{x}) \tag{2.2}$$

До всіх п’яти шарів використовується dropout з ймовірністю виключення 0.3. Для навчання нейронної мережі використовується пара (аудіо, транскрипт). Весь набір даних розділяється на 3 категорії: набір для тренування (train), набір для перевірки (dev) та набір для тестування (test).

Набори train, dev та test зазвичай отримуються з вихідного набору даних, завдяки розподілу на пропорції 60:20:20 відповідно. Набір для тренування використовується для коригування векторів нейроної мережі в процесі навчання, набір для перевірки використовується, відповідно, для перевірки точності нейроної мережі в процесі навчання на інших даних, аніж набір для тренування, тому що нейронна мережа може перенавчитися та адаптуватися саме з прикладами з тренувального набору. Тестовий набір використовується для фінальної перевірки в кінці навчання, на цьому етапі розраховується середнє значення CTC Loss, та середнє значення WER по всім прикладам для виявлення точності навченої моделі на рівні слів.

Функція CTC Loss була створена у 2006 році і активно використовується для навчання RNN. Кожен елемент початкової послідовності y = $N\omega$(x) є вектором розподілення вероґідностей по кожному з символів алфавіту A' в конкретний час t. Тому елемент $\mathrm{y}_{\mathrm{k}}^{\mathrm{t}}$ – це вирогідність, що в момент часу на початковій послідовності був використаний символ k з алфавіту.

Якщо розглядати послідовності довжини T, де кожен елемент є номером з [1, n], який відповідає символу в алфавіті, то кожну таку послідовність можна перетворити на послідовність символів, використовуючи заміну номера на відповідний символ. Вирогідність кожної такох послідовності при існуючому x можна знайти по формулі:

$$\mathrm{p}(\pi|\mathrm{x}) = \prod_{\mathrm{t}=1}^{\mathrm{T}} \mathrm{y}_{\pi_{\mathrm{t}}}^{\mathrm{t}} \ , \forall \pi \in A'^{T} \tag{2.3}$$

При розпізнаванні голосу очікується, що кожній вимовленій літері буде відповідати один символ у вихідній послідовності, а проміжки між словами будуть відображатися як один знак пробілу. Але так як в аудіо деякі звуки, що відповідають одному символу, можуть розтягуватися на період часу більший, аніж 30 мілісекунд, в матриці одному символу, який очікується у вихідній послідовності, може відповідати кілька стовпців. Тому, для того, аби отримати фінальну строку тексту, необхідно перетворити послідовність.

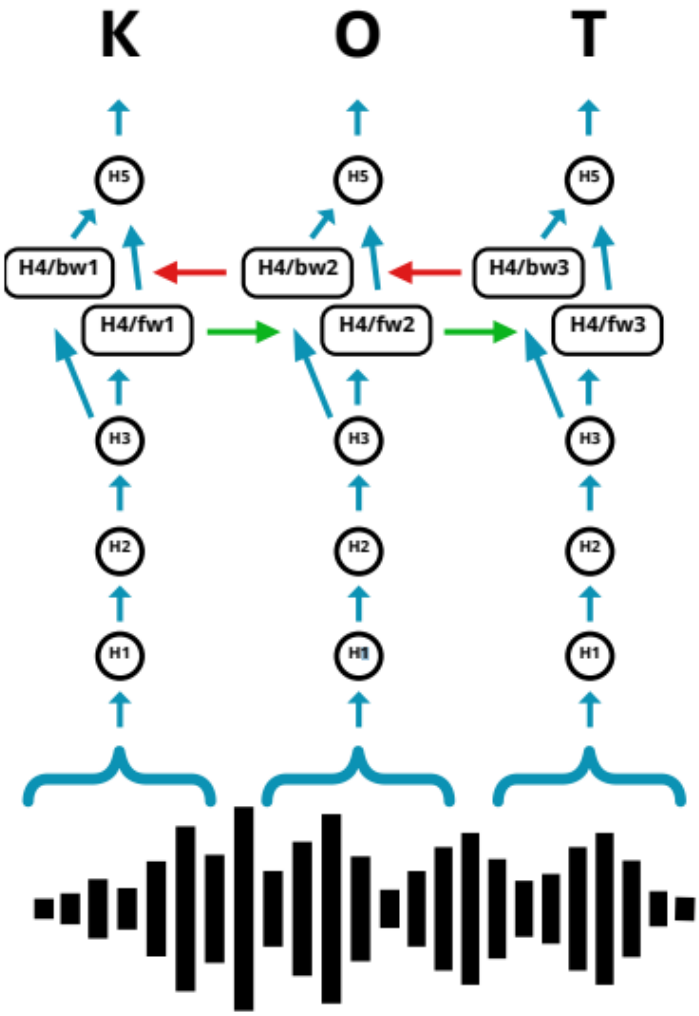

Рис. 2.1. Шлях даних через нейронну мережу від аудіо сигналів

Для навчання нейронної мережі за допомогою метода виявлення помилки,

необхідна сама функція для розрахунку послідовностей x, нейронної мережі, і очікуваного результату. Розрахувати таку функцію помилки можна як мінус натуральний логарифм від вирогідності очікуваного результату:

$$\mathrm{L_{CTC}} = -\ln\big(p(\mathrm{l}|\mathrm{x})\big) \tag{2.4}$$

Для отримання вирогідності результата необхідно скласти вирогідності всіх послідовностей. Таким чином можна отримати функцію $\mathrm{L_{CTC}}$:

$$\frac{\partial L_{CTC}}{\partial y_k^t} = \frac{1}{\alpha_T([l'] - 1) + \alpha_T([l'])} \frac{1}{{y_k^t}^2} \sum_{s:l'_s=k} \alpha_t(s)\beta_t(s) \tag{2.5}$$

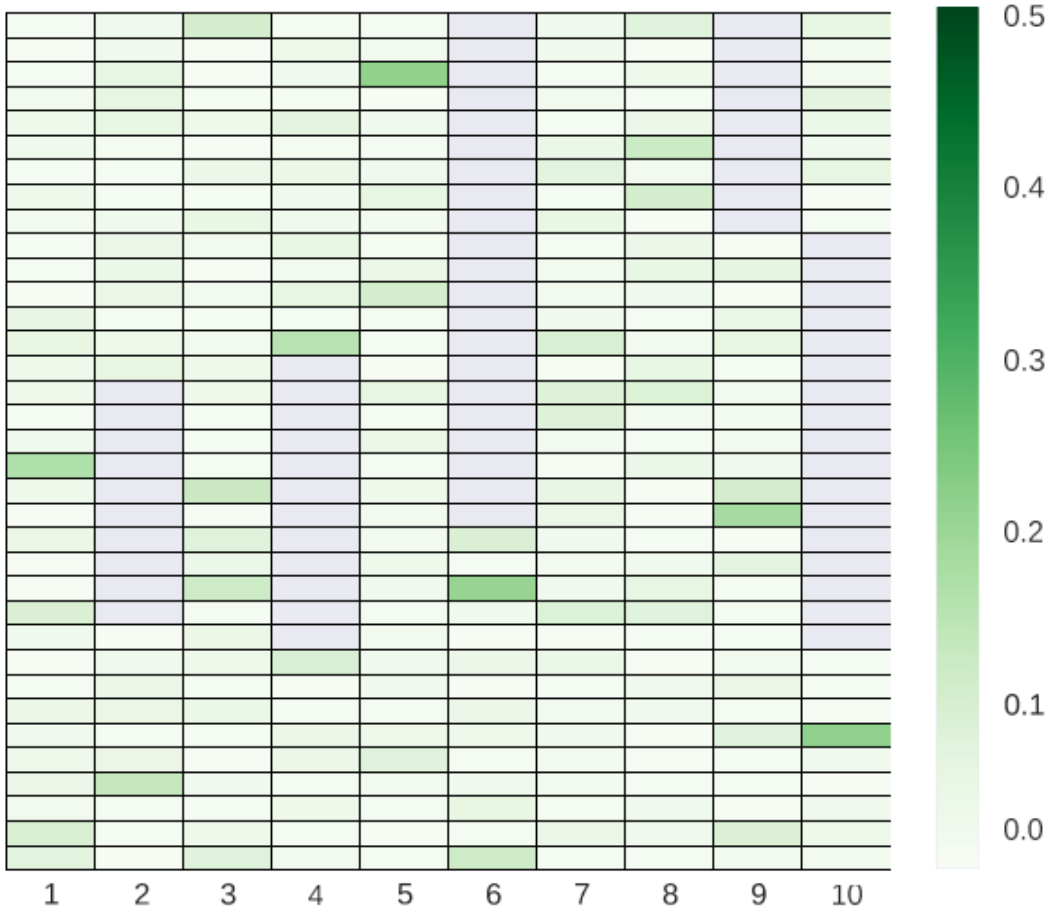

Рис. 2.2. Передбачена акустичною моделлю матриця, де кожен стовпець містить розподіл ймовірностей за символами на кожен крок часу

2.2. Налаштування RNN тренування

Процес розпізнавання мовлення складається з гарнітури, за допомогою якого люди можуть промовляти слова, програмного забезпечення для розпізнавання мовлення та мобільного телефона або комп’ютера для обробки процесу. Для перетворення звуку в текст необхідно передати розпізнаний голос в програму. Оскільки звукові хвилі є безперервним сигналом, перше, що потрібно зробити – провести дискретизацію сигналу. Цей сигнал надходить в нейронну мережу, але необхідно виконати попередню обробку сигналу, щоб отримати кращий результат і точні прогнози слів, що розпізнаються. Попередня обробка – це угруповання великого дискретизованого сигналу на невеликі фрагменти по 30 мс.

RNN – це мережа з пам'яттю, яка визначає майбутні прогнози. Це пов'язано з тим, що, оскільки він передбачає одну літеру, це впливає на ймовірність наступної літеру, яку він також передбачає. Отже, наявність пам'яті про попередні прогнози допомагає мережі робити точніші прогнози в майбутньому. RNN використовує ідею послідовної інформації, так як ця нейронна мережа має пам'ять, що впливає на майбутні прогнози. Для прогнозів використовується послідовна інформація, що зберігається у пам'яті RNN. Ідея використовувати RNN замість традиційної нейронної мережі полягає в тому, що в традиційній нейронній мережі передбачається, що кожен вхід і кожен результат залежать друг від друга. Отже, використання традиційної нейронної мережі – погана ідея при обробці мови. Передбачення будь-яких слів у реченні вимагає інформації про слово, що використовується до того, як минуле слово обробляється. Наявність пам'яті – одна з особливостей RNN, що робить її унікальною проти іншими мережами. Отже, RNN є найефективнішою для розпізнавання мови.

Ядром системи є рекурентна нейронна мережа (RNN), навчена приймати мовні спектрограми і створювати транскрипції українського тексту. Візьмемо один вираз x і одна мітка y навчальний набір $X = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots\}$.

Кожен вираз x є залежним від ряду довжини T, де кожен часовий відрізок є вектором звукових характеристик $x_t^{(i)}, t = 1, \dots, T^{(i)}$. Використаємо спектрограми як характеристики, через що $X_{t,p}^{(i)}$ визначає потужність p-го діапазону частот у звуковому кадрі в момент часу t. Метою RNN є перетворення вхідної послідовності x на послідовність ймовірностей символів y, з $y_t = P(C_t|x)$, де $c_t \in \{а, б, в, \dots, я, \text{пробіл}, \text{апостроф}\}$.

Модель RNN складається з 5 шарів прихованих одиниць. Перші три шари не повторюються. Для першого шару в кожен момент часу t вихід залежить від кадру спектрограми $x_t^{(i)}$разом із кадрами C на кожній стороні. Решта неповторюваних шарів працюють незалежно на кожному етапі і кроку часу. Таким чином, для кожного моменту часу t, перші три шари обчислюються за допомогою формули:

$$h_t^{(l)} = g\left(W^{(l)}h_t^{(l-1)} + b^{(l)}\right) \tag{2.6}$$

Четвертий шар – двонаправлений рекурентний шар. Він включає два набори прихованих одиниць: набір із прямим повторенням $h^{(f)}$ і набір із зворотнім повторенням $h^{(b)}$:

$$h_t^{(f)} = g\left(W^{(4)}h_t^{(3)} + W_r^{(f)}h_{t-1}^{(f)} + b^{(4)}\right) \tag{2.7}$$

$$h_t^{(b)} = g\left(W^{(4)}h_t^{(3)} + W_r^{(f)}h_{t+1}^{(f)} + b^{(4)}\right) \tag{2.8}$$

П’ятий (неповторюваний) рівень приймає як набір із прямим повторюванням, так і зі зворотнім як вхідні дані. Вихідний шар є стандартною функцією, що дає прогнозовані ймовірності символів для кожного відрізку часу і символу в алфавіті:

$$h_{t,k}^{(6)} = \hat{y}_{t,k} \equiv P(c_t = k|x) = \frac{\exp\left(W_k^{(6)}h_t^{(5)} + b_k^{(6)}\right)}{\Sigma_j \exp\left(W_j^{(6)}h_t^{(5)} + b_j^{(6)}\right)} \tag{2.9}$$

Після того, як був обчислений прогноз, можна прорахувати втрати CTC для вимірювання помилки в передбачувані. Під час навчання можна порахувати градієнт по відношенню до виходів мережі, зважаючи на послідовність символів y. Для навчання можна використати швидкий градієнтний метод Нестерова.

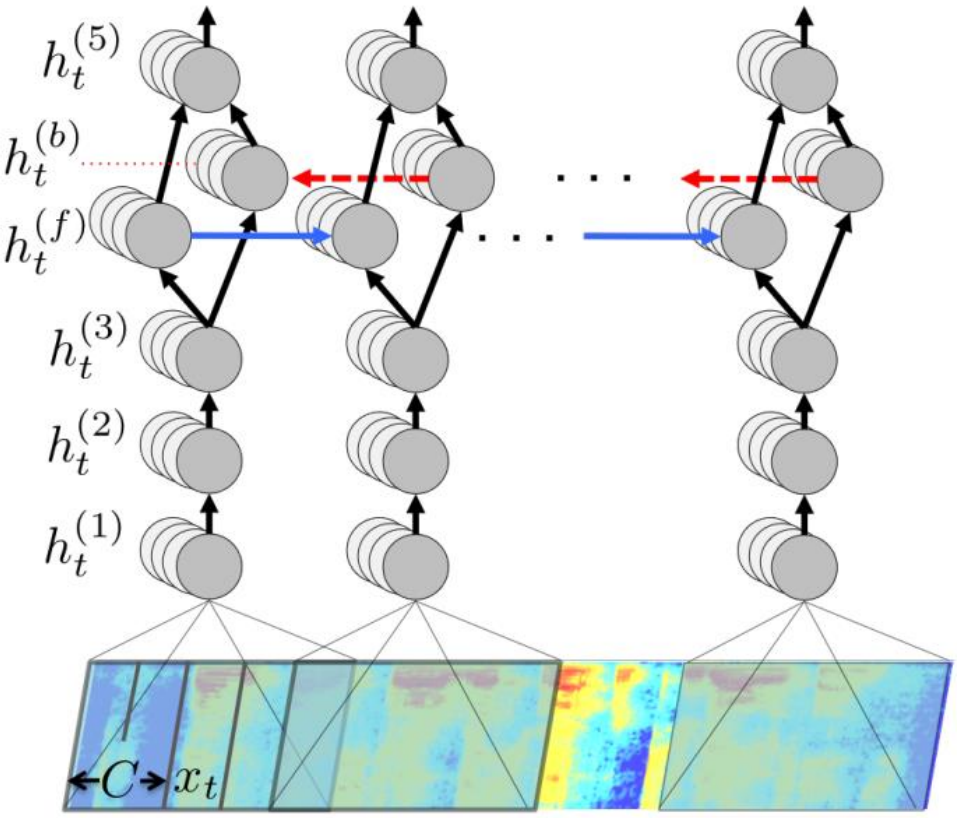

Рис. 2.3. Структура RNN моделі

2.3. Мовна модель

При навчанні на основі великої кількості мовних даних, модель RNN може навчитися створювати читабельні транскрипції на рівні символів. Для багатьох транскрипцій найімовірніша послідовність символів, передбачена RNN, є точно правильною без зовнішніх мовних обмежень. Помилки, зроблені RNN у цьому випадку, є фонетично вірогідним визначенням українських слів. Багато помилок виникають в словах, які рідко або взагалі не зустрічаються в навчальному наборі.

RNN	Транскрипція
Сьогодні чудова погода, перший день зими	Сьогодні чудова погода, перший день зим

Рис. 2.4. Приклад транскрипції

Враховуючи результат моделі, можна виконати пошук, щоб знайти послідовність символів c1, c2,…, що найбільше відповідають до вихідних даних RNN і мовної моделі (де мовна модель інтерпретує рядом символів як слова). Зокрема, необхідно знайти послідовність c:

$$Q(c) = \log\big(P(c|x)\big) + \alpha \log\big(p_{Im(C)}\big) + \beta(C) \qquad (2.10)$$

2.4. Паралелізм даних

Для ефективної обробки даних можна використовувати два рівні паралелізму даних. По-перше, кожен процесор обробляє багато прикладів паралельно. Це робиться звичайним шляхом об’єднання багатьох прикладів у єдину матрицю. Наприклад, замість виконання єдиного множення матриці на вектор у повторюваному шарі, краще зробити багато процесів, обчислюючи $w_r H_t$. Процесор більш ефективний, коли H відносно великий (приблизно 1000 прикладів або більше), тому за краще обробляти якомога більше процесів на одному графічному процесорі (до обмеження пам’яті графічного процесора).

Паралелізм даних забезпечує пришвидшене навчання, проте групування більшої кількості прикладів в одне оновлення градієнту не покращить швидкість навчання. Тобто обробка вдвічі більшої кількості прикладів на вдвічі більшу кількість процесорів не прискорює навчання у два рази.

2.5. Синтез порівнянням сигналів

Для розширення потенціалу навчальних даних, використовується синтез даних, який успішно застосовується в інших контекстах для збільшення ефективної кількості навчальних вибірок [26]. Метою цієї роботи є підвищення продуктивості розпізнавання голосу, особливо в шумному середовищі, так як наявні системи показують низьку продуктивність в подібних умовах.

Використовування міток для пошуку даних з фоновим шумом непрактично, саме тому встає гостра необхідність у пошуку інших способів генерувати такі дані.

В першу чергу, аудіо сигнали генеруються через процес порівняння наявних вихідих сигналів. Тому можна використовувати це для синтезу шумових навчальних даних. Наприклад, якщо є звукова доріжка з голосом $x^{(i)}$ і звукова доріжка з фоновим шумом $\xi^{(i)}$, тоді можна згенерувати доріжку з голосом і фоновим шумом як $\hat{x}^{(i)} = x^{(i)} + \xi^{(i)}$, аби симулювати доріжку в шумному середовищі. Якщо необхідно, можна додати реверберацію, ехо або інші форми звукових фонових шумів для створення реалістичних звукових сцен.

Однак в цьому підході є певні ризики. Наприклад, для того аби обробити 1000 годин звукової доріжки з чистим мовленням і створити 1000 годин доріжки з фоновим шумом, знадобляться унікальні шумові треки, що охоплюють приблизно таку кількість годин. Не можна використовувати, умовно, 10 годин повторюваного шуму для цієї задачі, оскільки рекурентна мережа може запам’ятати лише конкретний шумовий трек і «віднімати» його з синтезованих даних.

Таким чином, замість використання одного джерела шуму тривалістю 1000 годин, можна використовувати велику кількість коротких треків (які легше збирати з загальнодоступних джерел відео) і розглядати їх як окремі джерела шуму: $\hat{x}^{(i)} = x^{(i)} + \xi_1^{(i)} + \xi_2^{(i)} + \cdots$.

При накладенні багатьох сигналів, зібраних із відеокліпів, можна отримати

«шумові» звуки, які відрізняються від шумів, записаних у реальному середовищі. Щоби запезпечити гарну відповідність між синтетичними даними та реальними даними, треба відхили будь-які потенційні шумові треки, де середня потужність у кожній частотній смузі значно б відрізнялась від середньої потужності, спостережуваної в реальному шуму запису.

Одним із складних ефектів, з яким стикаються системи розпізнавання мовлення в шумному середовищі – це «Ефект Ломбарда» [13]: мовці активно змінюють висоту або інтонацію свого голосу, щоби подолати шум навколо себе. Цей мимовільний ефект не проявляється в записаних наборах даних мовлення. Саме тому для подолання цього ефекту при тренуванні набору даних людині необхідно записувати голос в умовах фонового шуму в навушниках, аби зафіксувати цей ефект.

РОЗДІЛ 3
РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Технології розробки програмного забезпечення

Dart — мова програмування, яку розробляє компанія Google, позиціонуючи як мову структурованого програмування для Веб. Розробники вважали, що в довгостроковій перспективі Dart може стати прогресивною заміною JavaScript, котрий потерпає від наявних в даний час проблем з розширюваністю, продуктивністю і підтримкою розробки складних застосунків. Мова має схожий на Java синтаксис, не вимагає явного визначення типів і її можна використовувати для створення серверних та клієнтських застосунків.

У березні 2015 компанія Google представила оновлену стратегію просування Dart, у котрій вирішено не прив'язувати Dart до браузеру і відмовитися від ідеї інтеграції віртуальної машини Dart у Chrome. Розробку буде зосереджено на застосуванні Dart як проміжної мови, скомпільованої в JavaScript. Розвиток Dart як окремої мови, альтернативної JavaScript і безпосередньо підтримуваної у браузерах, визнано недоцільним. Замість цього Dart рухатиметься у бік якіснішої інтеграції з JavaScript і генерації оптимального JavaScript-коду. При цьому розробку віртуальної машини Dart VM буде продовжено, але вона позиціонуватиметься в основному для створення серверних і мобільних застосунків.

Влітку 2014 асоціація ECMA International, що займається стандартизацією інформаційних і комунікаційних технологій, затвердила специфікацію ECMA-408, яка стандартизує синтаксис і семантику мови Dart, а також склад базових бібліотек і супутніх мові технологій, відтоді мова Dart є офіційним стандартом Ecma. Надання Dart статусу стандарту Ecma дозволить розширити область використання мови та прискорити забезпечення його підтримки в наявних на ринку браузерах і продуктах. Вибір Ecma International як організації для

стандартизації обумовлений тим, що ця асоціація вже розвиває близькі до специфіки Dart стандарти для мов JavaScript, Eiffel і C#. Просуванню Dart як стандарту сприяло надання компанією Google всіх пов'язаних з розробкою патентів у безоплатне використання, що не вимагає оплати відрахувань (royalty free).

Flutter – це розроблений Google фреймворк з відкритим програмним кодом, який дозволяє просто і швидко створювати мобільні додатки для iOS і Android.

При цьому в роботі Flutter не використовує нативні компоненти зовсім. Замість цього всі UI-елементи у фреймворку створюються за допомогою власного графічного движка. Flutter дозволяє створювати всі елементи призначеного для користувача інтерфейсу додатку з готових віджетів. У цьому Flutter схожий з іншими фреймворками – React і Vue, і в той же час має ряд відмінностей від них. Так, він не використовує мову програмування Javascript, натомість Flutter використовує мову Dart.

Обидва фреймворка використовуються для розробки мобільних додатків. В цілому, у React Native більша власна бібліотека UI-елементів, ніж бібліотека віджетів Flutter.

Однак перевага останнього в даному випадку в тому, що він не настільки залежний від сторонніх бібліотек-елементів, як React Native. Деякі елементи в них виявляють несумісність з конкретними платформами. Можна сказати, що Flutter в даному випадку більш універсальний і широко застосовується. Крім того, Flutter перевершує React Native і по продуктивності, використовуючи повністю відмінний підхід до рендерингу.

Так, Flutter створює власні віджети і використовує графічний процесор для рендеринга, а не запозичує нативні компоненти з інших платформ.

Написаний на мові Dart код Flutter компілює в безпосередньо оброблюючий процесором код ARM. Завдяки цьому додатки, створювані на Flutter, працюють помітно швидше. Тоді як у React Native міст Javascript, який використовується для інтерпретації UI-елементів та викликає Java API або

Objective-C для відображення відповідно компонентів iOS і Android, може уповільнювати роботу додатків.

TensorFlow – це бібліотека програмного забезпечення з відкритим кодом для чисельних розрахунків з використанням графів потоку даних. Вузли графу представлені у вигляді математичних операцій, в той час як ребра графу представляються багатовимірними масивами даних (тензорами), що передаються між вузлами.

TensorFlow був створений і підтримується командою Google Brain в рамках дослідницької організації Google Machine Intelligence для ML та DL. Зараз він випускається під ліцензією відкритого коду Apache 2.0. Інтерфейси програмування TensorFlow включають Python та C ++ з планами для API Java, GO, R та Haskell, також підтримуються хмарні середовища Google та Amazon. TensorFlow призначений для широкомасштабних розподілених тренувань, сюди входить 200 стандартних операцій, включаючи математичні, маніпуляції з масивом, управління потоками та операції з управління станом, написані на C ++. На відміну від інших бібліотек DL, які в основному орієнтовані на дослідження (наприклад, Theano), TensorFlow був розроблений для використання як у системах досліджень, так і у розробці та виробництві ПЗ. TensorFlow підтримується процесорами, графічними процесорами, мобільними пристроями і широкомасштабними розподіленими системами, які складаються з сотень вузлів. Крім того, існує TensorFlow Lite – легке рішення TensorFlow для мобільних та вбудованих пристроїв. Це дає змогу випускати ML-орієнтовані рішення на пристрої користувачів дуже швидко та з невеликим двійковим розміром, але має підтримку обмеженого набору систем. Також підтримується апаратне прискорення за допомогою Android Neural Networks API.

Сильні сторони:

– На сьогоднішній день найпопулярніший інструмент DL, з відкритим кодом, швидким залученням, який добре підтримується сильною промисловою компанією (Google).

- Потужна чисельна бібліотека для програмування потоків даних, яка є основою для досліджень та розробок DL.
- Ефективно працює з математичними виразами, що включають багатовимірні масиви.
- Дуже добре задокументована.
- Обчислення на GPU / CPU, мобільні обчислення, висока масштабованість обчислень на різних машинах та величезні набори даних.
- Вищий шар абстракції, ніж Theano.

Слабкі сторони:

- API нижчого рівня важко використовувати безпосередньо для створення моделей DL.
- Кожен обчислювальний потік повинен бути побудований як статичний графік (хоча пакет Tensorflow Fold намагається полегшити цю проблему), а також відсутні символьни цикли.

Python – мова програмування загального призначення, орієнтована на підвищення продуктивності розробника і читання коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій.

Мова програмування Python – це потужний інструмент для створення програм найрізноманітнішого призначення, доступний навіть для новачків. З його допомогою можна вирішувати завдання різних типів.

Python має деякі примітні особливості, які обумовлюють її широке поширення.

Python – інтерпретована мова програмування. З одного боку, це дозволяє значно спростити налагодження програм, з іншого – зумовлює порівняно низьку швидкість виконання.

- Інтерпретатор Python реалізований практично на всіх платформах та операційних системах.

- Розширюванність мови (є можливість удосконалювати мову усіма зацікавленими програмістами)
- Інтерпретатор написаний на C і вихідний код доступний для будь-яких маніпуляцій. У випадку необхідності можна вставити його в свою програму та використовувати як вбудовану оболонку. Або ж, написавши на C свої доповнення, отримати “розширений” інтерпретатор з новими можливостями.
- Наявність великої кількості модулів, що підключаються до програми, які забезпечують різноманітні додаткові можливості.
- Простота та гнучкість мови.

3.2. Збір даних для навчання акустичної моделі

Для навчання нейронних мереж розпізнаванню мовлення потрібні сотні годин тренувальних даних. Через відсутність у відкритому доступі достатньо великих наборів даних для навчання розпізнаванню української мови, найбільш трудомістким завданням цієї роботи стало завдання збору даних. Зважаючи на обмеженість ресурсів і часу, оптимальним рішенням стала розробка методів для автоматичного збору та фільтрації даних з відкритих джерел.

Набір даних є сукупністю з 3-х текстових файлів формату csv: train.csv, dev.csv і test.csv та відповідних директорій з аудіофайлами уривків мови у форматі wav. Аудіофайли формату wav мають єдиний звуковий канал (моно), частоту дискретизації 16000 Гц та закодовані з шириною 2 байти для кожного значення.

Також з алгоритму декодування CTC матриці випливає обмеження на співвідношення довжини аудіо та кількість символів у транскрипті, а саме: кількість кроків у матриці CTC має бути більшою, ніж кількість символів у транскрипті.

Кожен із csv файлів містить таблицю даних зі стовпцями wav_filename, wav_filesize та transcript. Стовпець wav_filename вказує повний шлях до файлу wav, а в стовпці transcript вказаний текст, який вимовляється в цьому аудіо-файлі.

У стовпці wav_filesize вказується розмір відрізка аудіо в байтах. Інформація про розмір файлу використовується для сортування прикладів під час навчання нейронної мережі.

На початку процесу отримання транскрибованих аудіофайлів проводиться пошук відео за допомогою YouTube API. Встановлюється прапорець для пошуку відео лише з субтитрами, які були додані автором відео. Для пошуку безлічі запитів у циклі існує файл зі списком текстових запитів. Цей файл заповнюється запитами на певну тему вручну. Після пошуку відеозаписів, знайдені ідентифікатори відео зберігаються в список для подальшої обробки.

Далі для кожного з відео з отриманого списку робиться спроба скачування автоматично розпізнаних субтитрів YouTube у форматі субтитрів vtt. Для завантаження субтитрів та аудіо з YouTube була використана утиліта youtube-dl. В автоматичних субтитрах зазначено час початку кожного із розпізнаних слів. У python скрипті проводиться побудова відповідності слова та його тимчасового проміжку на основі цих даних.

Після цього завантажуються завантажені автором субтитри (вони є тому, що відео знайдено за допомогою відповідного фільтра). Далі для кожного розміченого в авторських субтитрах відрізка здійснюється пошук точної відповідності (за словами) в автоматично згенерованих субтитрах. Такої відповідності може не знайтись, тому що якісь слова були не розпізнані або тому, що авторські субтитри не відповідають мовленню у відео. Якщо ж відповідність знайдена, то ми отримуємо більш точні, ніж в авторських субтитрах, позначки часу початку і кінця мови, а також впевненість у тому, що в аудіо файлі є певний текст (оскільки автоматична система розпізнавання «почула» цей текст у аудіо). Отримати точні позначки часу початку і кінця розмови важливо, щоб не обрізати на півслові і не включити слова, яких немає в транскрипті. Якщо хоч одну відповідність було знайдено, то проводиться завантаження аудіо-файлу та його розбиття.

Так як позначки часу, отримані з файлу автоматично згенерованих субтитрів, все одно іноді виявляються посередині слова, що вимовляється, то

проводиться корекція цих позначок до найближчого місця тиші в межах 0.5 секунди. Для визначення тиші в аудіо використовується утиліти Voice Activity Detector із open-source проекту WebRTC.

Для кожного відео в окремій директорії зберігаються всі дані цього відео: вихідний аудіо-файл, конвертований wav аудіо-файл, parts.csv - документ з транскриптами для успішно оброблених відрізків цього відео, директорія parts з wav аудіо-відрізками, а також файл stats.csv, де збережена статистика про обробку цього відео: кількість секунд та кількість успішно вилучених аудіо-відрізків.

Для успішно скоригованого відрізка (якщо корекція була потрібна), з аудіо файлу вирізається розмічений відрізок аудіо у форматі wav і зберігається в директорію parts, а файл parts.csv записується повний шлях до цього відрізка, розмір відрізка аудіо в байтах і відповідний йому текст . Якщо на етапі обробки відео відбувається якась помилка, то це відео додається до списку проблемних відео з описом проблеми, щоб у подальшому не витрачати на нього ресурси, якщо воно знову трапляється в пошуку за будь-яким запитом. Ідентифікатори успішно оброблених відео також зберігаються в окремий файл, щоб запобігти повторній обробці та статистиці.

Даний підхід виявився найточнішим у сенсі відповідності тексту транскрипта та сказаного в аудіо тексту. Недоліком цього підходу виявилася занадто висока вибірковість алгоритму: якщо для рядка авторських субтитрів алгоритм розпізнавання мови YouTube розпізнав хоча б одне слово невірно, то цей рядок викидається з набору даних.

Внаслідок цього швидкість складання корпусу виявилася дуже низькою. Також за статистикою збору даних було зрозуміло, що лише близько 2.5% від усього завантаженого аудіо входить до результуючого корпусу. У результаті таким способом за кілька днів вдалося створити корпус обсягом 140 годин промови.

Виходячи з того, що в багатьох джерелах вказується необхідний об'єм корпусу для навчання нейронної мережі розпізнаванню мови, що дорівнює 400-

1000 годин, цей підхід бачився не оптимальним в умовах обмеженого часу та кількості відео з субтитрами в українському сегменті YouTube.

На початку процесу також виходить будь-яким шляхом список ідентифікаторів відео YouTube. Далі відфільтровуються всі відео, для яких не було створено автоматичні субтитри YouTube. Для відео з автоматичними субтитрами здійснюється завантаження аудіо та файлу vtt автосубтитрів.

Аудіо конвертується у формат wav із параметрами: 16кГц, моно, 16 біт. Зі скачуваного файлу субтитрів vtt витягується розпізнаний системою транскрибування YouTube приблизний (з помилкою близько 0.3-0.5 секунд) час початку кінця кожного слова. Аудіо розбивається в проміжках між мовою (які визначаються за допомогою детектора WebRTC VAD) на відрізки від 5 до 20 секунд.

Після цього для кожного з відрізків вибираються слова, які за часом належать йому. Вибрані слова сортуються за часом початку їх вимови і збираються в один рядок, розділений символом пропуску. Після цього до аудіо застосовується коригування гучності, щоб довести його до рівня -10 децибел і полосовий частотний фільтр (bandpass filter) з межами 200Гц та 3000Гц.

Даний спосіб генерації корпусу набагато швидший за попередній з двох причин. Перше це те, що точна обрізка аудіо за проміжками тиші проводиться відразу на першому етапі послідовно для всього аудіофайлу, на відміну від попереднього способу, де для кожного відрізка була спроба скоригувати початок і кінець аудіо.

А друга полягає в тому, що в цьому способі не використовуються субтитри користувача, що дає доступ до набагато більшого обсягу відео ціною якості транскрипта.

Отримання 100 годин мови займає 1:00 реального часу. Що порівняно з попереднім способом у 50 разів швидше.

3.3. Навчання моделі

При наявності архітектури нейроннох мережі, функції втрат та даних для навчання, можна зробити навчання акустичної моделі.

Для налаштування процесу навчання доступні такі основні параметри:

1) n_hidden – число нейронів у прихованих шарах нейроннох мережі. Вважається, що цей параметр повинен бути обраний залежно від обсягу навчальних даних та його різноманітності. Для англійської мови у проєкті DeepSpeech стандартне значення дорівнює 2048. Максимальне значення обмежене обсягом доступної відеопам’яті.
2) learning_rate – швидкість навчання. Стандартне значеня з Mozilla DeepSpeech: 0.0001.
3) epoch – число епох або кількість повних проходжень процесу навчання за всіма тренувальними даними.
4) train_batch_size, dev_batch_size, test_batch_size – кількість прикладів в одному батчі для тренування, валідації та тестування відповідно. Обирається максимальне значення, виходячи з обсягу відеопам’яті. Зазвичай використовують значення від 8 до 64.
5) dropout_rate – можливість виключення нейрона з обчислень під час одного кроку навчання. Вважається, що цей підхід дозволяє уникнути перенавчання мережі. Стандартне значення 0.3.

Також дуже важливим параметром, який не було внесено до списку, є дані для навчання та тестування. Саме від їньої якості залежить кінцевий результат.

Експерименти проводилися з використанням трьох наборів даних: uk_1k, uk_clean_650 та uk_speach_clean. Перші два набори складаються з промови людей у відео-роліках Youtube, записаних у різних шумових умовах. Останній набір складається з чистих записів читання вголос уривків літературних творів українською мовою.

Навчання нейронної мережі в більшості випадків проводилося до того моменту, коли протягом кількох епох метрика втрат CTC Loss перестає знижуватися або починає

збільшуватися. В решті експериментів навчання проводилося до досягнення мінімального показника WER.

На початку були проведені іспити на основі частини набору «uk_clean_650» для визначення оптимальної кількості нейронів у прихованих шарах нейронної мережі (параметр n_hidden). При тестування застосовувалася мовна модель із довжиною послідовності 3. Накріщий варіант обирався на основі мінімального показника WER.

Після експериментів з виявлення найкращого значення параметра n_hidden були проведені експерименти на обсязі даних обох наборів для визначення мінімально досяжної помилки WER. Також був проведений експеримент із донавчанням моделі, навченої на базі «uk_1k» і «uk_clean_650». Тестування проводилося на наборах даних «uk_clean_650_test» та «uk_speach_clean_test». Для кожного набору було проведено тестування з використанням мовної моделі і без моделі. При першому випадку тестування застосовувалася модель із довжиною послідовності 3.

Далі були проведені експерименти щодо виявлення оптимальної розмірності мовної моделі. Було здійснено тестування кращої акустичної моделі з попереднії експериментів із різними конфігураціями мовної моделі. Найркаще значення розмірності обиралося на основі мінімального значення WER.

3.4. Опис інтерфейсу користувача

Результатом роботи є функціонуючий мобільний застосунок з функцією розпізнавання української мови і конвертування в текст.

Інтерфейс додатку є стандартним та зручним для розуміння користувачу будь-якого рівня. Після запуску програми, користувач має змогу бачити головне меню додатку (рис.3.1). Увесь функціонал додатку є інтуїтивно зрозумілим і не передбачає спеціалізованих знань у даній сфері. Окрім того, для кращого розуміння роботи додатку, користувач може прочитати коротку інструкцію з користування.

Розроблений програмний продукт призначений для розпізнавання голосу українською мовою за допомогою навчених моделей мови на базі двигуна DeepSpeach на мобільній платформі за операційних систем Android та IOS. Існує

перевірка на сумсність версії операційної системи з розробленим додатком.

Кожен застосунок на Android або IOS складається з безлічі різних файлів, як в даному випадку, два функціональних класів, один файл з константами і чотири файли з особистою бібліотекою для відкриття каналів і перевірки локальних моделей мови. Щоб додаток міг коректно встановлюватися і працювати, необхідно правильно упакувати всі складові частини в один великий архівний файл, який називається APK (Android Package Kit). Для установки файлу APK не потрібно нічого, крім самого смартфона або планшета на Android або IOS.

Після загрузки файлу APK, необхідно пройти усі етапи встановлення додатку на смартфон за рекомендаціями від Android або IOS. Після успішного встановлення буде екран підтвердження завершення завантаження додатку. Іконка встановленого мобільного додатку автоматично буде відображатись на головному екрані телефону користувача. Наступним етапом буде запуск додатку на смартфоні чи планшеті (рис.3.1).

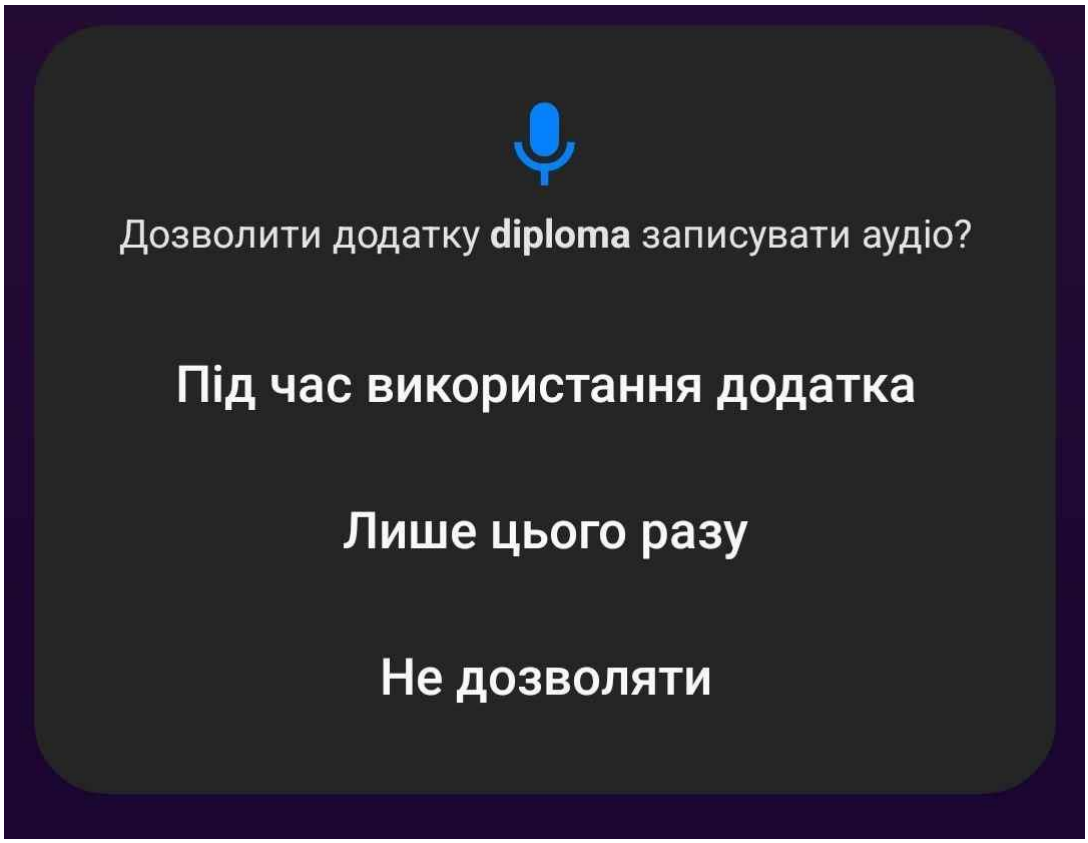

Рис.3.1 Перший запуск мобільного застосунку

У ході тестування мобільного застосунку було виявлено, що сервіси

розпізнавання голосу на основі двигуна DeepSpeech можуть працювати некоректно в залежності від версії операційної системи та потужності внутрішніх компонентів. Так, з двох реальних мобільних смартфонів, Samsung Galaxy S21, куплений у 2021 році, працював досить швидко, без помилок, відображаючи більш точні результати, на відміну від Ulefone S7, який був куплений у 2017 році.

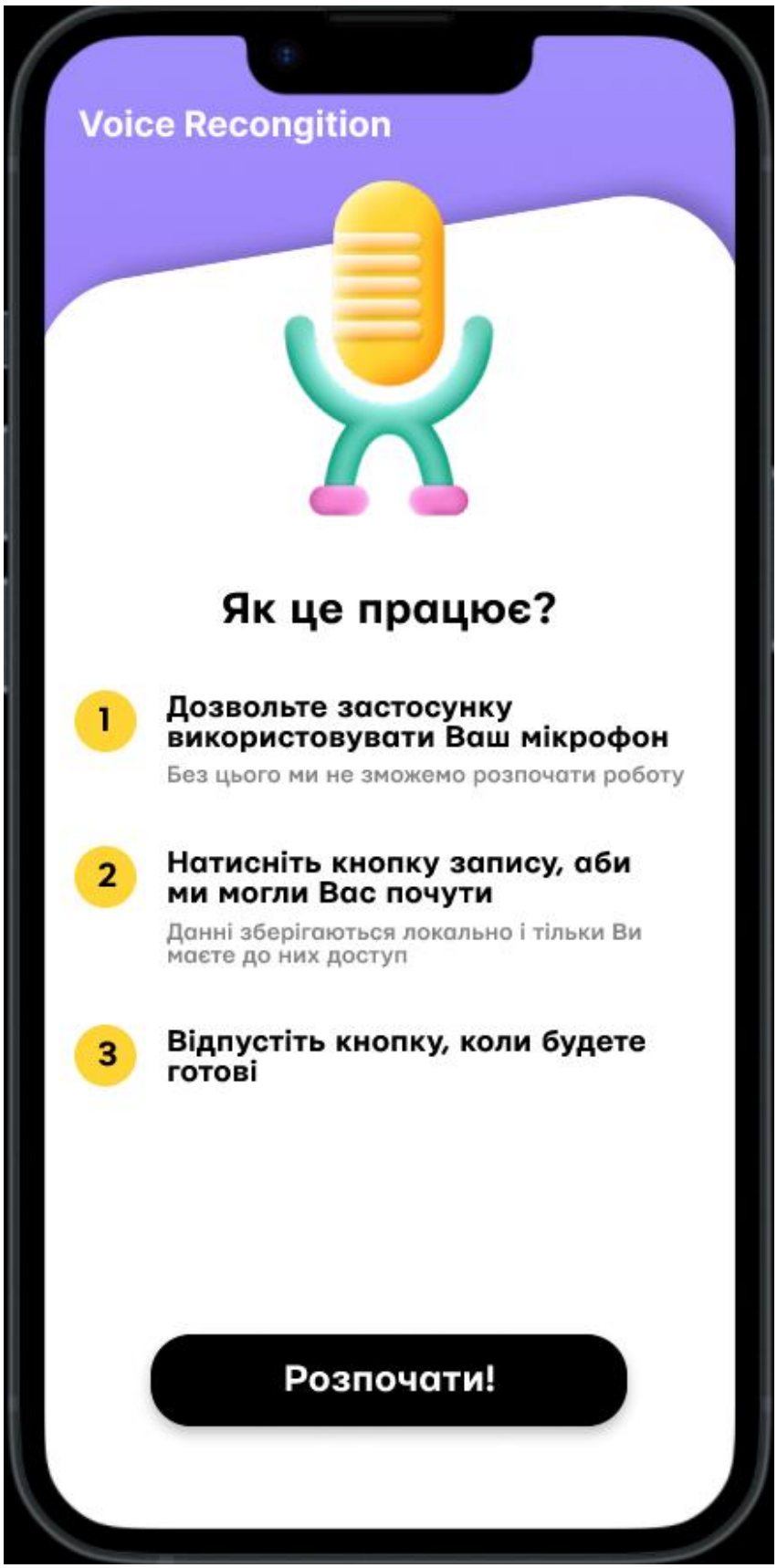

Рис.3.2. Початковий екран застосунку

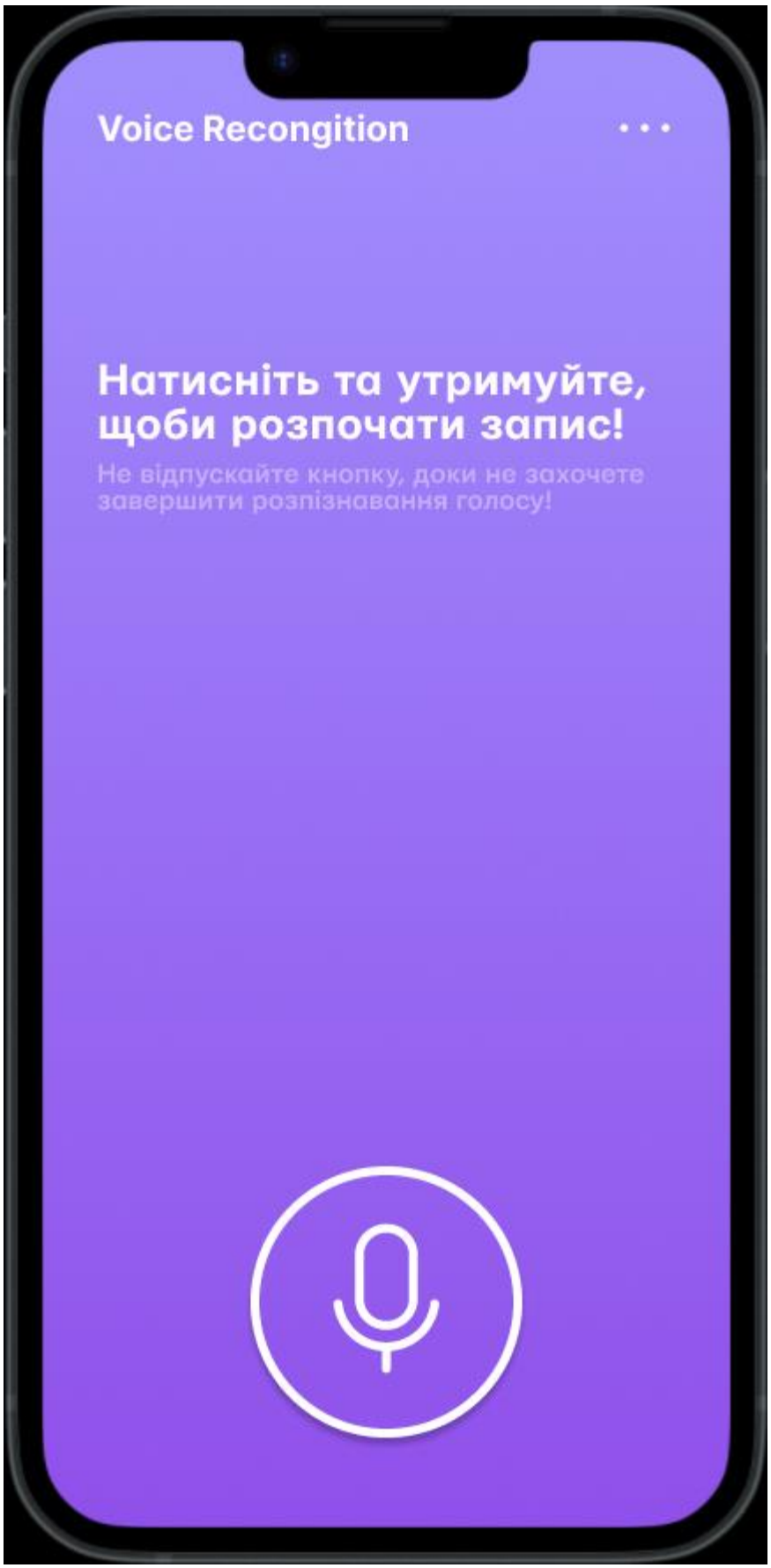

Рис.3.3. Головний екран застосунку

Рис.3.4. Екран початку запису голосу

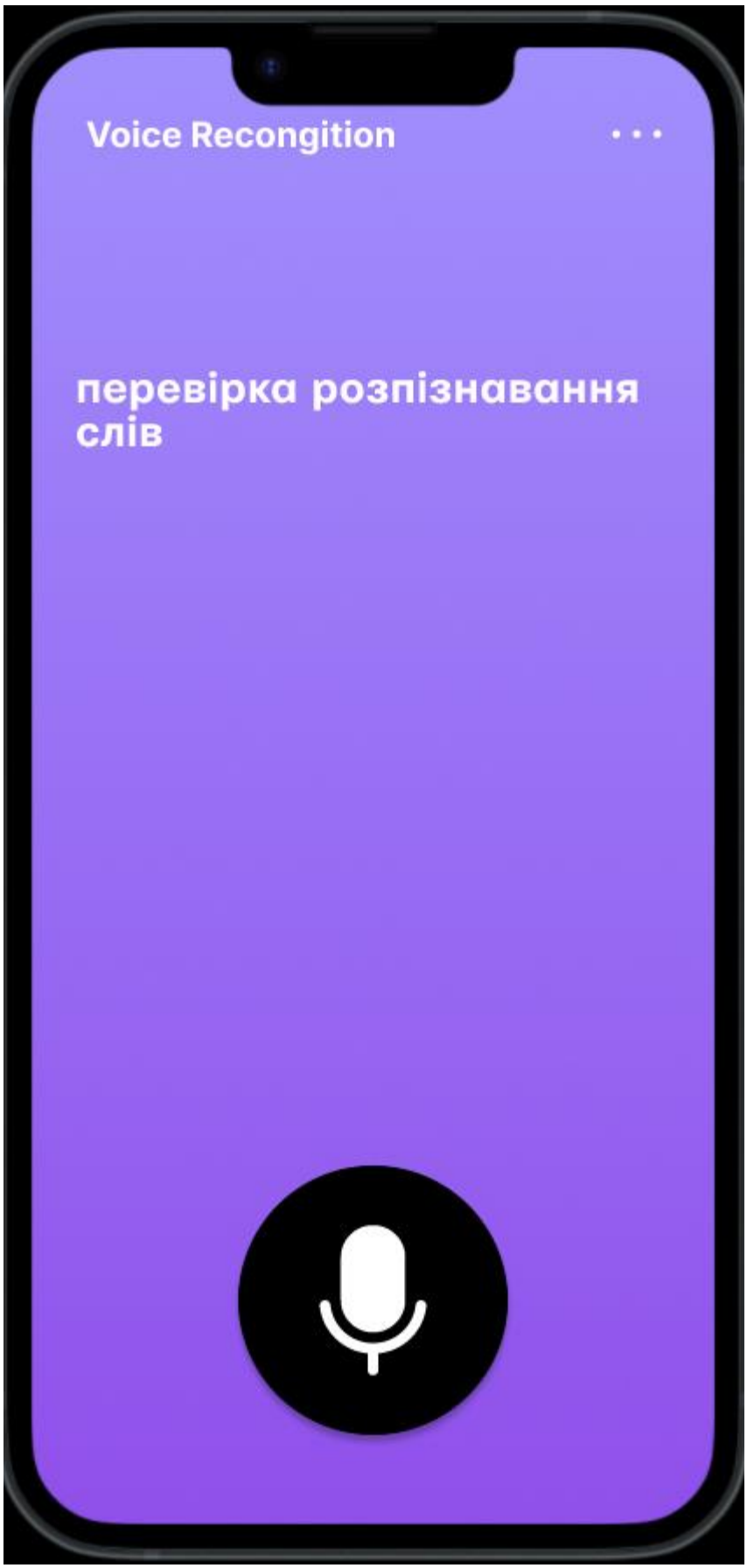

Рис.3.5. Вивід результату на екран

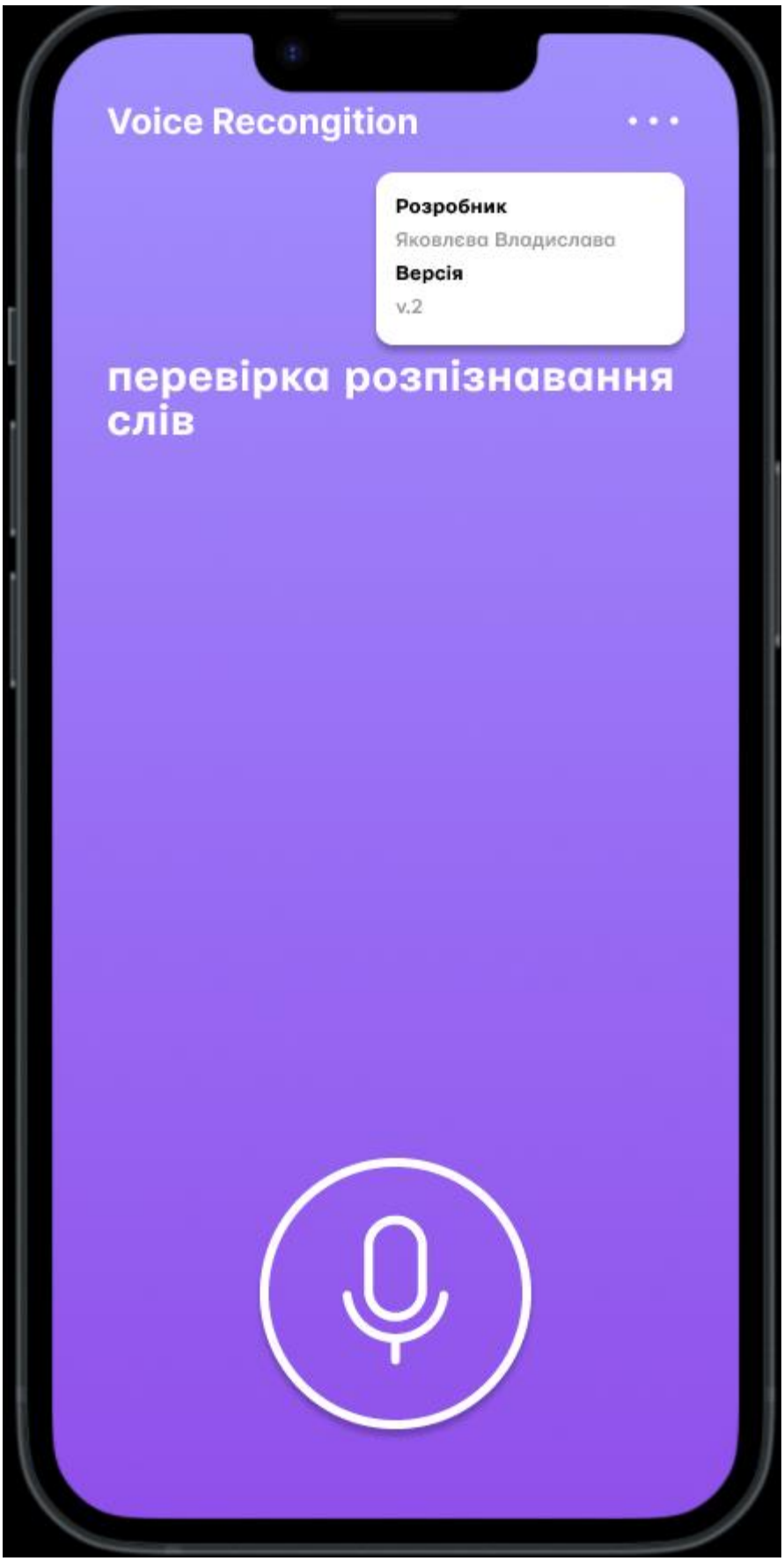

Рис.3.6. Додаткова кнопка

При практичній перевірці функціоналу розпізнавання голосу (рис.3.5.), якість виводу тексту була задовільною.

ВИСНОВКИ

За результатами проведеної роботи можна зробити такі висновки:

1) розглянуто принцип роботи системи розпізнавання голосу з використанням нейроних мереж на основі архітектури DeepSpeech;
2) досліджено методи автоматичного збору даних для створення акустичних моделей транскрибованого мовлення;
3) створена система розпізнавання української мови, яка показує результат в 31% WER на корпусі з різномаїттям голосів та фонових шумів;
4) створений корпус української мови обсягом у 650 годин, які можна використовувати для сучасних систем розпізнавання голосу з використанням нейронних мереж.

На основі метода створенний корпус транскрибованого мовлення для української мови, запропонованого в цій роботі, можуть бути зібрані моделі для інших мов. Архітектура DeepSpeech також має мінімальну кількість прив’язок до конкретної мови (окрім застосованого алфавіту), що дозволяє використовувати процес навчання, описаний в цій роботі, при створенні систем розпізнавання голосу і для інших мов.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A novel connectionist system for unconstrained handwriting recognition / A. Graves [и др.] // IEEE transactions on pattern analysis and machine intelligence. — 2009. — Т. 31, № 5. — С. 855—868.
2. Addressing the rare word problem in neural machine translation / M.-T. Luong // arXiv preprint arXiv:1410.8206. — 2014.
3. Advances in joint CTC-attention based end-to-end speech recognition with a deep CNN encoder and RNN-LM / T. Hori // arXiv preprint arXiv:1706.02737. — 2017.
4. Attention-based models for speech recognition / J. K. Chorowski // Advances in neural information processing systems. — 2015. — С. 577—585.
5. Awan A. A., Subramoni H., Panda D. K. An In-depth Performance Characterization of CPU-and GPU-based DNN Training on Modern Architectures // Proceedings of the Machine Learning on HPC Environments. — ACM. 2017. — С. 8.
6. Bahdanau D., Cho K., Bengio Y. Neural machine translation by jointly learning to align and translate // arXiv preprint arXiv:1409.0473. — 2014.
7. Cold fusion: Training seq2seq models together with language models / A. Sriram // arXiv preprint arXiv:1708.06426. — 2017.
8. Collobert R., Weston J. A unified architecture for natural language processing: Deep neural networks with multitask learning // Proceedings of the 25th international conference on Machine learning. — ACM. 2008. — С. 160—167.
9. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks / A. Graves [и др.] // Proceedings of the 23rd international conference on Machine learning. — ACM. 2006. — С. 369—376.
10. CTC Networks and Language Models: Prefix Beam Search Explained. — 2018. — Last Accessed: 2018-04-18. https://medium.com/corti-ai/ctc-networksand-language-models-prefix-beam-search-explained-c11d1ee23306.
11. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups / G. Hinton // IEEE Signal Processing Magazine. — 2012. — Т. 29, № 6. — С. 82—97.

12. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin / D. Amodei // arXiv preprint arXiv:1512.02595. — 2015.
13. Deep speech: Scaling up end-to-end speech recognition / A. Hannun // arXiv preprint arXiv:1412.5567. — 2014.
14. Dropout: A simple way to prevent neural networks from overfitting / N. Srivastava // The Journal of Machine Learning Research. — 2014. — Т. 15, № 1. — С. 1929—1958.
15. Empirical evaluation of gated recurrent neural networks on sequence modeling / J. Chung // arXiv preprint arXiv:1412.3555. — 2014.
16. Google voice search: faster and more accurate / H. Sak // Google Research blog. — 2015. — URL: https://research.googleblog.com/2015/09/ google-voice-search-faster-and-more.html.
17. Google's neural machine translation system: Bridging the gap between human and machine translation / Y. Wu // arXiv preprint arXiv:1609.08144. — 2016.
18. Graves A. Generating sequences with recurrent neural networks // arXiv preprint arXiv:1308.0850. — 2013.
19. Graves A. Supervised sequence labelling // Supervised sequence labelling with recurrent neural networks. — Springer, 2012. — С. 52—73.
20. Graves A., Jaitly N. Towards end-to-end speech recognition with recurrent neural networks // International Conference on Machine Learning. — 2014. — С. 1764—1772.
21. Graves A., Mohamed A.-r., Hinton G. Speech recognition with deep recurrent neural networks // Acoustics, speech and signal processing (icassp), 2013 ieee international conference on. — IEEE. 2013. — С. 6645—6649.
22. Heafield K. KenLM: Faster and smaller language model queries // Proceedings of the Sixth Workshop on Statistical Machine Translation. — Association for Computational Linguistics. 2011. — С. 187—197.
23. How many hidden units should I use? — 2014. — Last Accessed: 2018-05-20. http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-10.html.
24. Hu H., Zahorian S. A. Dimensionality reduction methods for HMM phonetic recognition // Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on. — IEEE. 2010. — С. 4854—4857.

25. Hybrid CTC/Attention Architecture for End-to-End Speech Recognition / S. Watanabe // IEEE Journal of Selected Topics in Signal Processing. — 2017. — Т. 11, № 8. — С. 1240—1253.
26. Kiros R., Salakhutdinov R., Zemel R. S. Unifying visual-semantic embeddings with multimodal neural language models // arXiv preprint arXiv:1411.2539. — 2014.
27. Le Q. V., Jaitly N., Hinton G. E. A simple way to initialize recurrent networks of rectified linear units // arXiv preprint arXiv:1504.00941. — 2015.
28. Medsker L., Jain L. Recurrent neural networks // Design and Applications. — 2001. — Т. 5.
29. Mu¨ller M. Dynamic time warping // Information retrieval for music and motion. — 2007. — С. 69—84.
30. Pascanu R., Mikolov T., Bengio Y. Understanding the exploding gradient problem // CoRR, abs/1211.5063. — 2012.
31. Rabiner L. R. A tutorial on hidden Markov models and selected applications in speech recognition // Proceedings of the IEEE. — 1989. — Т. 77, № 2. — С. 257—286.
32. Sak H., Senior A., Beaufays F. Long short-term memory recurrent neural network architectures for large scale acoustic modeling // Fifteenth annual conference of the international speech communication association. — 2014.
33. The Microsoft 2016 conversational speech recognition system / W. Xiong [и др.] // Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on. — IEEE. 2017. — С. 5255—5259.
34. Weigend A. On overfitting and the effective number of hidden units // Proceedings of the 1993 connectionist models summer school. Т. 1. — 1994. — С. 335—342.
35. Yan Z.-J., Huo Q., Xu J. A scalable approach to using DNN-derived features in GMM-HMM based acoustic modeling for LVCSR. // Interspeech. — 2013. — С. 104—108.
36. Zaremba W., Sutskever I. Learning to execute // arXiv preprint arXiv:1410.4615. — 2014.
37. Hochreiter S., Schmidhuber J. Long short-term memory // Neural computation. — 1997. — Т. 9, № 8. — С. 1735—1780.
38. D. Yu and L. Deng, Automatic Speech Recognition, London, U.K.:Springer, 2016.

39. B. Liao, Y. Ali, S. Nazir, L. He and H. U. Khan, "Security analysis of IoT devices by using mobile computing: A systematic literature review", IEEE Access, vol. 8, pp. 120331-120350, 2020.
40. G. Pironkov, S. U. Wood and S. Dupont, "Hybrid-task learning for robust automatic speech recognition", Comput. Speech Lang., vol. 64, Nov. 2020.
41. Y. Liu and K. Kirchhoff, "Graph-based semisupervised learning for acoustic modeling in automatic speech recognition", IEEE/ACM Trans. Audio Speech Language Process., vol. 24, no. 11, pp. 1946-1956, Nov. 2016.
42. Scalable modified Kneser-Ney language model estimation / K. Heafield // Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). T. 2. — 2013. — C. 690— 696.
43. Show and tell: A neural image caption generator / O. Vinyals // Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on. — IEEE. 2015. — C. 3156— 3164.
44. Simonyan K., Zisserman A. Very deep convolutional networks for large-scale image recognition // arXiv preprint arXiv:1409.1556. — 2014.
45. State-of-the-art speech recognition with sequence-to-sequence models / C.-C. Chiu // arXiv preprint arXiv:1712.01769. — 2017.

Додаток А

Лістинг програми

AndroidManifest.xml //файл описує важливу інформацію про інструменти збірки

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.diploma">
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
   <application
        android:label="diploma"
        android:name="${applicationName}"
        android:icon="@mipmap/ic_launcher">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:launchMode="singleTop"
            android:theme="@style/LaunchTheme"

android:configChanges="orientation|keyboardHidden|keyboard|screenSize|smallestScreenSize|locale|layout
Direction|fontScale|screenLayout|density|uiMode"
            android:hardwareAccelerated="true"
            android:windowSoftInputMode="adjustResize">
            <!-- Specifies an Android theme to apply to this Activity as soon as
                 the Android process has started. This theme is visible to the user
                 while the Flutter UI initializes. After that, this theme continues
                 to determine the Window background behind the Flutter UI. -->
            <meta-data
              android:name="io.flutter.embedding.android.NormalTheme"
              android:resource="@style/NormalTheme"
              />
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <!-- Don't delete the meta-data below.
             This is used by the Flutter tool to generate GeneratedPluginRegistrant.java -->
        <meta-data
            android:name="flutterEmbedding"
            android:value="2" />
    </application>
</manifest>
```

main.dart

```
import 'package:flutter/material.dart';
import 'package:diploma/presentation/home_view.dart';
void main() => runApp(const HomeView());
```

home_view.dart

```
import 'dart:async';
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:flutter_deep_speech/flutter_deep_speech.dart';
```

```
import 'package:path_provider/path_provider.dart';

class HomeView extends StatefulWidget {
  const HomeView({Key? key}) : super(key: key);

  @override
  State<HomeView> createState() => _HomeViewState();
}

class _HomeViewState extends State<HomeView> {
  static const tfliteName = 'uk.tflite';
  static const scorerName = 'kenlm.scorer';

  final FlutterDeepSpeech deepSpeech = FlutterDeepSpeech();

  String text = 'Натисність і утримуйте, щоби почати запис.';
  bool modelLoaded = false;
  bool isListening = false;

  @override
  void initState() {
    super.initState();
    loadModel();
  }

  @override
  void dispose() {
    deepSpeech.dispose();
    super.dispose();
  }

  Future<void> loadModel() async {
    await copyModelsToExternalFilesDir();

    modelLoaded = await deepSpeech.loadModelFromName(
      modelName: 'uk.tflite',
      scorerName: 'kenlm.scorer',
    );
    debugPrint('MODEL LOADED: $modelLoaded');
  }

  Future<void> copyModelsToExternalFilesDir() async {
    final externalFilesDirPath = (await getExternalStorageDirectory())!.path;

    final tfliteDestination = '$externalFilesDirPath/$tfliteName';
    if (!File(tfliteDestination).existsSync()) {
      await copyBytesFromAsset('assets/models/$tfliteName', tfliteDestination);
    }

    final scorerDestination = '$externalFilesDirPath/$scorerName';
    if (!File(scorerDestination).existsSync()) {
      await copyBytesFromAsset('assets/models/$scorerName', scorerDestination);
```

```
  }
}

Future<void> copyBytesFromAsset(String source, String dest) async {
  final bytes = await rootBundle.load(source);
  final file = await File(dest).writeAsBytes(bytes.buffer.asUint8List());
  debugPrint('Copied $source to ${file.path}');
}

Future<void> start() async {
  if (!modelLoaded) {
    debugPrint('ERROR: Model has not been loaded yet!');
    return;
  }

  debugPrint('START LISTENING...');
  setState(() {
    text = 'Я слухаю...';
    isListening = true;
  });

  try {
    await deepSpeech.listen(
      onError: (error) {
        debugPrint('ERROR: ${error.error}');
        debugPrint('STACKTRACE: ${error.stackTrace}');
      },
      onResult: (result) {
        setState(() => text = result.text);
      },
    );
  } on MicPermissionDeniedException catch (e) {
    debugPrint('Error: ${e.toString()}');
    setState(() {
      text = 'Дозвіл не надано!';
      isListening = false;
    });
  }
}

Future<void> stop() async {
  debugPrint('STOP LISTENING...');
  setState(() => isListening = false);
  await deepSpeech.stop();
}

@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Container(
      decoration: const BoxDecoration(
          gradient: LinearGradient(
```

```
        begin: Alignment.topCenter,
        end: Alignment.bottomCenter,
        colors: [
       Color(0xffa13eb4),
       Color(0xff4c118c),
     ])),
child: Scaffold(
  backgroundColor: Colors.transparent,
  body: Center(
    child: SafeArea(
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.center,
        children: [
          Padding(
            padding: const EdgeInsets.all(20.0),
            child: Text(
              text,
              style: const TextStyle(
                fontSize: 24.0,
                color: Colors.white,
                fontWeight: FontWeight.w500
              ),
            ),
          ),
          const Spacer(),
          Listener(
            onPointerUp: (_) => stop(),
            onPointerDown: (_) => start(),
            onPointerCancel: (_) => stop(),
            child: Container(
              width: 100.0,
              height: 100.0,
              decoration: BoxDecoration(
                border: Border.all(
                  color: Colors.white,
                  width: 2,
                ),
                color: isListening ? Colors.white : Colors.transparent,
                shape: BoxShape.circle,
              ),
              child: Center(
                child: Text(
                  isListening ? 'ЗУПИНИТИ' : 'РОЗПОЧАТИ',
                  style: TextStyle(
                    color: isListening ? Colors.black : Colors.white,
                    fontSize: 24.0,
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ),
            ),
          ),
        ),
```

```
                const SizedBox(height: 32.0),
              ],
            ),
          ),
        ),
      ),
    ),
  );
 }
}
```

voice_recognition.dart

```
export 'src/voice_recognition_error.dart';
export 'src/voice_recognition_result.dart';
export 'src/voice_recognition.dart';
```

src/voice_recognition.dart

```
import 'dart:async';

import 'package:flutter/services.dart';

import 'package:diploma/voice_recognition/src/voice_recognition_error.dart';
import 'package:diploma/voice_recognition/src/voice_recognition_result.dart';

typedef ResultCallback = void Function(VoiceRecognitionResult);
typedef ErrorCallback = void Function(VoiceRecognitionError);

class VoiceRecognition {
 static const MethodChannel _methodChannel = MethodChannel(
  'dplm/voice_recognition',
 );

 static const EventChannel _listenEventChannel = EventChannel(
  'dplm/voice_recognition/listen',
 );

 StreamSubscription? _listenEventSub;

 bool _modelLoaded = false;

 bool _hasMicroPermission = false;

 bool _isListening = false;

 Future<bool> loadModelFromName({
  required String modelName,
  required String scorerName,
 }) async {
  if (_modelLoaded) return true;

  final params = {
```

```
    'modelName': modelName,
    'scorerName': scorerName,
  };

  return _modelLoaded =
      await _methodChannel.invokeMethod<bool>('loadModelFromName', params) ??
          false;
}

Future<bool> requestMicPermission() async {
  return _hasMicroPermission =
  await _methodChannel.invokeMethod('requestMicPermission');
}

Future<void> listen({
  void Function(VoiceRecognitionError)? onError,
  void Function(VoiceRecognitionResult)? onResult,
}) async {
  if (!_modelLoaded || _isListening) return;

  if (!_hasMicroPermission) {
    await requestMicPermission();
    if (!_hasMicroPermission) {
      throw const MicPermissionDeniedException(
        'Microphone permission denied!',
      );
    }
  }

  _listenEventSub = _listenEventChannel
      .receiveBroadcastStream()
      .distinct()
      .map(_parseResult)
      .listen(
    onResult,
    onDone: stop,
    onError: (error, stackTrace) {
      onError?.call(VoiceRecognitionError(error: error, stackTrace: stackTrace));
    },
  );

  _isListening = true;
  await _methodChannel.invokeMethod('listen');
}

VoiceRecognitionResult _parseResult(dynamic result) {
  return VoiceRecognitionResult.fromMap(Map<String, dynamic>.from(result));
}

Future<void> stop() async {
  if (!_modelLoaded || !_isListening) return;
```

```
    _isListening = false;
    _listenEventSub?.cancel();
    await _methodChannel.invokeMethod('stop');
  }

  Future<void> dispose() async {
    if (!_modelLoaded) return;

    if (_isListening) await stop();
    _modelLoaded = false;
    await _methodChannel.invokeMethod('dispose');
  }
}

class MicPermissionDeniedException implements Exception {
  final String message;

  const MicPermissionDeniedException(this.message);

  @override
  String toString() => message;
}

voice_recognition_error.dart
class VoiceRecognitionError {
  final Object error;
  final StackTrace stackTrace;

  VoiceRecognitionError({
    required this.error,
    required this.stackTrace,
  });
}

voice_recognition_result.dart
class VoiceRecognitionResult {
  final String text;
  final bool isFinal;

  VoiceRecognitionResult({
    required this.text,
    required this.isFinal,
  });

  factory VoiceRecognitionResult.fromMap(Map<String, dynamic> map) {
    return VoiceRecognitionResult(
      text: map['text'] ?? '',
      isFinal: map['isFinal'] ?? false,
    );
  }
}
```

widget_test.dart

```
import 'package:diploma/presentation/home_view.dart';
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';

import 'package:diploma/main.dart';

void main() {
  testWidgets('Counter increments smoke test', (WidgetTester tester) async {
    // Build our app and trigger a frame.
    await tester.pumpWidget(const HomeView());

    // Verify that our counter starts at 0.
    expect(find.text('0'), findsOneWidget);
    expect(find.text('1'), findsNothing);

    // Tap the '+' icon and trigger a frame.
    await tester.tap(find.byIcon(Icons.add));
    await tester.pump();

    // Verify that our counter has incremented.
    expect(find.text('0'), findsNothing);
    expect(find.text('1'), findsOneWidget);
  });
}
```

pubspec.yaml

```
name: diploma
description: A new Flutter project.

publish_to: 'none'
version: 1.0.0+1

environment:
  sdk: '>=2.18.2 <3.0.0'


dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^1.0.2
  path_provider: ^2.0.11

dev_dependencies:
  flutter_test:
    sdk: flutter

  flutter_lints: ^2.0.0
```

```
flutter:

  uses-material-design: true

  assets:
    - assets/models/
```

AppFramewordInfo.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>CFBundleDevelopmentRegion</key>
  <string>en</string>
  <key>CFBundleExecutable</key>
  <string>App</string>
  <key>CFBundleIdentifier</key>
  <string>io.flutter.flutter.app</string>
  <key>CFBundleInfoDictionaryVersion</key>
  <string>6.0</string>
  <key>CFBundleName</key>
  <string>App</string>
  <key>CFBundlePackageType</key>
  <string>FMWK</string>
  <key>CFBundleShortVersionString</key>
  <string>1.0</string>
  <key>CFBundleSignature</key>
  <string>????</string>
  <key>CFBundleVersion</key>
  <string>1.0</string>
  <key>MinimumOSVersion</key>
  <string>11.0</string>
</dict>
</plist>
```

engine_flutter_controll.dart

```
import 'dart:ffi';
import 'dart:io';
import 'dart:typed_data';

import 'package:ffi/ffi.dart';

typedef DSVersion = Pointer<Utf8> Function();
typedef DSNativeFreeStr = Void Function(Pointer<Utf8>);
typedef DSFreeStr = void Function(Pointer<Utf8>);
typedef CreateModel = Pointer Function(Pointer<Utf8>);
typedef NativeFreeModel = Void Function(Pointer);
typedef FreeModel = void Function(Pointer);
typedef NativeModelSampleRate = Uint64 Function(Pointer);
typedef ModelSampleRate = int Function(Pointer);
typedef NativeSpeechToText = Pointer<Utf8> Function(Pointer, Pointer<Uint8>, Uint64);
```

```
typedef SpeechToText = Pointer<Utf8> Function(Pointer, Pointer<Uint8>, int);
typedef NativeEnableScorer = Int32 Function(Pointer, Pointer<Utf8>);
typedef EnableScorer = int Function(Pointer, Pointer<Utf8>);
typedef NativeDisableScorer = Int32 Function(Pointer);
typedef DisableScorer = int Function(Pointer);
typedef NativeSetScorerAlphaBeta = Int32 Function(Pointer, Float, Float);
typedef SetScorerAlphaBeta = int Function(Pointer, double, double);

class DeepspeechFlutter {
  factory DeepspeechFlutter() => _instance;
  static final DeepspeechFlutter _instance = DeepspeechFlutter._internal();

  DeepspeechFlutter._internal() {
    _deepspeech = Platform.isAndroid ? DynamicLibrary.open("libdeepspeechlibc.so") :
DynamicLibrary.process();

    _dsVersion = _deepspeech.lookupFunction<DSVersion, DSVersion>('deepspeech_verison');
    _dsFreeStr = _deepspeech.lookupFunction<DSNativeFreeStr, DSFreeStr>('deepspeech_free_str');
    _dsCreateModel = _deepspeech.lookupFunction<CreateModel, CreateModel>('create_model');
    _dsFreeModel = _deepspeech.lookupFunction<NativeFreeModel, FreeModel>('free_model');
    _dsModelSampleRate = _deepspeech.lookupFunction<NativeModelSampleRate,
ModelSampleRate>('model_sample_rate');
    _dsSpeechToText = _deepspeech.lookupFunction<NativeSpeechToText,
SpeechToText>('speech_to_text');
    _dsEnableScorer = _deepspeech.lookupFunction<NativeEnableScorer,
EnableScorer>('enable_external_scorer');
    _dsDisableScorer = _deepspeech.lookupFunction<NativeDisableScorer,
DisableScorer>('disable_external_scorer');
    _dsSetScorerAlphaBeta =
        _deepspeech.lookupFunction<NativeSetScorerAlphaBeta,
SetScorerAlphaBeta>('set_scorer_alpha_beta');
  }

  late final DynamicLibrary _deepspeech;

  // Reference to functions.
  late final DSVersion _dsVersion;
  late final DSFreeStr _dsFreeStr;
  late final CreateModel _dsCreateModel;
  late final FreeModel _dsFreeModel;
  late final ModelSampleRate _dsModelSampleRate;
  late final SpeechToText _dsSpeechToText;
  late final EnableScorer _dsEnableScorer;
  late final DisableScorer _dsDisableScorer;
  late final SetScorerAlphaBeta _dsSetScorerAlphaBeta;

  // Pointer to loaded model state
  Pointer? _modelCtxPointer;

  String getVersion() {
    Pointer<Utf8> _version = _dsVersion();
    String value = _version.toDartString();
```

```
  _dsFreeStr(_version);
  return value;
}

void createModel(String modelPath) {
  Pointer<Utf8> _modelPath = modelPath.toNativeUtf8();
  _modelCtxPointer = _dsCreateModel(_modelPath);
  print('_modelCtxPointer: $_modelCtxPointer');
}

int getSampleRate() {
  if (_modelCtxPointer == null || _modelCtxPointer == nullptr) {
    return -1;
  }

  int _sampleRate = _dsModelSampleRate(_modelCtxPointer!);
  return _sampleRate;
}

String speechToText(Uint8List samples) {
  Pointer<Uint8> samplePointer = calloc.call<Uint8>(samples.length);
  for (int index = 0; index < samples.length; index++) {
    samplePointer.elementAt(index).value = samples[index];
  }

  Pointer<Utf8> _result = _dsSpeechToText(_modelCtxPointer!, samplePointer, samples.length);
  malloc.free(samplePointer);

  return _result.toDartString();
}

int enableExternalScorer(String scorerFilePath) {
  Pointer<Utf8> _path = scorerFilePath.toNativeUtf8();
  if (_modelCtxPointer == null || _modelCtxPointer == nullptr) {
    return -1;
  }

  int statusCode = _dsEnableScorer(_modelCtxPointer!, _path);
  return statusCode;
}

int disableExternalScorer() {
  if (_modelCtxPointer == null || _modelCtxPointer == nullptr) {
    return -1;
  }

  int statusCode = _dsDisableScorer(_modelCtxPointer!);
  return statusCode;
}

int setScorerAlphaBeta(double alpha, double beta) {
  if (_modelCtxPointer == null || _modelCtxPointer == nullptr) {
```

```
      return -1;
    }

    int statusCode = _dsSetScorerAlphaBeta(_modelCtxPointer!, alpha, beta);
    return statusCode;
  }
}
```

voice_libc.dart

```
#include <stdlib.h>
#include <string>

// Deepspeech 0.9.3 library header file.
#include "../libdeepspeech_0.9.3/deepspeech.h"

// This library header file.
#include "libc_deepspeech.h"

char *deepspeech_verison(void)
{
    char *version = DS_Version();
    return version;
}

void deepspeech_free_str(char *string)
{
    if (string != NULL)
    {
        DS_FreeString(string);
    }
}

void *create_model(char *model_path)
{
    if (model_path == NULL)
    {
        return NULL;
    }

    ModelState *ctx;
    int status = DS_CreateModel(model_path, &ctx);

    return (void *)ctx;
}

void free_model(void *model_state)
{
    if (model_state != NULL)
    {
        ModelState *ptr = (ModelState *)model_state;
        DS_FreeModel(ptr);
    }
```

```
}

uint64_t model_sample_rate(void *model_state)
{
    if (model_state == NULL)
    {
        return 0;
    }

    ModelState *ptr = (ModelState *)model_state;
    int sample_rate = DS_GetModelSampleRate(ptr);
    return sample_rate;
}

char *speech_to_text(void *model_state, char *buffer, uint64_t buffer_size)
{
    Metadata *result = DS_SpeechToTextWithMetadata((ModelState *)model_state, (short *)buffer,
buffer_size / 2, 3);

    const CandidateTranscript *transcript = &result->transcripts[0];
    std::string retval = "";
    for (int i = 0; i < transcript->num_tokens; i++)
    {
        const TokenMetadata &token = transcript->tokens[i];
        retval += token.text;
    }
    char *encoded = strdup(retval.c_str());

    DS_FreeMetadata(result);

    return encoded;
}

int enable_external_scorer(void *model_state, char *model_path)
{
    ModelState *ptr = (ModelState *)model_state;
    int status = DS_EnableExternalScorer(ptr, model_path);
    return status;
}

int disable_external_scorer(void *model_state)
{
    ModelState *ptr = (ModelState *)model_state;
    int status = DS_DisableExternalScorer(ptr);
    return status;
}

int set_scorer_alpha_beta(void *model_state, float alpha, float beta)
{
    ModelState *ptr = (ModelState *)model_state;
    int status = DS_SetScorerAlphaBeta(ptr, alpha, beta);
    return status;
```

```
}
```

voice_ua.kt

```
package com.example.deepspeech_flutter

import androidx.annotation.NonNull

import io.flutter.embedding.engine.plugins.FlutterPlugin
import io.flutter.plugin.common.MethodCall
import io.flutter.plugin.common.MethodChannel
import io.flutter.plugin.common.MethodChannel.MethodCallHandler
import io.flutter.plugin.common.MethodChannel.Result
import io.flutter.plugin.common.PluginRegistry.Registrar

/** DeepspeechFlutterPlugin */
class DeepspeechFlutterPlugin: FlutterPlugin, MethodCallHandler {
  /// The MethodChannel that will the communication between Flutter and native Android
  ///
  /// This local reference serves to register the plugin with the Flutter Engine and unregister it
  /// when the Flutter Engine is detached from the Activity
  private lateinit var channel : MethodChannel

  override fun onAttachedToEngine(@NonNull flutterPluginBinding: FlutterPlugin.FlutterPluginBinding) {
    channel = MethodChannel(flutterPluginBinding.binaryMessenger, "deepspeech_flutter")
    channel.setMethodCallHandler(this)
  }

  override fun onMethodCall(@NonNull call: MethodCall, @NonNull result: Result) {
    if (call.method == "getPlatformVersion") {
      result.success("Android ${android.os.Build.VERSION.RELEASE}")
    } else {
      result.notImplemented()
    }
  }

  override fun onDetachedFromEngine(@NonNull binding: FlutterPlugin.FlutterPluginBinding) {
    channel.setMethodCallHandler(null)
  }
}
```

deepspeech.h

```
#ifndef DEEPSPEECH_H
#define DEEPSPEECH_H

#ifdef __cplusplus
extern "C" {
#endif

#ifndef SWIG
  #if defined _MSC_VER
    #define DEEPSPEECH_EXPORT __declspec(dllexport)
  #else
```

```
      #define DEEPSPEECH_EXPORT __attribute__ ((visibility("default")))
   #endif /*End of _MSC_VER*/
#else
   #define DEEPSPEECH_EXPORT
#endif

typedef struct ModelState ModelState;

typedef struct StreamingState StreamingState;

/**
 * @brief Stores text of an individual token, along with its timing information
 */
typedef struct TokenMetadata {
  /** The text corresponding to this token */
  const char* const text;

  /** Position of the token in units of 20ms */
  const unsigned int timestep;

  /** Position of the token in seconds */
  const float start_time;
} TokenMetadata;

/**
 * @brief A single transcript computed by the model, including a confidence
 *        value and the metadata for its constituent tokens.
 */
typedef struct CandidateTranscript {
  /** Array of TokenMetadata objects */
  const TokenMetadata* const tokens;
  /** Size of the tokens array */
  const unsigned int num_tokens;
  /** Approximated confidence value for this transcript. This is roughly the
   * sum of the acoustic model logit values for each timestep/character that
   * contributed to the creation of this transcript.
   */
  const double confidence;
} CandidateTranscript;

/**
 * @brief An array of CandidateTranscript objects computed by the model.
 */
typedef struct Metadata {
  /** Array of CandidateTranscript objects */
  const CandidateTranscript* const transcripts;
  /** Size of the transcripts array */
  const unsigned int num_transcripts;
} Metadata;

// sphinx-doc: error_code_listing_start
```

```
#define DS_FOR_EACH_ERROR(APPLY) \
  APPLY(DS_ERR_OK,                    0x0000, "No error.") \
  APPLY(DS_ERR_NO_MODEL,              0x1000, "Missing model information.") \
  APPLY(DS_ERR_INVALID_ALPHABET,      0x2000, "Invalid alphabet embedded in model. (Data
corruption?)") \
  APPLY(DS_ERR_INVALID_SHAPE,         0x2001, "Invalid model shape.") \
  APPLY(DS_ERR_INVALID_SCORER,        0x2002, "Invalid scorer file.") \
  APPLY(DS_ERR_MODEL_INCOMPATIBLE,    0x2003, "Incompatible model.") \
  APPLY(DS_ERR_SCORER_NOT_ENABLED,    0x2004, "External scorer is not enabled.") \
  APPLY(DS_ERR_SCORER_UNREADABLE,     0x2005, "Could not read scorer file.") \
  APPLY(DS_ERR_SCORER_INVALID_LM,     0x2006, "Could not recognize language model header
in scorer.") \
  APPLY(DS_ERR_SCORER_NO_TRIE,        0x2007, "Reached end of scorer file before loading
vocabulary trie.") \
  APPLY(DS_ERR_SCORER_INVALID_TRIE,   0x2008, "Invalid magic in trie header.") \
  APPLY(DS_ERR_SCORER_VERSION_MISMATCH, 0x2009, "Scorer file version does not match
expected version.") \
  APPLY(DS_ERR_FAIL_INIT_MMAP,        0x3000, "Failed to initialize memory mapped model.") \
  APPLY(DS_ERR_FAIL_INIT_SESS,        0x3001, "Failed to initialize the session.") \
  APPLY(DS_ERR_FAIL_INTERPRETER,      0x3002, "Interpreter failed.") \
  APPLY(DS_ERR_FAIL_RUN_SESS,         0x3003, "Failed to run the session.") \
  APPLY(DS_ERR_FAIL_CREATE_STREAM,    0x3004, "Error creating the stream.") \
  APPLY(DS_ERR_FAIL_READ_PROTOBUF,    0x3005, "Error reading the proto buffer model file.") \
  APPLY(DS_ERR_FAIL_CREATE_SESS,      0x3006, "Failed to create session.") \
  APPLY(DS_ERR_FAIL_CREATE_MODEL,     0x3007, "Could not allocate model state.") \
  APPLY(DS_ERR_FAIL_INSERT_HOTWORD,   0x3008, "Could not insert hot-word.") \
  APPLY(DS_ERR_FAIL_CLEAR_HOTWORD,    0x3009, "Could not clear hot-words.") \
  APPLY(DS_ERR_FAIL_ERASE_HOTWORD,    0x3010, "Could not erase hot-word.")

// sphinx-doc: error_code_listing_end

enum DeepSpeech_Error_Codes
{
#define DEFINE(NAME, VALUE, DESC) NAME = VALUE,
DS_FOR_EACH_ERROR(DEFINE)
#undef DEFINE
};

/**
 * @brief An object providing an interface to a trained DeepSpeech model.
 *
 * @param aModelPath The path to the frozen model graph.
 * @param[out] retval a ModelState pointer
 *
 * @return Zero on success, non-zero on failure.
 */
DEEPSPEECH_EXPORT
int DS_CreateModel(const char* aModelPath,
                   ModelState** retval);

/**
 * @brief Get beam width value used by the model. If {@link DS_SetModelBeamWidth}
```

```
 *       was not called before, will return the default value loaded from the
 *       model file.
 *
 * @param aCtx A ModelState pointer created with {@link DS_CreateModel}.
 *
 * @return Beam width value used by the model.
 */
DEEPSPEECH_EXPORT
unsigned int DS_GetModelBeamWidth(const ModelState* aCtx);

/**
 * @brief Set beam width value used by the model.
 *
 * @param aCtx A ModelState pointer created with {@link DS_CreateModel}.
 * @param aBeamWidth The beam width used by the model. A larger beam width value
 *                   generates better results at the cost of decoding time.
 *
 * @return Zero on success, non-zero on failure.
 */
DEEPSPEECH_EXPORT
int DS_SetModelBeamWidth(ModelState* aCtx,
                         unsigned int aBeamWidth);

/**
 * @brief Return the sample rate expected by a model.
 *
 * @param aCtx A ModelState pointer created with {@link DS_CreateModel}.
 *
 * @return Sample rate expected by the model for its input.
 */
DEEPSPEECH_EXPORT
int DS_GetModelSampleRate(const ModelState* aCtx);

/**
 * @brief Frees associated resources and destroys model object.
 */
DEEPSPEECH_EXPORT
void DS_FreeModel(ModelState* ctx);

/**
 * @brief Enable decoding using an external scorer.
 *
 * @param aCtx The ModelState pointer for the model being changed.
 * @param aScorerPath The path to the external scorer file.
 *
 * @return Zero on success, non-zero on failure (invalid arguments).
 */
DEEPSPEECH_EXPORT
int DS_EnableExternalScorer(ModelState* aCtx,
                            const char* aScorerPath);

/**
```

```
 * @brief Add a hot-word and its boost.
 *
 * @param aCtx The ModelState pointer for the model being changed.
 * @param word The hot-word.
 * @param boost The boost.
 *
 * @return Zero on success, non-zero on failure (invalid arguments).
 */
DEEPSPEECH_EXPORT
int DS_AddHotWord(ModelState* aCtx,
                  const char* word,
                  float boost);

/**
 * @brief Remove entry for a hot-word from the hot-words map.
 *
 * @param aCtx The ModelState pointer for the model being changed.
 * @param word The hot-word.
 *
 * @return Zero on success, non-zero on failure (invalid arguments).
 */
DEEPSPEECH_EXPORT
int DS_EraseHotWord(ModelState* aCtx,
                    const char* word);

/**
 * @brief Removes all elements from the hot-words map.
 *
 * @param aCtx The ModelState pointer for the model being changed.
 *
 * @return Zero on success, non-zero on failure (invalid arguments).
 */
DEEPSPEECH_EXPORT
int DS_ClearHotWords(ModelState* aCtx);

/**
 * @brief Disable decoding using an external scorer.
 *
 * @param aCtx The ModelState pointer for the model being changed.
 *
 * @return Zero on success, non-zero on failure.
 */
DEEPSPEECH_EXPORT
int DS_DisableExternalScorer(ModelState* aCtx);

/**
 * @brief Set hyperparameters alpha and beta of the external scorer.
 *
 * @param aCtx The ModelState pointer for the model being changed.
 * @param aAlpha The alpha hyperparameter of the decoder. Language model weight.
 * @param aLMBeta The beta hyperparameter of the decoder. Word insertion weight.
 *
```

```
 * @return Zero on success, non-zero on failure.
 */
DEEPSPEECH_EXPORT
int DS_SetScorerAlphaBeta(ModelState* aCtx,
                          float aAlpha,
                          float aBeta);

/**
 * @brief Use the DeepSpeech model to convert speech to text.
 *
 * @param aCtx The ModelState pointer for the model to use.
 * @param aBuffer A 16-bit, mono raw audio signal at the appropriate
 *                sample rate (matching what the model was trained on).
 * @param aBufferSize The number of samples in the audio signal.
 *
 * @return The STT result. The user is responsible for freeing the string using
 *         {@link DS_FreeString()}. Returns NULL on error.
 */
DEEPSPEECH_EXPORT
char* DS_SpeechToText(ModelState* aCtx,
                      const short* aBuffer,
                      unsigned int aBufferSize);

/**
 * @brief Use the DeepSpeech model to convert speech to text and output results
 * including metadata.
 *
 * @param aCtx The ModelState pointer for the model to use.
 * @param aBuffer A 16-bit, mono raw audio signal at the appropriate
 *                sample rate (matching what the model was trained on).
 * @param aBufferSize The number of samples in the audio signal.
 * @param aNumResults The maximum number of CandidateTranscript structs to return. Returned value
might be smaller than this.
 *
 * @return Metadata struct containing multiple CandidateTranscript structs. Each
 *         transcript has per-token metadata including timing information. The
 *         user is responsible for freeing Metadata by calling {@link DS_FreeMetadata()}.
 *         Returns NULL on error.
 */
DEEPSPEECH_EXPORT
Metadata* DS_SpeechToTextWithMetadata(ModelState* aCtx,
                                      const short* aBuffer,
                                      unsigned int aBufferSize,
                                      unsigned int aNumResults);

/**
 * @brief Create a new streaming inference state. The streaming state returned
 *        by this function can then be passed to {@link DS_FeedAudioContent()}
 *        and {@link DS_FinishStream()}.
 *
 * @param aCtx The ModelState pointer for the model to use.
 * @param[out] retval an opaque pointer that represents the streaming state. Can
```

```
 *                 be NULL if an error occurs.
 *
 * @return Zero for success, non-zero on failure.
 */
DEEPSPEECH_EXPORT
int DS_CreateStream(ModelState* aCtx,
                    StreamingState** retval);

/**
 * @brief Feed audio samples to an ongoing streaming inference.
 *
 * @param aSctx A streaming state pointer returned by {@link DS_CreateStream()}.
 * @param aBuffer An array of 16-bit, mono raw audio samples at the
 *                appropriate sample rate (matching what the model was trained on).
 * @param aBufferSize The number of samples in @p aBuffer.
 */
DEEPSPEECH_EXPORT
void DS_FeedAudioContent(StreamingState* aSctx,
                         const short* aBuffer,
                         unsigned int aBufferSize);

/**
 * @brief Compute the intermediate decoding of an ongoing streaming inference.
 *
 * @param aSctx A streaming state pointer returned by {@link DS_CreateStream()}.
 *
 * @return The STT intermediate result. The user is responsible for freeing the
 *         string using {@link DS_FreeString()}.
 */
DEEPSPEECH_EXPORT
char* DS_IntermediateDecode(const StreamingState* aSctx);

/**
 * @brief Compute the intermediate decoding of an ongoing streaming inference,
 *        return results including metadata.
 *
 * @param aSctx A streaming state pointer returned by {@link DS_CreateStream()}.
 * @param aNumResults The number of candidate transcripts to return.
 *
 * @return Metadata struct containing multiple candidate transcripts. Each transcript
 *         has per-token metadata including timing information. The user is
 *         responsible for freeing Metadata by calling {@link DS_FreeMetadata()}.
 *         Returns NULL on error.
 */
DEEPSPEECH_EXPORT
Metadata* DS_IntermediateDecodeWithMetadata(const StreamingState* aSctx,
                                            unsigned int aNumResults);

/**
 * @brief Compute the final decoding of an ongoing streaming inference and return
 *        the result. Signals the end of an ongoing streaming inference.
 *
```

```
 * @param aSctx A streaming state pointer returned by {@link DS_CreateStream()}.
 *
 * @return The STT result. The user is responsible for freeing the string using
 *         {@link DS_FreeString()}.
 *
 * @note This method will free the state pointer (@p aSctx).
 */
DEEPSPEECH_EXPORT
char* DS_FinishStream(StreamingState* aSctx);

/**
 * @brief Compute the final decoding of an ongoing streaming inference and return
 *        results including metadata. Signals the end of an ongoing streaming
 *        inference.
 *
 * @param aSctx A streaming state pointer returned by {@link DS_CreateStream()}.
 * @param aNumResults The number of candidate transcripts to return.
 *
 * @return Metadata struct containing multiple candidate transcripts. Each transcript
 *         has per-token metadata including timing information. The user is
 *         responsible for freeing Metadata by calling {@link DS_FreeMetadata()}.
 *         Returns NULL on error.
 *
 * @note This method will free the state pointer (@p aSctx).
 */
DEEPSPEECH_EXPORT
Metadata* DS_FinishStreamWithMetadata(StreamingState* aSctx,
                                      unsigned int aNumResults);

/**
 * @brief Destroy a streaming state without decoding the computed logits. This
 *        can be used if you no longer need the result of an ongoing streaming
 *        inference and don't want to perform a costly decode operation.
 *
 * @param aSctx A streaming state pointer returned by {@link DS_CreateStream()}.
 *
 * @note This method will free the state pointer (@p aSctx).
 */
DEEPSPEECH_EXPORT
void DS_FreeStream(StreamingState* aSctx);

/**
 * @brief Free memory allocated for metadata information.
 */
DEEPSPEECH_EXPORT
void DS_FreeMetadata(Metadata* m);

/**
 * @brief Free a char* string returned by the DeepSpeech API.
 */
DEEPSPEECH_EXPORT
void DS_FreeString(char* str);
```

```
/**
 * @brief Returns the version of this library. The returned version is a semantic
 *        version (SemVer 2.0.0). The string returned must be freed with {@link DS_FreeString()}.
 *
 * @return The version string.
 */
DEEPSPEECH_EXPORT
char* DS_Version();

/**
 * @brief Returns a textual description corresponding to an error code.
 *        The string returned must be freed with @{link DS_FreeString()}.
 *
 * @return The error description.
 */
DEEPSPEECH_EXPORT
char* DS_ErrorCodeToErrorMessage(int aErrorCode);

#undef DEEPSPEECH_EXPORT

#ifdef __cplusplus
}
#endif

#endif /* DEEPSPEECH_H */

flutter_service.dart
class FlutterService extends StatefulWidget {
  @override
  _ FlutterService createState() => _ FlutterService ();
}

class _ FlutterService extends State< FlutterService > {
  final _deepspeech = DeepspeechFlutter();
  int _sampleRate = 0;
  String? _processedText;
  Uint8List? _wavFile;
  bool _wavFileLoaded = false;

  @override
  void initState() {
    super.initState();
    _loadModel();
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Mozilla DeepSpeech Example'),
```

```
      ),
      body: Center(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            SizedBox(height: 40),
            Text('DeepSpeech Version: ${_deepspeech.getVersion()}',
                style: TextStyle(fontSize: 20)),
            SizedBox(height: 40),
            Text('Model Sample Rate: $_sampleRate',
                style: TextStyle(fontSize: 20)),
            SizedBox(height: 40),
            OutlinedButton(
              child: Text('Load WAV File'),
              onPressed: _loadWavFile,
            ),
            SizedBox(height: 40),
            if (_wavFileLoaded) ...[
              Text('Loaded: new-home-in-the-stars-16k.wav'),
              SizedBox(height: 10),
              OutlinedButton(
                child: Text('Run Speech To Text'),
                onPressed: _runSpeechToText,
              ),
            ],
            SizedBox(height: 40),
            if (_processedText != null) ...[
              Text(
                'Converted Speech to Text:',
                style: TextStyle(fontSize: 20),
              ),
              // SizedBox(height: 30),
              Padding(
                padding: const EdgeInsets.all(20.0),
                child: Text(_processedText!,
                    style:
                        TextStyle(fontSize: 30, fontStyle: FontStyle.italic)),
              ),
            ],
          ],
        ),
      ),
    ),
  );
}

// Load deepspeech english pre-trained model.
Future<void> _loadModel() async {
  final bytes =
      await rootBundle.load('assets/deepspeech-0.9.3-models.tflite');
  final directory = (await getApplicationDocumentsDirectory()).path;
  final buffer = bytes.buffer;
```

```
    final path = '$directory/deepspeech-0.9.3-models.tflite';
    await File(path).writeAsBytes(
        buffer.asUint8List(bytes.offsetInBytes, bytes.lengthInBytes));

    _deepspeech.createModel(path);

    setState(() {
      _sampleRate = _deepspeech.getSampleRate();
    });
  }

  Future<void> _loadWavFile() async {
    final bytes = await rootBundle.load('assets/new-home-in-the-stars-16k.wav');
    _wavFile = bytes.buffer.asUint8List();

    setState(() {
      _wavFileLoaded = true;
    });
  }

  void _runSpeechToText() async {
    if (_wavFile != null) {
      final _result = _deepspeech.speechToText(_wavFile!);

      setState(() {
        _processedText = _result;
      });
    }
  }
}
```

Додаток Б

ВІДГУК
наукового керівника

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

Факультет інформаційних технологій
Кафедра програмного забезпечення комп'ютерних систем

ВІДГУК

Наукового керівника _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада, місце роботи)

На кваліфікаційну роботу

студента _____
(прізвище, ім'я, по батькові)

курсу II групи _____
спеціальності _____
на тему _____

Актуальність теми _____

Мета досліджень _____

Коротка характеристика розділів роботи _____

Практичне значення роботи _____

Зауваження та недоліки _____

Висновки та оцінка _____

Науковий керівник _____
(прізвище, ім’я, по батькові, посада, місце роботи)

«___»_____20__ р. _____
(підпис)

Додаток В

РЕЦЕНЗІЯ

РЕЦЕНЗІЯ
на кваліфікаційну роботу

студента _____
(прізвище, ім'я, по батькові)

курсу II групи _____
кафедри програмного забезпечення комп'ютерних систем
спеціальності _____
Тема роботи _____

Стисла характеристика розділів роботи _____

Пропозиції, внесені студентом, рівень їх наукового обґрунтування _____

Практичне значення роботи _____

Якість оформлення роботи _____

Недоліки в роботі _____

Загальний висновок _____
(підготовленість студента до самостійної роботи як спеціаліста)

Оцінка магістерської роботи _____

Рецензент _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада, місце роботи)

«___»_____20__ р. _____
(підпис)

Додаток Г

АКТ ВПРОВАДЖЕННЯ

ЗАТВЕРДЖУЮ
Начальник ЦІТ і ТЗ
ДД ВАТ “Укртелеком”
_____Коваленко О.А.
“_____”_____2018р.

А К Т

впровадження в дослідно-промислову експлуатацію програмної підсистеми “Обробка даних про платежі, що надходять (перераховуються) на прибутковий рахунок ДД ВАТ “Укртелеком”

Ми, що підписалися нижче, члени комісії у складі:

Начальник сектора ЦІТ і ТЗ	Губар С.І.
Провідний інженер зв’язку	Часник Т.Г.
Провідний економіст ЦИТ і ТЗ	Топча Л.В.

Склали цей акт про те, що в березні 2018 р. студентка Смірнова А.М. впровадила програмну підсистему “Обробка даних про платежі, що надходять (перераховуються) на прибутковий рахунок ДД ВАТ “Укртелеком”.

Означена підсистема входить до комплексу програм інформаційної системи “Обробка даних платників” та призначена для автоматизації процесу обробки інформації зі структурних підрозділів ДД ВАТ “Укртелеком”, що підвищує продуктивність та поліпшує якість роботи економістів.

Протягом березня 2018 р. підсистема була впроваджена та протестована в економічному відділі ЦІТ і ТЗ, а також проведене навчання користувачів, перевірена відповідність вимогам технічного завдання, одержані вихідні форми згідно технічного завдання.

Рішення комісії:

1. Вважати програмну підсистему “Обробка даних про платежі, що надходять (перераховуються) на прибутковий рахунок ДД ВАТ “Укртелеком” прийнятою в дослідно-промислову експлуатацію.

Начальник сектора ЦІТ іТЗ	Губар С.І.
Провідний інженер зв’язку	Часник Т.Г.
Провідний економіст ЦИТ і ТЗ	Топча Л.В.

Додаток Д

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім'я файла	**Опис**
Пояснювальні документи	
Диплом_Яковлєва.doc	Пояснювальна записка кваліфікаційної роботи. Документ Word.
Диплом_Яковлєва.pdf	Пояснювальна записка кваліфікаційної роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_Яковлєва.ppt	Презентація до магістерської роботи