

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Інститут Прикладного Системного Аналізу

Кафедра Системного Проектування

До захисту допущено:

Завідувач кафедри

Петренко А.І.

«__»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Інтелектуальні сервіс-орієнтовані
розподілені обчислювання»**

спеціальності 122 «Комп'ютерні науки»

**на тему: «Розробка системи автоматичного розпізнавання українського
мовлення»**

Виконав:

студент IV курсу, групи ДА-72

Самвелян Артур Рафаельович _____

Керівник:

асистент, аспірант

Шаптала Роман Віталійович _____

Консультант з економічного розділу:

Кандидат економічних наук, доцент

Рощина Надія Василівна _____

Рецензент:

доцент, к.т.н.

Дідковська Марина Віталіївна _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів без
відповідних посилань.

Студент _____

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра Системного проектування

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Інтелектуальні сервіс-орієнтовані
розподілені обчислювання»

ЗАТВЕРДЖУЮ

Завідувач кафедри

А.І.Петренко

«___» _____ 20__ р.

ЗАВДАННЯ

на дипломну роботу студенту Самвеляну Артуру Рафаельовичу

1. Тема роботи «Розробка системи автоматичного розпізнавання українського мовлення», керівник роботи Шаптала Роман Віталійович, асистент, затверджені наказом по університету від «___» _____ 20__ р. No _____

2. Термін подання студентом роботи 7 червня 2021 р.

3. Вихідні дані до роботи

Мобільний додаток, повинен забезпечувати:

4. Зміст роботи:

1. Вступ.
2. Аналіз предметної області.
3. Аналіз сучасних рішень у сфері автоматичного розпізнавання мовлення.
4. Розробка власної системи розпізнавання мовлення.
5. Функціонально-вартісний аналіз програмного продукту.

6. Висновки.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо):

1. Презентація до захисту роботи.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Економічний	Рощина Н.В.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	10.02.2021	
2	Аналіз сучасних архітектур для задачі розпізнавання мовлення.	20.03.2021	
3	Дослідження архітектур які є оптимальними у відношенні час навчання/якість. Проведення порівняльного аналізу.	02.04.2021	
4	Зібрати транскрибовані аудіозаписи на українській	15.04.2021	

	мові для тренування та тестування моделей.		
5	Тренування моделі на зібраних даних	15.05.2021	
6	Тестування моделі на тестових даних. Порівняння якості та швидкості роботи моделей	25.05.2021	

Студент

Самвелян А.Р.

Керівник

Шаптала Р.В.

АНОТАЦІЯ

бакалаврської дипломної роботи Самвеліяна Артура Рафаельовича на тему
“Розробка системи автоматичного розпізнавання українського мовлення”

Метою даної роботи була розробка системи автоматичного розпізнавання українського мовлення. Результатом роботи є реалізована система автоматичного розпізнавання українського мовлення для демонстрації можливостей сучасних технологій розпізнавання мовлення.

Дипломна робота включає в себе розгляд сучасних архітектур систем автоматичного розпізнавання мовлення та історію їх розвитку. В ході виконання було сформовано опис вибору застосовуваних технологій, аргументовано рішення щодо формату розроблюваної системи та наведено детальний опис процесу її створення.

Оскільки задача автоматичного розпізнавання мовлення є надзвичайно актуальною, і актуальність її все збільшується з часом, розроблена система автоматичного розпізнавання українського мовлення має забезпечити попереднє ознайомлення з сучасними можливостями технологій у сфері аналізу та обробки мовлення та як програма полегшити перетворення аудіо даних в текст для подальшої обробки.

Загальний обсяг роботи: 68 сторінок, 18 рисунків, 10 таблиць, 17 посилань.

Ключові слова: розпізнавання мовлення, автоматичне розпізнавання мовлення, нейронна мережа, Kaldi, Python, Google Platform.

ANNOTATION

for the bachelor's work of Samvelyan Artur Rafaelovych
on "Development of automatic speech recognition system for Ukrainian"

The purpose of this work is to perform a development of an automatic speech recognition system for Ukrainian language. The result of the work is an implemented automatic speech recognition system for demonstrating its accuracy.

This thesis includes reviewing state-of-the-art architectures of automatic speech recognition systems and their development history. During the implementation, a description of the choice of applied technologies was formed, the decision on the format of the developed system was argued and a detailed description of the process of its creation was given.

As the task of automatic speech recognition is extremely important, and its relevance is increasing over time, the developed system of automatic speech recognition should provide a preliminary acquaintance with modern technologies in the field of speech analysis and processing and as a program to facilitate the conversion of audio data into text for further processing.

Total volume of the work: 68 pages, 18 illustrations, 10 tables, 17 links.

Key words: speech recognition, automatic speech recognition, neural network, Kaldi, Python, Google Platform.

ЗМІСТ

ЗМІСТ	7
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ	9
ВСТУП	10
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	12
1.1 Поняття процесу автоматичного розпізнавання мовлення	12
1.2 Застосування автоматичного розпізнавання мовлення	12
1.2.1 Мовленнєва аналітика	12
1.2.2 Застосунки для людей з вадами слуху	13
1.2.3 Додатки для нотування записів голосом	13
1.2.3 Голосові помічники	14
1.3 Загальний алгоритм розпізнавання мовлення	14
1.3.1 Акустична передобробка	16
1.3.2 Мел-кепстральні коефіцієнти	17
1.3.2 Акустична модель	20
1.3.3 Мовна модель	21
1.3.4 N-gram мовна модель	22
1.3.5 Декодер	22
1.3.6 WFST	22
1.4 Математична модель системи АРМ	23
1.5 Метрики точності моделей АРМ	24
1.6 Висновки до розділу	25
2. АНАЛІЗ СУЧАСНИХ РІШЕНЬ У СФЕРІ АРМ	26
2.1 Архітектура HMM-GMM	26
2.2 Архітектура HMM-DNN	27
2.2.1 Нейронні мережі з часовою затримкою	28
2.2.2 Архітектура нейронних мереж з часовою затримкою	29
2.3 Архітектура Silero Speech-to-text	31
2.4 Архітектура wav2vec	32
2.5 Архітектура Deep Speech	33
2.6 Висновки до розділу	34

3. РОЗРОБКА ВЛАСНОЇ СИСТЕМИ АРМ	35
3.1 Вибір фреймворку розробки моделі АРМ	35
3.2 Опис фреймворку Kaldi	35
3.3 Вибір та опис “рецепту” моделі для АРМ	36
3.3.1 Алгоритм тренування моделей за допомогою рецепту mini_librispeech	36
3.4 Тренування моделі	38
3.5 Вибір даних для тестування моделі	38
3.6 Вибір існуючих моделей для порівняння та вимірювання їх якості	39
3.7 Вимірювання якості моделей та їх порівняння	39
4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ	41
4.1 Постановка задачі проектування	41
ВИСНОВКИ	63
ПЕРЕЛІК ПОСИЛАНЬ	65
ДОДАТОК А	68
ДОДАТОК Б	69
ДОДАТОК В	73
ДОДАТОК Г	79

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

APM - автоматичне розпізнавання мовлення

RNN - рекурентна нейронна мережа

TDNN - нейронна мережа з часовою затримкою

CNN - згорткова нейронна мережа

DTW - алгоритм динамічної трансформації часової шкали

PLP - коефіцієнти перцептивного лінійного передбачення

LPC - лінійне передбачуване кодування

LPCC - кепстральні коефіцієнти лінійного передбачення

MFCC - мел-кепстральні коефіцієнти

CRF - умовне випадкове поле

MLP - мульти шаровий перцептрон

LSTM - нейронна мережа короткотривалої-довготривалої пам'яті

ВСТУП

В наш час взаємодія із технікою стає більш інтерактивною. З року в рік людство шукає шляхи для того щоб зробити цю взаємодію більш зручною для себе. Під цими діями виникають такі способи взаємодії як голосове управління, в якому невід'ємною складовою є автоматичне розпізнавання мовлення наряду з синтезом мовлення та розумінням тексту.

Оскільки людям набагато зручніше вести діалоги та робити пошуки за допомогою голосу, методи рішення даної задачі здійснили довгий шлях: від статистичних методів, до нейромережевих, і до сих пір цей розвиток продовжується.

Першим хто почав займатися проблемою розпізнавання неперервного мовлення був Редж Редді (Raj Reddy), студент Стенфордського університету, який вирішив взяти задачу розпізнавання мовлення як кваліфікаційну роботу у 1960 році. В той час коли попередні системи обробки неперервного мовлення потребували наявності явної паузи у мовленні, система Реджа автоматично визначала дані паузи, що робило її більш універсальною на той момент.

Приблизно в той же час радянські вчені винайшли алгоритм динамічної трансформації цифрової шкали (DTW) і використали його для створення своєї системи автоматичного розпізнавання мовлення. DTW обробляв мовлення, поділяючи його на короткі кадри, напр. 10 мс сегментів, і обробка кожного кадру як єдиної одиниці. Хоча DTW буде замінено пізнішими алгоритмами, дана методика збереглася до наших днів. Хоча досягнення незалежності спікера на той час залишалося невирішеним.

Зараз існує достатньо різні рішення даної задачі: від гібридних моделей, до чисто нейромережевих. Такі великі компанії як Facebook, Google, Amazon займаються постійними дослідженнями у сфері обробки та розпізнавання мовлення та постійно пропонують нові архітектури та підходи до даної задачі.

Не зважаючи на різкий прогрес у сфері обробки та розпізнавання мовлення, до сих пір є невирішені проблеми таких систем. Однією з головних проблем є проблема слова поза словника. Дана проблема виникає коли спікер або диктор промовляє слово яке мовна модель ніколи не бачила, тому в результаті буде виведене інше слово, яке, на думку мовної моделі, найкраще описує мовний сигнал.

Дані наявні проблеми і недоліки в сфері автоматичного розпізнавання мовлення роблять її цікавою і перспективною для користувачів та розробників зі всього світу. Тому на даний момент аналіз та розробка моделі для автоматичного розпізнавання мовлення є доцільною, як ніколи.

Результуюча модель зможе знайти своє практичне застосування в різних сферах і закріпити результат який люди змогли досягти на сьогодні.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття процесу автоматичного розпізнавання мовлення

Автоматичне розпізнавання мовлення - це процес перетворення мовленнєвого сигналу в цифрову інформацію (в даному випадку в текстові дані).

Девайс, який здійснює дане перетворення називається розпізнавачем мовлення і може бути реалізований за допомогою різних підходів та архітектур.

1.2 Застосування автоматичного розпізнавання мовлення

Процес автоматичного розпізнавання мовлення широко застосовуються у нашому повсякденному житті. Починаючи з років цифрової революції розумні девайси та смартфони стали невід'ємною частиною нашого життя, тому людина постійно придумує запроваджує нові технології які можуть спростити їй життя.

1.2.1 Мовленнєва аналітика

Мовленнєва аналітика є однією з найважливіших завдань бізнесу як в IP-телефонії, звичайній телефонії та дослідженнях. Вона дозволяє ефективно приймати рішення за короткий термін аналізу мовлення. З її допомогою дуже легко знизити відтік існуючих клієнтів і збільшити число нових покупців для бізнесу.

За допомогою системи автоматичного розпізнавання мовлення можна дуже сильно зменшити час який витрачається на проведення мовленнєвої аналітики способом автоматичного перетворення аудіо в текст, який далі може бути проаналізований як людиною так і іншою моделлю машинного навчання.

Також система автоматичного розпізнавання мовлення значно спрощує роботу журналістам та телевізійних каналів. Кожен день

працюючи журналісти змушені проводити інтерв'ю з багатьма людьми, записуючи їх на диктофон або відеокамеру, і прослуховувати їх від початку до кінця виписуючи сказані фрази. З системою АРМ в середньому журналісти економлять 40% свого часу даючи системі розпізнавати мовлення, тим самим звільняють свій час для іншої роботи.

1.2.2 Застосунки для людей з вадами слуху

Технологія автоматичного розпізнавання мовлення дає надію людям з вадами слуху на спрощення повсякденної комунікації. Недочування партнера по розмові, або повна відсутність слуху - в тій чи іншій мірі це реальність з якою люди з вадами слуху стикаються кожен день. Дані ситуації часто змушують людей без слуху уникати соціальних ситуацій в своєму житті що знижує якість їх життя.

Саме технологія автоматичного розпізнавання мовлення є ключем до вирішення багатьох проблем, тільки маючи при собі девайс з даною технологією люди з вадами слуху можуть дати прослуховування напарника по розмові даному девайсу, і в результаті бачити текст сказаного.

1.2.3 Додатки для нотування записів голосом

Додатки для нотування в нашому житті також займають не останнє місце. Кожен день нам приходять до голови ідеї, плани або просто завдання, які ми записуємо в ці додатки. Але бувають ситуації коли фізично людина не може зробити запис в такий додаток, наприклад коли вона знаходиться за кермом або в будь-якій іншій ситуації в якій відволікатися на те щоб щось написати в телефоні немає можливості. І вирішення даних ситуацій також є автоматичне розпізнавання мовлення. Використовуючи тільки голосовий інтерфейс, людина має змогу додавати записи та змінювати існуючі.

1.2.3 Голосові помічники

Автоматичне розпізнавання мовлення займає ключову роль й у технології голосового помічника.

Голосові помічники, або як їх ще називають, віртуальні помічники - це програмні продукти які за допомогою технологій автоматичного розпізнавання мовлення, аналізу та розуміння тексту та синтезу мовлення проводять пряме спілкування з людиною і виконують її команди.

Дані програмні продукти складаються із трьох функціональних частин:

- 1) Розпізнавання мовлення
- 2) Розуміння та аналіз тексту
- 3) Синтез мовлення

При надходженні звукового сигналу до помічника, розпізнавач мовлення проводить перетворення цього сигналу в текст, який потім йде до модулю аналізу та розуміння тексту. Модуль розуміння тексту аналізує вхідний текст і видає якусь відповідь також у вигляді тексту. Вихідний текст йде до модулю синтезу мовлення який вже генерує аудіо сигнал з цього тексту і програвляє його. Дана технологія з роками набуває все більше популярності.

1.3 Загальний алгоритм розпізнавання мовлення

Для вирішення задачі розпізнавання мовлення використовують як гібридні архітектури, які складаються з акустичної моделі та мовної, так і неймережеві або “end-to-end” архітектури.

Типова система розпізнавання мови складається з компонентів, які включають акустичний інтерфейс, акустичну модель, лексику, мовну модель та декодер. Акустичний інтерфейс дбає про перетворення мовного сигналу у відповідний вигляд, що надає корисну інформацію для розпізнавання, наприклад у вектори мел-кепстральних коефіцієнтів, вектору віконного перетворення Фур'є, або інших. Вхідна форма звуку від

мікрофона перетворюється у послідовність фіксованого розміру - акустичні вектори - цей процес називається вилученням ознак. Параметри слова оцінюються на основі акустичних векторів навчальних даних.

Декодер працює шляхом пошуку у всіх можливих послідовностях слів, щоб знайти послідовність слів, які найімовірніше підходять до даного аудіо сигналу. Дана ймовірність складає з себе як акустичну модель $P(O | W)$ так і мовну модель $P(W)$.



Рисунок 1.1 - Загальна архітектура системи АРМ

Більш наглядно процес розпізнавання мовлення ілюструє рисунок 1.2 на якому графічно показано усі складові системи автоматичного розпізнавання мовлення з результатами:

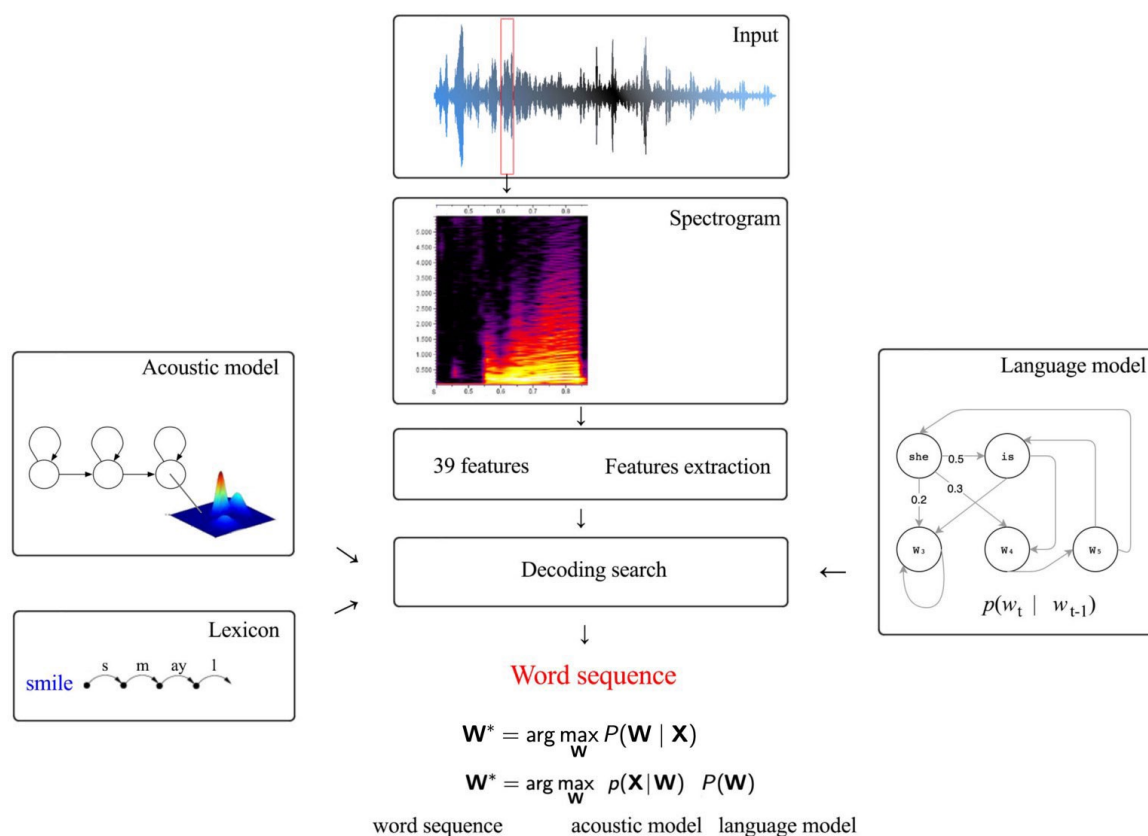


Рисунок 1.2 - Наглядний процес розпізнавання мовлення [14]

1.3.1 Акустична передобробка

Першим шляхом в будь-якій системі розпізнавання мовлення буде вилучення з нього акустичних признаков. Задача полягає в тому, щоб виділити тільки потрібну інформацію з мовного сигналу, яка в подальшому буде використана для процесу розпізнавання, і не виділяти непотрібну інформацію, таку як:

- акцент людини
- характеристики мікрофону та каналу передачі звуку
- фоновий шум, акустику ітп.

Зазвичай більшість методів обробки мовних сигналів при автоматичному розпізнаванню мовлення засновані на Мел-частотних коефіцієнтах (Mel Frequency Cepstral Coefficient - MFCC) і перцептуальних коефіцієнтах лінійного передбачення (Perceptual Linear Prediction - PLP).

Обидва цих методи створюють вектори малої розмірності, які характеризують спектр мовлення на невеликому часовому інтервалі, як правило від 20 до 50 мілісекунд.

1.3.2 Мел-кепстральні коефіцієнти

Основне значення для розуміння мови полягає в тому, що звуки, що утворюються людиною, фільтруються за формою голосового тракту, включаючи язик, зуби тощо. Ця форма визначає, який звук виходить. Якщо ми можемо точно визначити форму, це повинно дати нам точне уявлення про вироблену фонему. Форма голосового тракту проявляється в оболонці короткого часу енергетичного спектру, а завдання MFCC - точно представляти цю оболонку.

Кепстральні коефіцієнти частоти Mel (MFCC) - це ознака, яка широко використовується в автоматичному розпізнаванні мови та звуків. Вони були представлені Девісом і Мермельштейнам у 1980-х роках і з тих пір були найсучаснішим методом для вилучення ознак. До впровадження MFCC існували також коефіцієнти лінійного прогнозування (LPC) та коефіцієнти лінійного прогнозування кепстральних коефіцієнтів (LPCC) і були головним типом функцій для автоматичного розпізнавання мови (ASR), особливо з класифікаторами HMM.

У звуковій обробці Mel частоти (MFC) - це представлення короткочасного спектру потужності звуку, заснованого на лінійному перетворенні косинусу спектру потужності в нелінійній шкалі Mel.

Кепстральні коефіцієнти частоти Мела (MFCC) - це коефіцієнти, які спільно складають кепстральний спектр (MFC). Вони походять від типу кепстрального зображення аудіокліпу (не лінійного "спектру спектру"). Різниця між кепструмом та кеплочастотою мель-частоти полягає в тому, що в MFC смуги частот однаково розташовані за шкалою розтоплення, що наближає реакцію слухової системи людини ближче, ніж до

лінійно-розташованих діапазонів частот, використовуваних у звичайній кепструмі. Це частотна викривлення може забезпечити кращу подання звуку, наприклад, при стисненні звуку.

MFCC зазвичай отримують наступним чином:

- 1) Розбити сигнал на частини (зазвичай продовжністю 25 мс і з переходом в 10 мс)
- 2) Візьміть віконне перетворення Фур'є сигналу від кожної частини.

До кожного фрагменту аудіо сигналу спочатку застосовуємо будь-яку віконну функцію (зазвичай використовується вікно Хеммінга):

$$x[n] = x[n]w[n] \quad (1.1)$$

$$w[n] = 0.54 - 0.46 \cdot \cos\left(\frac{2\pi n}{N-1}\right) \quad (1.2)$$

Графічний вигляд вікна Хеммінга показано на рисунку 1.3:

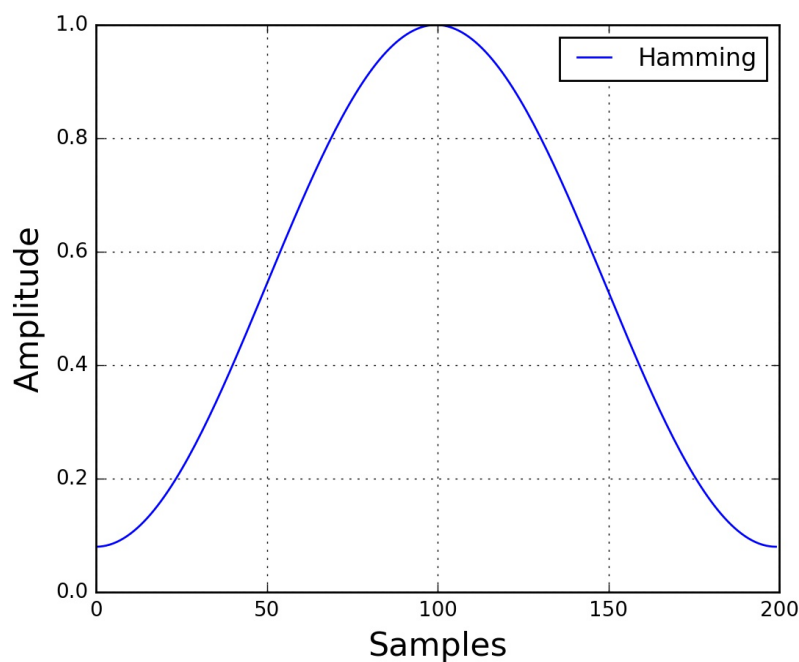


Рисунок 1.3 - Вікно Хеммінга

Потім застосовуємо швидке перетворення Фур'є (STFT) до вищезгаданих фреймів і отримуємо частотну характеристику цих фреймів.

- 3) Відобразити потужності отриманого вище спектру в Mel шкалу, використовуючи трикутні вікна що перекриваються.

Застосовуємо Mel шкалу для масштабування значень фреймів сигналу за допомогою формули:

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \frac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \leq k \leq f(m) \\ \frac{f(m+1) - k}{f(m+1) - f(m)} & f(m) \leq k \leq f(m+1) \\ 0 & k > f(m+1) \end{cases} \quad (1.3)$$

- 4) Взяти логарифми потужностей на кожній з Mel частот.
- 5) Взяти дискретне косинусне перетворення списку потужностей з попереднього пункту.
- 6) Отримали мел-кепстральні коефіцієнти. MFCC - це амплітуди отриманого спектру.

Послідовність отримання Мел-кепстральних коефіцієнтів зображено на малюнку 1.3:

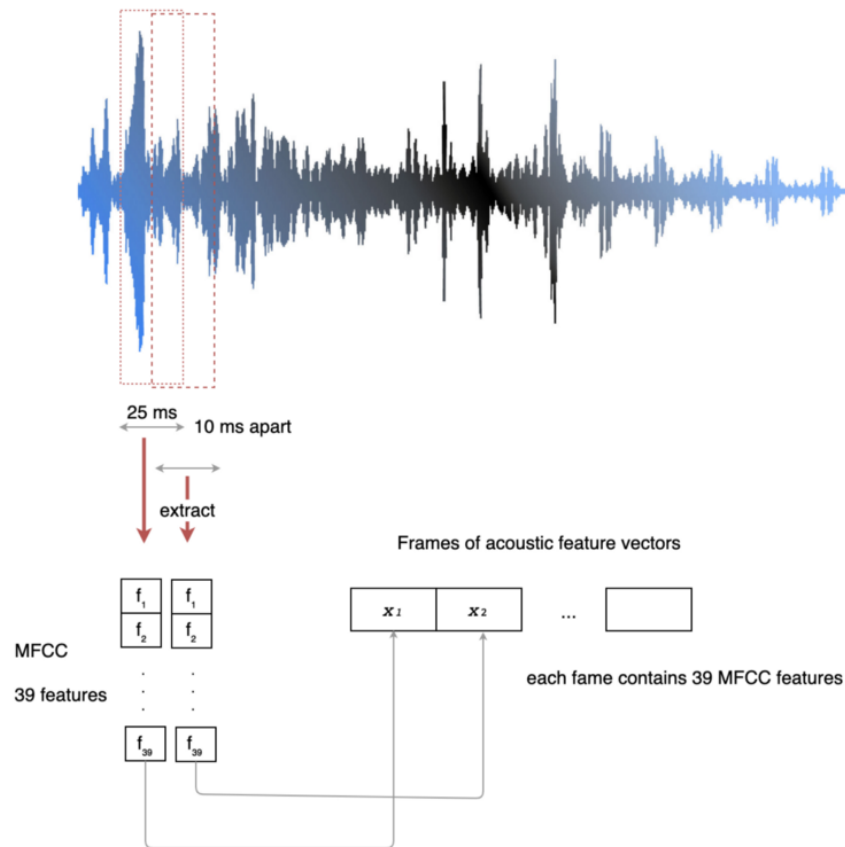


Рисунок 1.3 - MFCC [4]

1.3.2 Акустична модель

Однією з головних складових системи розпізнавання мовлення є акустична модель. Вона представляє з себе математичну модель яка перетворює вхідні акустичні признаки у вихідні фонемі для кожного признаку. До різкого розповсюдження використання нейронних мереж, акустичні моделі створювали на основі Гаусових мікстурних моделей (Gaussian Mixture Model - GMM) або скриті Марковські моделі. Зараз же широко використовуються різні нейромережеві архітектури: від рекурентних нейронних мереж (Recurrent Neural Network - RNN) до нейронних мереж із часовою затримкою (Time Delay Neural Network - TDNN).

Загальна архітектура акустичної моделі представлена на рисунку 1.4:

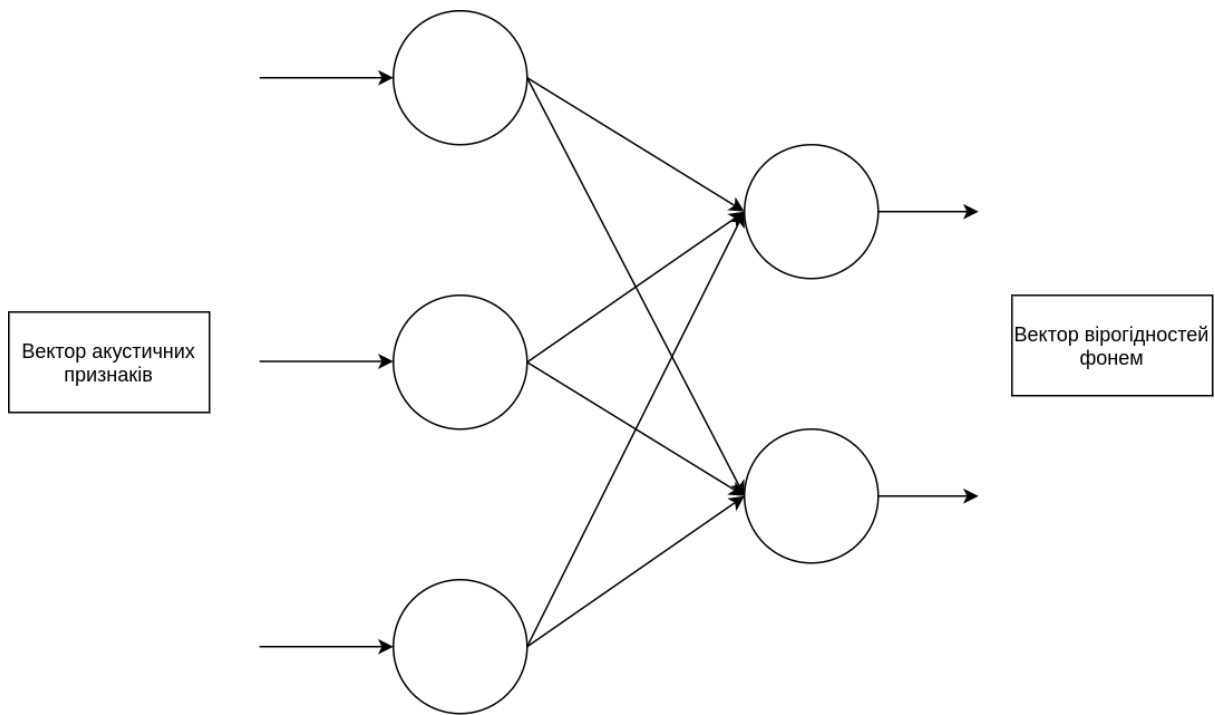


Рисунок 1.4

1.3.3 Мовна модель

Іншою невід'ємною частиною системи розпізнавання мовлення є мовна модель. Дана модель співставляє вірогідність виникнення наданої послідовності слів. Головна ціль мовної моделі - це надання інформації про конкретну мову, її синтаксис, граматику, семантику та прагматику. Так як мовна модель працює тільки з текстовими даними, і не залежить від акустичних даних, її можна натренувати на великих текстових корпусах відповідної мови.

Мовні моделі бувають як статистичні (n-gram, markov models) так і неймережеві мовні моделі (RNNLM, BERT, GPT-2, GPT-3).

1.3.4 N-gram мовна модель

Найчастіше використовувана мовна модель для побудови системи автоматичного розпізнавання мовлення була N-gram мовна модель.

Хоч зараз дослідники та компанії починають використовувати нейронні мовні моделі побудовані на різних архітектурах, в тому числі й на архітектурі “трансформер”, n-грамна мовна модель має перевагу в швидкості роботи та ощадливому відношенні до ресурсів.

N-gram мовна модель представляє з себе статистичну модель яка передбачає найбільш вірогідне слово яке буде йти наступним для N-1 даних слів. Формально N-gram модель можна описати так

$$P(w_n | w_{1:n-1}) \approx P(w_n | w_{n-N+1:n-1}) \quad (1.4)$$

$$P(w_{1:n}) \approx \prod_{k=1}^n P(w_k | w_{k-1}) \quad (1.5)$$

1.3.5 Декодер

В кінцевому етапі, коли треба визначити найбільш вірогідний текст для відповідного мовного сигналу, використовується декодер. Він отримує дані від акустичної моделі, мовної моделі та лексикону і перетворює їх на найбільш вірогідний текст для відповідного набору.

Для декодеру найчастіше використовують такий алгоритм як променевий пошук, але для даної задачі також можна використовувати алгоритм жадібного пошуку та ЕМ-алгоритм.

1.3.6 WFST

WFST (Weighted Finite-State Transducer) - це кінцевий автомат з вагами який повсемірно використовується в розробці систем автоматичного

розпізнавання мовлення. Він надає натуральну реалізацію таких систем як Сховані Марковські Моделі, мовні N-gram моделі, лексики та граматика.

Ключова роль кінцевого автомату з вагами є об'єднання багатьох структур даних в одну і їх оптимізація, тобто він надає зручний інтерфейс до створення таких комбінованих структур.

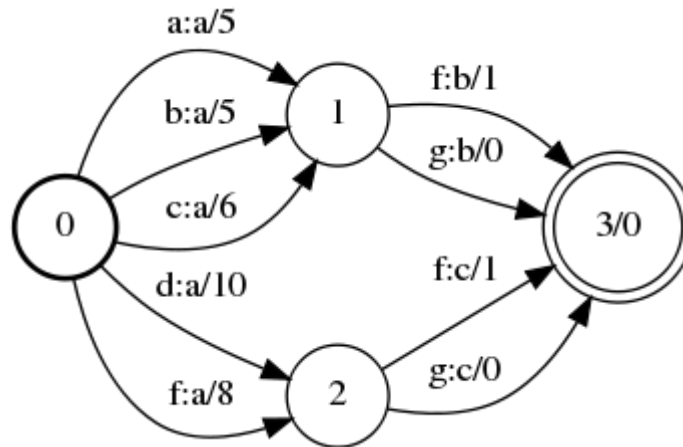


Рисунок 1.3.6 - Приклад WFST декодера [9]

1.4 Математична модель системи АРМ

Нехай ми маємо множу слов мови $W = \{w_1, w_2, \dots, w_n\}$ та множина спостережених аудіо сигналів $O = \{o_1, o_2, \dots, o_n\}$. Задача розпізнавання мовлення зводиться до знаходження такої послідовності слів W^* щоб вірогідність $P(w_1, w_2, \dots, w_n | o_1, o_2, \dots, o_n)$ була максимальна, тобто:

$$W^* = \underset{w}{argmax} P(W | O) \quad (1.6)$$

$$W^* = \underset{w}{argmax} P(w_1, w_2, \dots, w_k | o_1, o_2, \dots, o_m) \quad (1.7)$$

Перебір всіх варіантів для знаходження послідовності з максимальною вірогідністю займав би дуже багато часу та

обчислювальних ресурсів. Тому можна перетворити дану формулу за допомогою теореми Байеса:

$$W^* = \underset{w}{\operatorname{argmax}} \frac{P(O|W)P(W)}{P(O)} = \underset{w}{\operatorname{argmax}} P(O|W)P(W) \quad (1.8)$$

Де $P(O|W)$ - це акустична модель, а $P(W)$ - Мовна модель.

1.5 Метрики точності моделей АРМ

Для вимірювання точності моделей розпізнавання мовлення використовуються такі показники як WER (Word Error Rate) та CER (Character Error Rate). Визначаються вони за такими формулами:

$$WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C} \quad (1.9)$$

де S - кількість замін, D - кількість видалень, I - кількість вставок, C - кількість правильних слів, N - сума S , D та C .

$$CER = \frac{S_c + I_c + D_c}{N_c} \quad (1.10)$$

де відповідні значення виконуються не до слів, а до букв.

В даній роботі буде використаний показник WER так як його найчастіше надають в наукових статтях і аналізах моделей і також він є невід'ємною характеристикою усіх систем автоматичного розпізнавання мовлення які використовують компанії світу.

Модель автоматичного розпізнавання мовлення вважається кращою за іншу модель якщо її показник WER є меншим.

1.6 Висновки до розділу

В даному розділі було надано визначення процесу автоматичного розпізнавання мовлення, виділено загальну функціональну схему розпізнавання мовлення і описано кожну її частину та досліджено і порівняно сучасні архітектури для даних систем.

Також було надано визначення математичної моделі системи автоматичного розпізнавання мовлення та було введено такі метрики точності як WER та CER.

2. АНАЛІЗ СУЧАСНИХ РІШЕНЬ У СФЕРІ АРМ

2.1 Архітектура HMM-GMM

В розробці систем автоматичного розпізнавання мовлення найбільш розповсюдженим був підхід в якому використовувались Гаусові мікстурні моделі (GMM) та Скриті марковські моделі (HMM). Так як мовний сигнал може бути представлений як кусково стаціонарний сигнал, його можна змодельовати за допомогою схованих марковських моделей в яких параметрами можуть бути Гаусові мікстурні моделі. Такий підхід дозволяє змодельовати неперервний сигнал, просторами дискретних значень.

Не дивлячись на те що дана архітектура стала де-факто стандартом у вирішенні задачі розпізнавання автоматичного мовлення, як і будь-який інший підхід, вона має як свої переваги так і свої недоліки. Перевагою даної архітектури є швидкість її роботи та простота реалізації, коли головним її недоліком є статистична неефективність для моделювання нелінійних даних, якими є мовні сигнали.

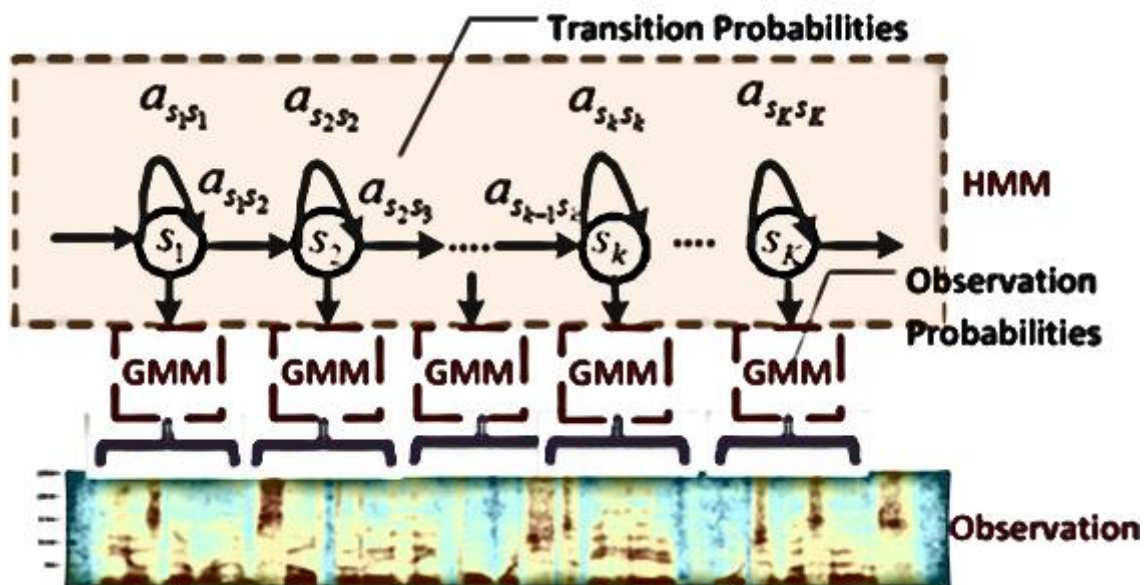


Рисунок 2.1 - Архітектура HMM-GMM [7]

2.2 Архітектура HMM-DNN

Глибоке навчання стало основним напрямком технології розпізнавання мови і успішно замінило Гаусові мікстурні моделі для розпізнавання мови і кодування функцій в усе більшому масштабі. Загалом є три типи архітектур глибокого навчання, які використовуються у розпізнаванні мовлення.

Перший тип складається з генеративних глибоких архітектур, які призначені для характеристики високого порядку кореляційних властивостей даних або спільних статистичних розподілів видимих даних і пов'язаних з ними класів. Використання правила Байєса може перетворити цей тип архітектури в дискримінаційний. Прикладами цього типу є різні форми глибоких автокодувальників, глибокі машини Больцмана, сумарно-виробничі мережі, та інші.

Другий тип глибоких архітектур є дискримінаційним за своєю природою, які класифікують дані, характеризуючи апостеріорні ймовірності класових міток обумовлених видимими даними. Приклади включають глибоко структурований CRF, архітектуру тандем-MLP та інші.

У третьому типі, або гібридних глибоких архітектурах, метою є розрізнення, але з допомогою генеративних архітектур. Генеративна складова в основному використовується для надання допомоги в боротьбі з дискримінацією в якості кінцевої мети гібридної архітектури.

Найбільш практичною реалізацією даної архітектури яка використовується у продакшені та у багатьох застосунках розумних колонок, доміє архітектура TDNN-HMM на основі нейронної мережі з часовою затримкою та схованих марковських моделей.

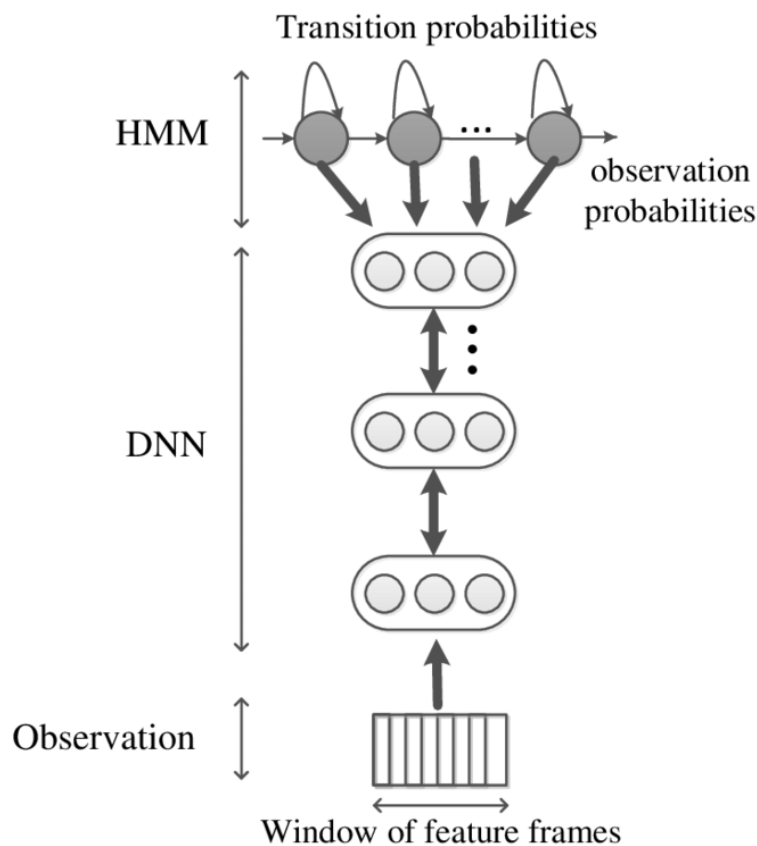


Рисунок 2.2 - Архітектура HMM-DNN [5]

2.2.1 Нейронні мережі з часовою затримкою

Рекурентні нейронні мережі показали себе дуже добре в задачах моделювання послідовностей різного типу, як то звуковий сигнал, показники цін на ринку, чи текст. Але однією з найбільших проблем з даним типом нейронних мереж є час їх тренування: через послідовну структуру такого типу нейронних мереж час їх тренування є більшим ніж час тренування згорткових нейронних мереж або нейронних мереж зворотного зв'язку (feed-forward neural network). Тому у 2015-му році на конференції Interspeech Vijayaditya Peddinti, Daniel Povey, Sanjeev Khudanpur запропонували альтернативу рекурентним нейронним мережам яка представляє з себе нейронну мережу з часовою затримкою (TDNN) [7], тим самим надаючи альтернативу яка так само гарно зможе моделювати послідовності і водночас буде мати менший час тренування.

Нейронна мережа з часовою затримкою (TDNN) — це архітектура штучної нейронної мережі, основною метою якої є класифікація патернів незалежно від зсуву, тобто не вимагає явного попереднього визначення початкової та кінцевої точок схеми. TDNN вперше запропонував класифікувати фонему в мовних сигналах для автоматичного розпізнавання мови, де автоматичне визначення точних сегментів або меж фіч є складним чи неможливим. TDNN розпізнає фонему та їхні основні акустичні / фонетичні особливості, незалежно від часових зрушень, тобто положення в часі.

Вхідний сигнал доповнюється затриманими копіями як іншими входами, нейронна мережа є інваріантною для часової зміни, оскільки вона не має внутрішнього стану.

У оригінальному документі була представлена мережа з персептронами, ваги якої пройшли навчання за допомогою алгоритму зворотного поширення помилки, це може бути здійснено партійно або в реальному часі.

2.2.2 Архітектура нейронних мереж з часовою затримкою

На відміну від стандартних глибинних нейронних мереж (DNN) в які на вхід подається увесь контекст сигналу, до нейронних мереж з часовою затримкою подається лише вузька частина контексту, в той час як більш глибокі шари мають змогу моделювати ширшу частину контексту.

Перетворення в даній нейронній мережі відбуваються через періоди часу, тому часто даний вид нейронної мережі розглядається як попередник згорткової нейронної мережі.

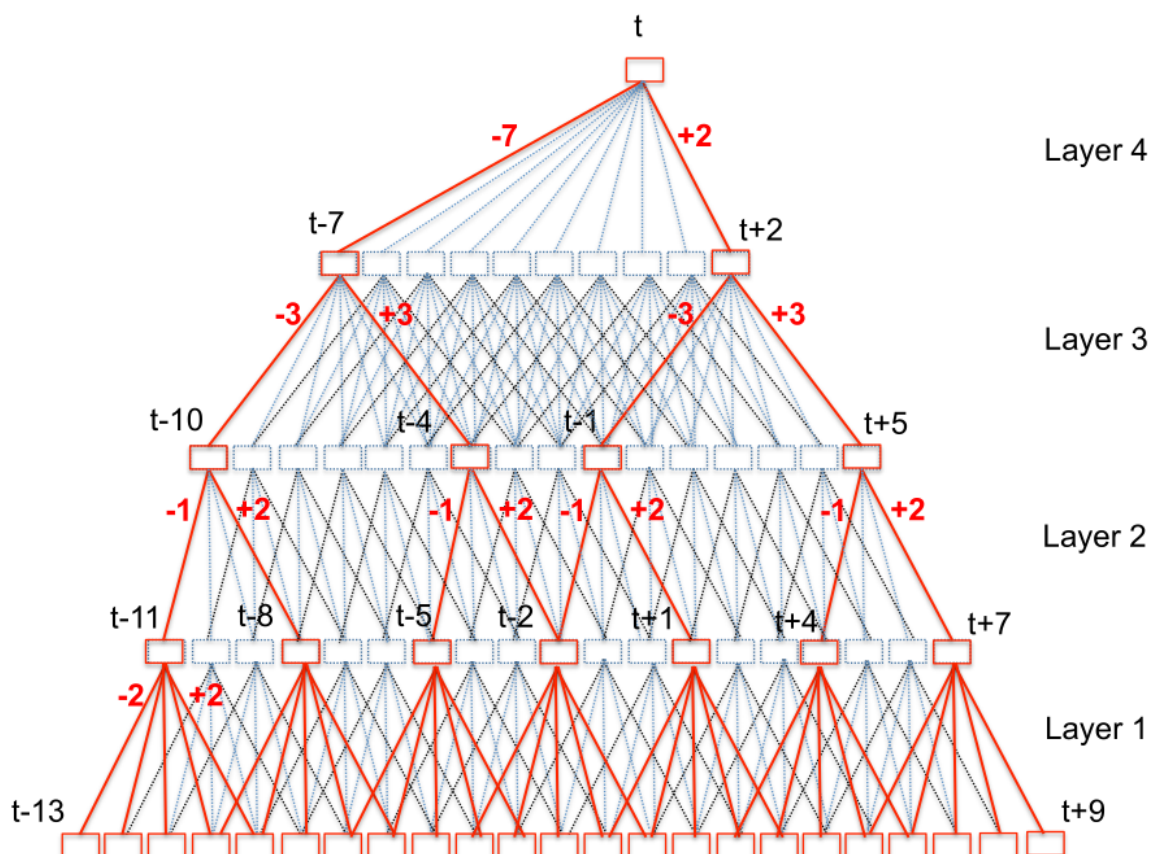


Рисунок 2.3 - Архітектура TDNN [15]

Гіпер-параметрами даної нейромережі є вхідний контекст кожного шару, який потрібен для розрахунку значення активації шару.

Характеристику шарів вищезазначеної нейронної мережі можна побачити в таблиці 1

Шар	Вхідний контекст	Вхідний контекст із підвибіркою
1	$[-2, +2]$	$[-2, 2]$
2	$[-1, 2]$	$\{-1, 2\}$
3	$[-3, 3]$	$\{-3, 3\}$
4	$[-7, 2]$	$\{-7, 2\}$
5	$\{0\}$	$\{0\}$

Таблиця 2.1 - Структура TDNN [15]

2.3 Архітектура Silero Speech-to-text

Один із підходів використання глибинних нейронних мереж є “end-to-end” нейромережева архітектура. В даному випадку розглядається архітектура побудована тільки на згорткових нейронних мережах. Вона складається із двох частин: частини кодування та частини декодування.

Загальна мотивація цієї архітектури це перехід до згорткових нейронних мереж, так як згорткові нейронні мережі вчать набагато швидше від рекурентних нейронних мереж та займають менше часу на вирахування передбачень.

На відміну від багатьох архітектур, дана архітектура не використовує Mel-кепстральні коефіцієнти як признаки які подаються до нейронної мережі. В даній архітектурі використовується звичайне швидке перетворення Фур'є з якого потім модель буде робити текстове передбачення.

Дана архітектура складається із двох частин: кодувальника (Encoder) та декодувальника (Decoder). Кодувальник складається з глибинних згорткових нейронних мереж, які витягують признаки із частотних характеристик. Декодувальник складається із згорткових нейронних мереж які отримують дані від кодувальника та

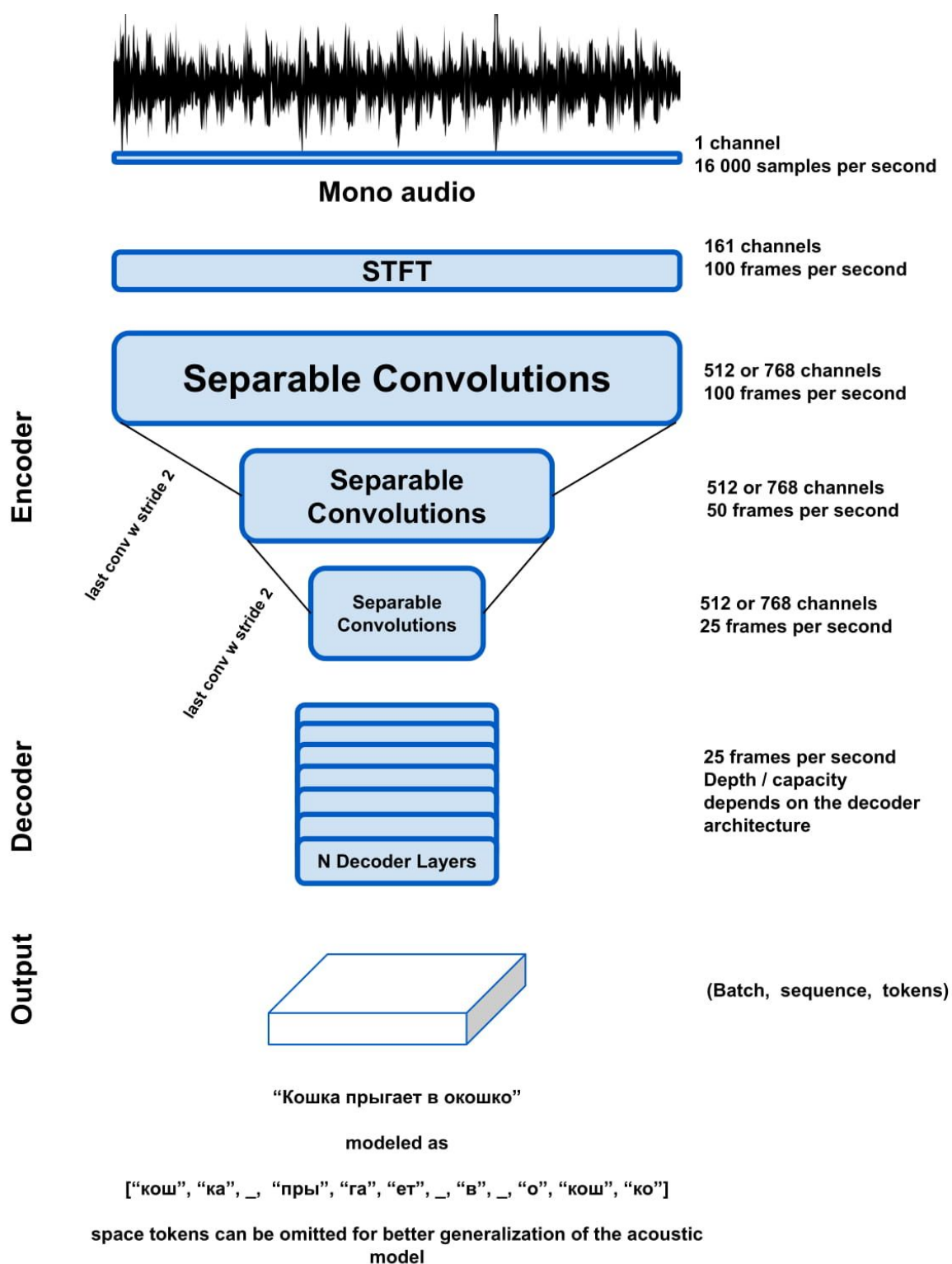


Рисунок 2.4 - Архітектура Silero STT [16]

2.4 Архітектура wav2vec

Ще однією важливою архітектурою є wav2vec. Це нейромережева архітектура яка навчається репрезентувати мовні сигнали. Дана архітектура

навчається за принципом “Self-supervised”, що значить самонавчання: для її тренування можна використовувати лише невелику кількість розмічених даних, а інші дані можуть бути не розміченими. Таким чином неймережа вчиться добре репрезентувати мовні сигнали, які їй надають.

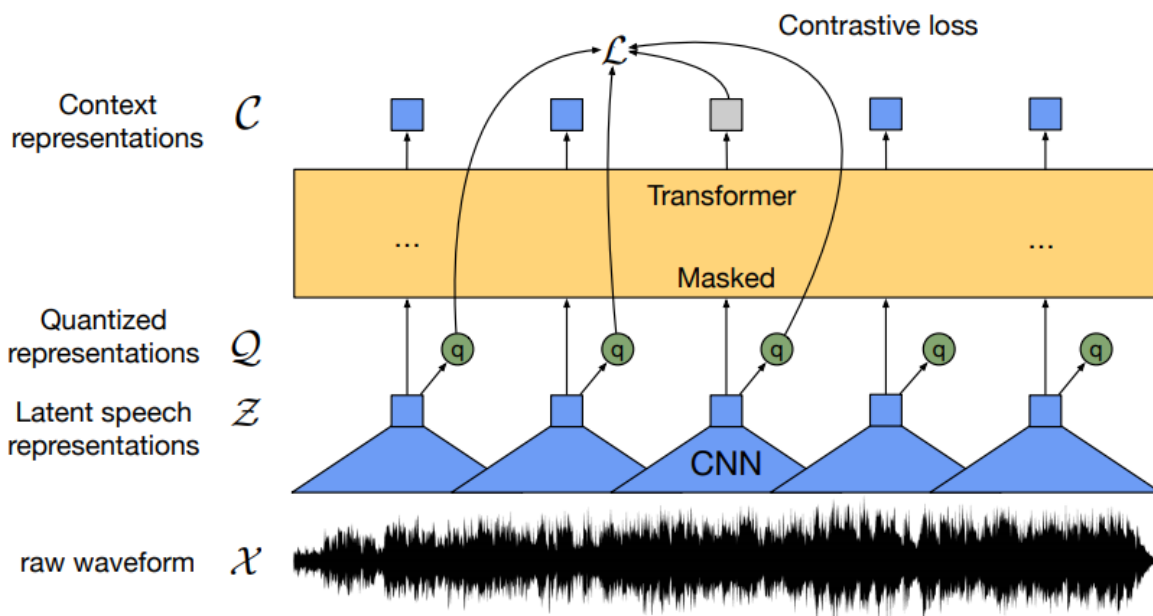


Рисунок 2.5 - Архітектура wav2vec [17]

Далі з репрезентацією працює будь-яка мовна модель, яка перетворює отриманий репрезентаційний контекст у цільовий текст.

Дана модель була мотивована моделлю векторного представлення тексту word2vec.

2.5 Архітектура Deep Speech

У грудні 2014 року команда Baidu Research з Silicon Valley AI Lab анонсувала свою архітектуру системи автоматичного розпізнавання мовлення яка була названа Deep Speech. Ключовим моментом їх підходу є добре оптимізована система для навчання рекурентних нейронних мереж на багатьох графічних картах, що дає приріст в швидкості та в якості результуючої системи.

Дана система представляє з себе єдину рекурентну нейронну мережу яка поєднує в собі як акустичну модель так і мовну модель, віддаючи в кінці текст аудіозапису.

Детальну архітектуру даної системи автоматичного розпізнавання мовлення можна бачити на рисунку нижче:

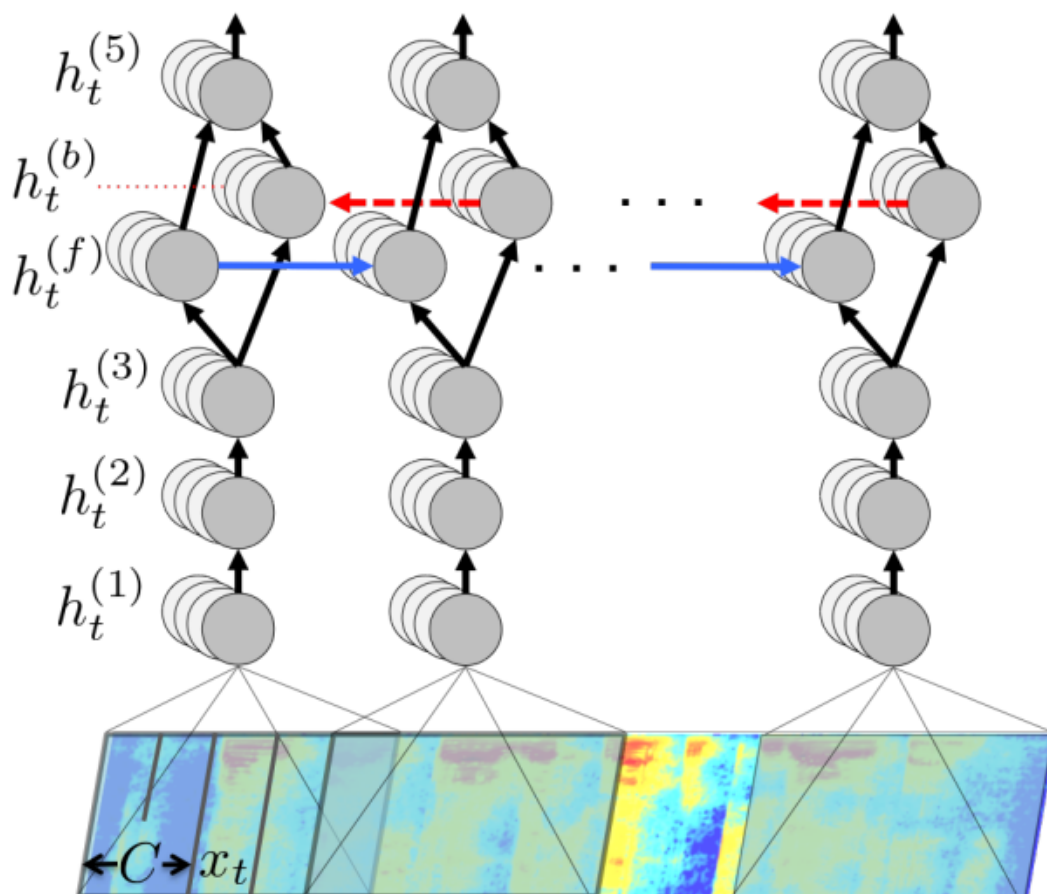


Рисунок 2.5 - Архітектура Deep Speech [1]

2.6 Висновки до розділу

В даному розділі було проведено аналіз сучасних архітектур систем автоматичного розпізнавання мовлення, визначено їх переваги та недоліки та порівняно між собою.

3. РОЗРОБКА ВЛАСНОЇ СИСТЕМИ АРМ

3.1 Вибір фреймворку розробки моделі АРМ

Для розробки власної моделі автоматичного розпізнавання мовлення було обрано фреймворк Kaldi через його легкість в засвоюванні та швидкість виконання коду яка надається йому мовою програмування C++ та bash скриптами.

3.2 Опис фреймворку Kaldi

Kaldi - це інструмент для створення систем розпізнавання мовлення, який базується на мові програмування C++. Даний інструмент був створений для полегшення створення систем розпізнавання мовлення шляхом написання готових “рецептів” коду для вирішення загальних задач, як то вилучення акустичних признаков чи виконання фільтрації, доповнення мовного чи звукового сигналу. Даний набір рецептів коду робить Kaldi зручним для проведення досліджень в області розпізнавання мовлення та у процесі створення робочих систем для розпізнавання мовлення.

Можна виділити головні якості фреймворку Kaldi:

- Наявність загальних алгоритмів та універсальних “рецептів” коду
- Всі алгоритми, які реалізовані в даному фреймворку, є гарно протестованими на помилки
- Даний фреймворк надає зрозумілий код для подальшого його покращення або інших змін

В даному розділі буде проведений аналіз архітектури системи розпізнавання мовлення інструменту Kaldi, яка складається із акустичної моделі TDNN та мовної моделі N-gram.

3.3 Вибір та опис “рецепту” моделі для АРМ

Інструмент калді надає так звані “рецепти” (recipes) створення моделей автоматичного розпізнавання мовлення які представляють з себе скрипти написані на мові bash в яких прописано увесь процес передпідготовки тренувальних даних, тренування акустичної моделі, тренування мовної моделі та тестування моделей.

material	[scripts,egs] Fix shebangs on bash scripts to #!/usr/bin/env bash, fo...
mgb2_arabic/s5	[egs] gale_arabic: add python script to process xml file (#3886)
mgb5	[scripts,egs] Fix shebangs on bash scripts to #!/usr/bin/env bash, fo...
mini_librispeech/s5	[scripts,egs] Create average_rnnlm.sh; add some CNN-based recipes for...
mobvoi	[egs] add MobvoiHotwords SLR87 recipes (#4060)
mobvoihotwords	[egs] add MobvoiHotwords SLR87 recipes (#4060)
multi_cn/s5	Speed up data prep for stcmd (#4433)

Рисунок 3.1

В даній роботі буде розглянуто такий “рецепт” моделі як mini_librispeech який з себе представляє архітектуру TDNN + N-gram + WFST.

3.3.1 Алгоритм тренування моделей за допомогою рецепту mini_librispeech

1) Підготовка даних для тренування моделі:

В самому початку для тренування треба підготувати аудіодані. Усі завантажені аудіозаписи треба переформатувати у .wav формат з частотою дискретизації 16000 Гц та кількістю каналів 1. Ці аудіодані повинні лежати в одній папці для подальшої обробки.

2) Підготовка необхідних файлів аудіозаписів:

Для тренування фреймворк Калді використовує абсолютні шляхи до аудіофайлів для того щоб уникнути проблем їх переміщення та сортування.

Тому в даному кроці потрібно підготувати конфігураційні файли які будуть містити в собі всю необхідну інформацію про аудіодані:

wav.scr - Представляє з себе формат файлу у якому через пробіл записується назва аудіозапису та абсолютний шлях до цього аудіозапису

utt2spk - Представляє себе файл у якому через пробіл записується назва аудіозапису та ім'я директору який проводив який записував цей аудіозапис

spk2utt - Даний файл оберненим до попереднього файлу тобто спочатку записується ім'я диктора а через пробіл записується назва аудіо файлу

text - представляє себе файл в якому через пробіл записується назва аудіозапису та текст який був прочитаний в даному аудіозаписи

spk2gender - представляє себе файл у якому через пробіл записується ім'я диктора та його стать

utt2dur - представляє себе файл в якому через пробіл записується назва аудіо файлу та його тривалість

даний набір файлів підготовлює ця як для тренувального набору аудіоданих так і для тестового та валідаційного набору

3) викликання скриптів для для перепідготовки необхідних файлів для мовної моделі та фонем

4) Викликання скрипту який генерує файли з MFCC із підготовлених аудіофайлів

5) Викликання скрипту який тренує моно-фонетичну модель Тобто статистичну модель кількості фонем які з'являються у тексті

6) Запуск скрипта які тренують трифону модель

7) Запуск скрипта які тренує я мовну м грамм модель

8) Запуск скрипта який тренує акустичну модель яка представляє з себе нейронну мережу часовою зі затримкою

9) Запуск скриптов який створює декодер для подальшого отримання результатів

10) Запуск скрипта який робить кінцеве комбінування графу декодера

3.4 Тренування моделі

Для тренування моделі було використано датасет Ukrainian Open Speech To Text Dataset який містить близько 1000 годин аудіозаписів українського мовлення у форматі .wav з різних доменів.

Загальне розподілення даних по доменам показано в таблиці 3.4:

Домен	Тривалість, год	Відсоток, %
Аудіокниги	863	85
Телевізор	104	10
Youtube	33	3
Телефонія	7	2

Таблиця 3.1 - Характеристики тренувального набору даних

Дивлячись на розподіл даних по доменам можна побачити що деякі домени складають більшу частину усього набору даних. Цей факт може сказатися на точності результуючої моделі на інших доменах при її тестуванні.

3.5 Вибір даних для тестування моделі

Насамперед для тестування потрібно вибрати дані з різних доменів де використовується запис мовлення. Загалом виділяються такі домени як: радіо, дзвінки по телефону, публічні виступи, аудіокниги засідання суда, пісні та відео з Youtube.

В даному порівнянні моделей будуть вибрані дані із трьох доменів: аудіокниги, ютуб та дзвінки по телефону які були завантажені із платформи Youtube.

В якості аудіокниги було взято фрагмент оповідання Панаса Мирного “Хіба ревуть воли, як ясла повні?”. Загалом фрагмент вийшов довжиною 1 хвилина та 50 секунд. В якості дзвінка по телефону було записано розмову з товаришем, яка зайняла 1 хвилину 43 секунди. Та в якості ютуб відео було взято фрагмент новин ТСН який триває 1 хвилину і 40 секунд.

Загальну структуру тестових даних зображено в таблиці:

Домен аудіозапису	Тривалість аудіозапису, с	Розмір аудіозапису, Мб
Аудіокнига	180	1.12
Телефонна розмова	176	1.10
Youtube відео	183	1.13

Таблиця 3.2 - Характеристики тестових аудіозаписів

3.6 Вибір існуючих моделей для порівняння та вимірювання їх якості

Серед простору існуючих моделей для автоматичного розпізнавання мовлення для порівняння було обрано дві моделі: Silero Speech-to-Text uk v3 model та Google Cloud Speech-to-Text API. Обидві моделі є безкоштовними у власному використанні і доступні без кількісного обмеження на використання.

3.7 Вимірювання якості моделей та їх порівняння

Після проведення тестування трьох моделей на тестових даних отримали такі результати:

Модель	WER, %		
	Аудіокниги	Дзвінки по телефону	Youtube
Власна модель	16.21	32.15	16.43
Silero STT uk v3	17.82	37.15	21.12
Google Cloud Speech-to-Text API	16.94	30.01	16.91

Таблиця 3.3 - Результати тестування моделей

В порівнянні з моделями Google та Silero власна модель показала гарні показники, випередивши ці моделі в домені аудіокниг та відео з Youtube. На домені телефонії власна модель показала гірший результат, зайнявши друге місце. Це обумовлено тим що в тренувальних даних, на яких було натреновано власну модель, дані з домену телефонії займали дуже маленький відсоток від всіх.

Причина різниць показників WER між моделями Google та Silero в різних доменах є приводом до подальших досліджень.

4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, розробленого для вирішення задач автоматичного розпізнавання мовлення для української мови.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи автоматичного розпізнавання мовлення. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему,

кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для автоматичного розпізнавання мовлення.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує задачу автоматичного розпізнавання мовлення для української мови та будує його модель. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування;

F_2 – вибір фреймворку машинного навчання;

F_3 – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1 :

а) Python

б) C++

Функція F_2 :

а) Pytorch & Kaldi;

б) Tensorflow;

Функція F_3 :

- a) Jupyter Notebook;
- б) Visual Studio.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 5.1).

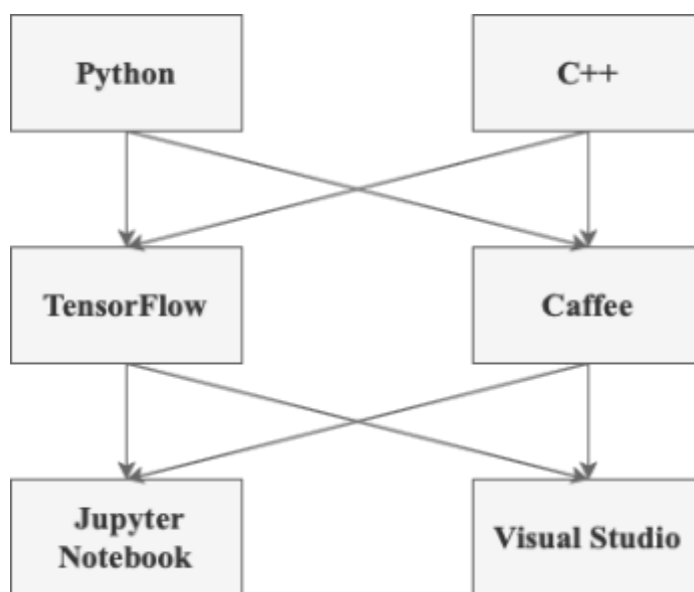


Рисунок 5.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій.

На основі цієї карти будуюмо позитивно-негативну матрицю варіантів основних функцій (Таб.5.2).

Основні функції	Варіанти реалізації	Переваги	Недоліки
	А	Швидка розробка програми, доступність бібліотек,	Низька швидкість роботи.

		кроссплатформеність.	
	Б	Швидке виконання коду	Розробка програм займає багато часу
	А	Надійність, швидкість роботи, наявність готових рецептів	Погана база для розгортання додатків.
	Б	Гарна база для розгортання додатків	Відсутність зворотної сумісності.
	А	Наявність інтерактивного режиму	Тяжка відладка коду
	Б	Багато інструментів, легка відладка коду	Відсутність інтерактивного режиму

Робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1 :

Перевагу віддаємо швидкості вивчення, простоті використання та наявності стандартних бібліотек для обчислення. Для спрощення роботи по написанню коду варіант Б має бути відкинтий.

Функція F_2 :

Обидва варіанти можна використовувати в розробці.

Функція F_3 :

Віддаємо перевагу варіанту А в разі вибору мови програмування Python.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

$$F_{1a} - F_{2a} - F_{3a}$$

$$F_{1a} - F_{2b} - F_{3a}$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів ПП

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- $X1$ – швидкодія мови програмування;
- $X2$ – об'єм пам'яті для обчислень та збереження даних;
- $X3$ – час навчання даних;
- $X4$ – потенційний об'єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 5.4.

Таблиця 5.4 - Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	$X1$	оп/мс	10000	14000	19000
Об'єм пам'яті	$X2$	Мб	420	128	64

Час попередньої обробки даних	X3	мс	4	3	2
Потенційний об'єм програмного коду	X4	кількість рядків коду	4000	2500	1000

За даними таблиці 5.3 будуються графічні характеристики параметрів – рис. 5.2 – рис. 5.5.

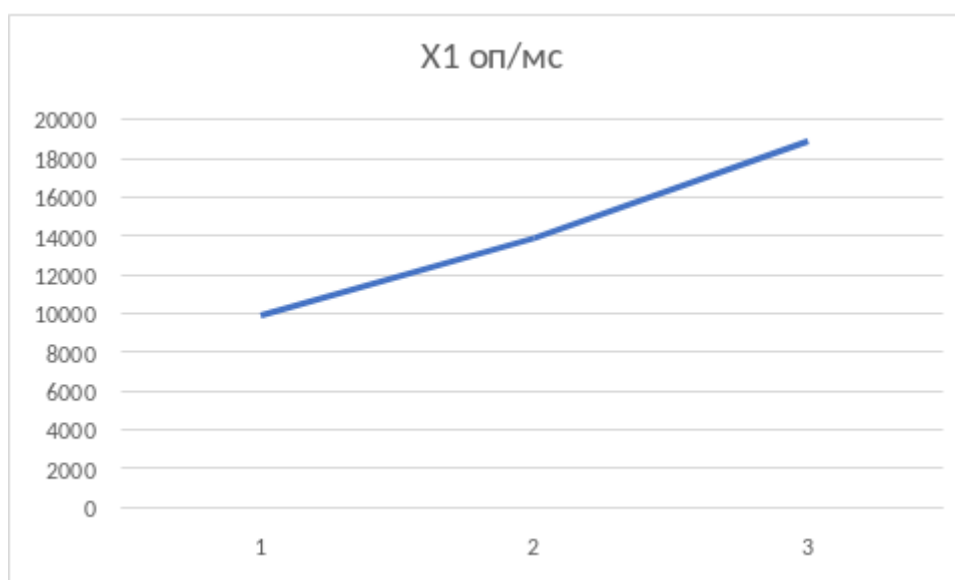


Рисунок 5.2 – X1, швидкодія мови програмування

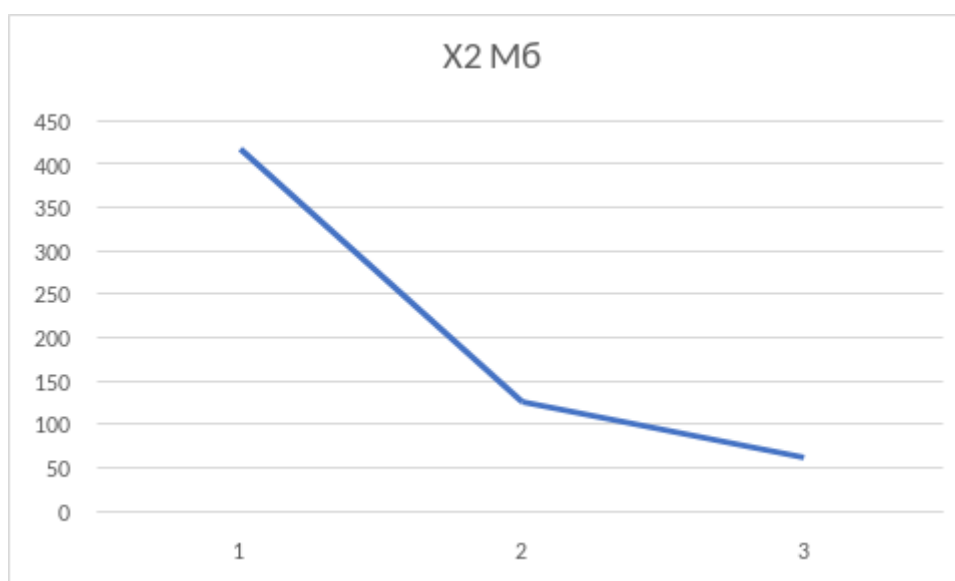


Рисунок 5.3 – X2, об'єм пам'яті

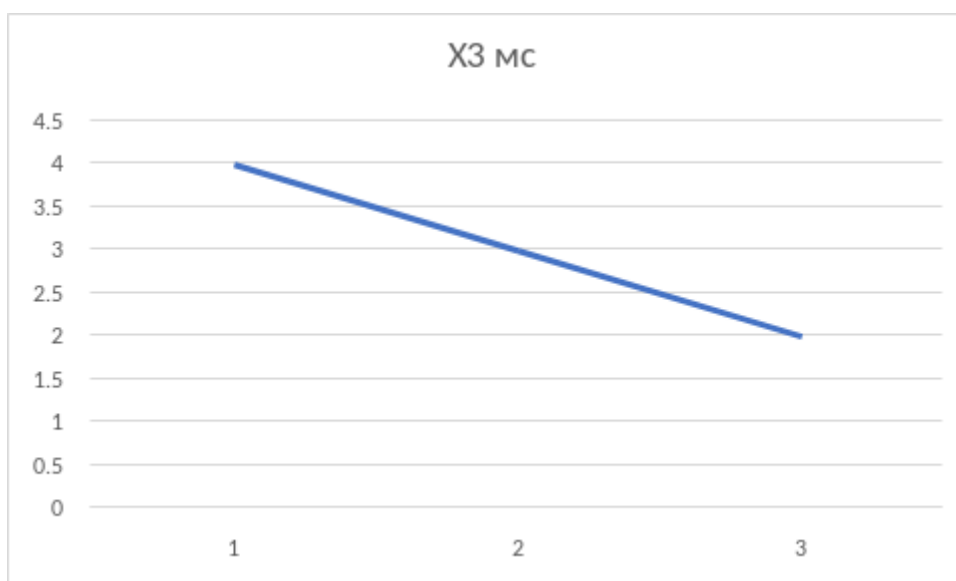


Рисунок 5.4 – X3, час попередньої обробки даних

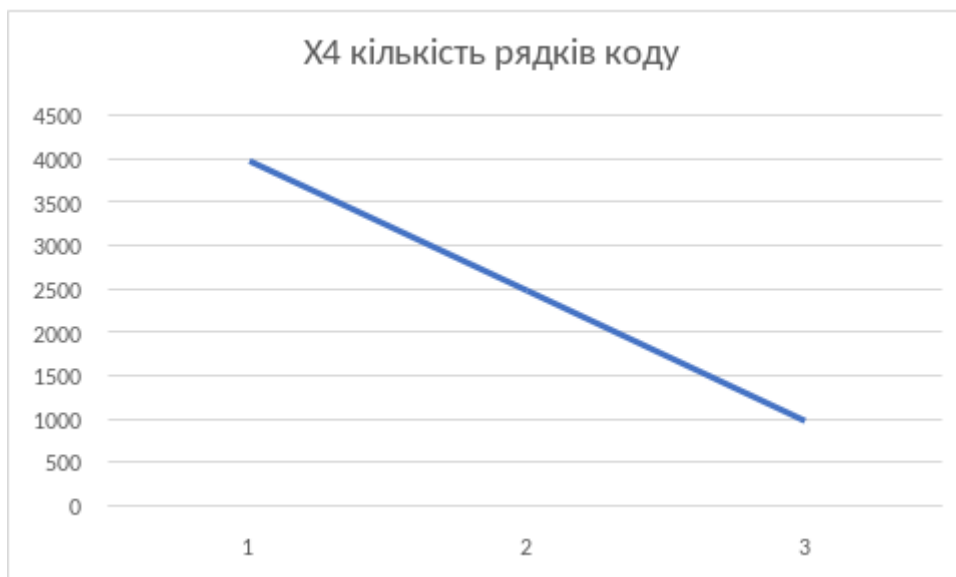


Рисунок 5.5 – X4, потенційний об'єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значущості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 5.4.

Таблиця 5.4 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів в R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
$X1$	Швидкодія мови програмування	Оп/мс	4	5	2	5	3	4	5	28	3,5	12,25
$X2$	Об'єм пам'яті	Мб	2	1	3	1	2	1	2	12	-12,5	156,25

<i>X3</i>	Час попередньої обробки даних	мс	5	3	5	5	4	5	3	30	5,5	30,25
<i>X4</i>	Потенційний об'єм програмного коду	Кількість рядків коду	3	5	4	3	5	4	4	28	3,5	12,25
	Разом		14	14	14	14	14	14	14	98	0	211

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 98, \# (5.1)$$

де N – число експертів,
 n – кількість параметрів;
 б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 24,5 \# (5.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \# (5.3)$$

Сума відхилень по всіх параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^N \Delta_i^2 = 211. \# (5.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 211}{7^2(4^3 - 4)} = 0,86 > W_k = 0,67. \# (5.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 5.5.

Таблиця 5.5 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	<	<	>	>	>	>	1,5
X1 і X3	<	>	<	=	<	<	>	<	0,5
X1 і X4	>	>	<	=	<	=	>	>	1,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	>	<	>	>	<	>	<	>	1,5

Числове значення, що визначає ступінь переваги i -го параметра над j -тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j. \# (5.6) \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A = \|a_{ij}\|$.

Для кожного параметра зробимо розрахунок вагомості K_{Bi} за наступними формулами:

$$K_{Bi} = \frac{b_i}{\sum_{i=1}^n b_i} \# (5.7)$$

$$b_i = \sum_{j=1}^N a_{ij} \# (5.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{Bi} = \frac{b'_i}{\sum_{i=1}^n b'_i}, \# (5.9)$$

$$b'_i = \sum_{j=1}^N a_{ij} b_j \# (5.10)$$

Як видно з таблиці 5.6, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 5.6 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер.	
	X1	X2	X3	X4	b_i	K_{Bi}	b_i^1	K_{Bi}^1	b_i^2	K_{Bi}^2
X1	1,0	1,5	0,5	1,5	4,5	0,36	17,75	0,25	73,38	0,25
X2	0,5	1,0	1,5	0,5	3,5	0,28	15,75	0,22	65,63	0,23

X3	1,5	1,5	1,0	1,5	5,5	0,44	22,75	0,33	93,6	0,32
X4	0,5	1,5	0,5	1,0	3,5	0,28	13,75	0,2	57,63	0,2
Всього:					12,5	1	70	1	290,25	1

5.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів $X2$ (Об'єм пам'яті), $X3$ (час попередньої обробки даних) та $X4$ (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра $X1$ (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 5.7):

$$K_K(j) = \sum_{i=1}^n K_{vi,j} B_{i,j}, \# (5.11)$$

де n – кількість параметрів;

K_{vi} – коефіцієнт вагомості i -го параметра;

B_i – оцінка i -го параметра в балах.

Таблиця 5.7 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт т вагомості параметра	Коефіцієнт т рівня якості
F1	A	X1	10000	6	0,25	1,5

F2	A	X2	64	6	0,23	1,38
	Б	X2	128	2	0,32	0,64
F3	A	X3	1000	10	0,2	2

За даними з таблиці 5.7 за формулою:

$$K_K = K_{TY} [F_{1k}] + K_{TY} [F_{2k}] + \dots + K_{TY} [F_{zk}], \#(5.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1,5 + 1,38 + 2 = 4,88,$$

$$K_{K2} = 1,5 + 0,64 + 2 = 4,14.$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;
2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_P \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М, \#} \quad (5.13)$$

де T_P – трудомісткість розробки ПП;

K_P – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_P = 90$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_P = 1.7$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою

коефіцієнта $K_{СТ} = 0.8$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_p = 27$ людино-днів, $K_{П} = 0.9$, $K_{СК} = 1$, $K_{СТ} = 0.8$:

$$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (122.4 + 19.44 + 4.8 + 19.44) \cdot 8 = 1328.64 \text{ людино-годин.}$$

$$T_{II} = (122.4 + 19.44 + 6.91 + 19.44) \cdot 8 = 1345.52 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 15000 грн., один аналітик в області даних з окладом 20000 грн. Визначимо середню зарплату за годину за формулою:

$$СЧ = \frac{М}{T_m \cdot t} \text{грн., \# (5.14)}$$

де $М$ – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$СЧ = (15000 + 15000 + 20000) / 3 \cdot 21 \cdot 8 = 99,2 \text{ грн}$$

$$СЧ = \frac{10000 + 10000 + 12500}{3 \cdot 21 \cdot 8} = 64,48 \text{ грн. \# (5.15)}$$

Тоді, розраховуємо заробітну плату за формулою:

$$СЗП = С_{\text{ч}} \cdot T_i \cdot КД, \# (5.16)$$

де $С_{\text{ч}}$ – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$КД$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$\text{I.} \quad СЗП = 99,2 \cdot 1328,64 \cdot 1,2 = 158161,3 \text{ грн.}$$

$$\text{II.} \quad СЗП = 99,2 \cdot 1345,52 \cdot 1,2 = 160170,7 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$\text{I.} \quad С_{\text{ВІД}} = СЗП \cdot 0,22 = 158161,3 \cdot 0,22 = 34795,48 \text{ грн.}$$

$$\text{II. } C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 160170,7 \cdot 0.22 = 35237,55 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. (C_M)

Так як одна ЕОМ обслуговує одного програміста з окладом 15000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_{\Gamma} = 12 \cdot M \cdot K_3 = 12 \cdot 15000 \cdot 0,2 = 36000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{\text{ЗП}} = C_{\Gamma} \cdot (1 + K_3) = 36000 \cdot (1 + 0.2) = 43200 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 43200 \cdot 0,22 = 9504 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 22000 грн.

$$C_A = K_{\text{ТМ}} \cdot K_A \cdot \text{Ц}_{\text{ПР}} = 1.15 \cdot 0.25 \cdot 22000 = 6325 \text{ грн.,}$$

де $K_{\text{ТМ}}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

K_A – річна норма амортизації;

$C_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_P = K_{TM} \cdot C_{ПР} \cdot K_P = 1.15 \cdot 22000 \cdot 0.05 = 1265 \text{ грн.},$$

де K_P – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$\begin{aligned} T_{ЕФ} &= (D_K - D_B - D_C - D_P) \cdot t_z \cdot K_B = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.9 = \\ &= 1677.6 \text{ годин}, \end{aligned}$$

де D_K – календарна кількість днів у році;

D_B, D_C – відповідно кількість вихідних та святкових днів;

D_P – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

K_B – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{ЕЛ} = T_{ЕФ} \cdot N_C \cdot K_3 \cdot C_{ЕН} = 1677.6 \cdot 0.3 \cdot 0.7 \cdot 3.51 = 1236.55 \text{ грн.},$$

де N_C – середньо-споживана потужність приладу;

K_3 – коефіцієнтом зайнятості приладу;

Π_{EH} – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_H = \Pi_{ПР} \cdot 0.67 = 22000 \cdot 0,67 = 14740 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{EKC} = C_{ЗП} + C_{ВІД} + C_A + C_P + C_{ЕЛ} + C_H, \#(5.17)$$

$$C_{EKC} = 43200 + 9504 + 6325 + 1265 + 1236.55 + 14740 = 76270,55 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{M-Г} = C_{EKC} / T_{ЕФ} = 76270,55 / 1677.6 = 45,46 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_M = C_{M-Г} \cdot T, \#(5.18)$$

$$I. \quad C_M = 45,46 \cdot 1328,64 = 60405,16 \text{ грн.}$$

$$\text{II. } C_M = 45,46 \cdot 1345,52 = 61172,83 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$C_H = C_{ЗП} \cdot 0,67, \#(5.19)$$

$$\text{I. } C_H = 158161,3 \cdot 0,67 = 105968,07 \text{ грн.}$$

$$\text{II. } C_H = 160170,7 \cdot 0,67 = 107314,36 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_M + C_H, \#(5.20)$$

$$\text{I. } C_{ПП} = 158161,3 + 34795,48 + 61383,16 + 105968,07 = 360308.01 \text{ грн.}$$

$$\text{II. } C_{ПП} = 160170,7 + 35237,55 + 62163,02 + 107314,36 = 364885.63 \text{ грн.}$$

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{ТЕРj} = K_{Кj} / C_{Фj}, \#(5.21)$$

$$K_{ТЕР1} = 4,88 / 360308.01 = 1,35 \cdot 10^{-5},$$

$$K_{\text{TEP}2} = 4,14 / 364885.63 = 1,13 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $K_{\text{TEP}1} = 1,35 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розробляється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості

$$K_{\text{TEP}} = 1,77 \cdot 10^{-5}.$$

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- Використання моделей з великою ємністю
- Використання стандартного інтерфейсу візуалізації, швидкість розробки

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

4.8 Висновки до розділу

Проведено повний функціонально-вартісний аналіз програмного продукту. Визначено та проведено оцінку основних функцій програмного продукту. Визначено параметри, які характеризують програмний продукт.

Проведено експертне оцінювання параметрів та аналіз якості варіантів реалізації функцій.

Проведено економічний аналіз варіантів розробки – трудомісткість, витрати на заробітну плату та інші витрати.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У ході виконання дипломної роботи було проведено аналіз історії та розвитку систем автоматичного розпізнавання мовлення, описані варіанти застосувань, де ця технологія може бути максимально практичною та корисною та наведені приклади її використання в реальному світі.

Для реалізації запланованої моделі автоматичного розпізнавання мовлення було використано фреймворк Kaldi через ряд його переваг:

- Наявність загальних алгоритмів та універсальних “рецептів” коду
- Всі алгоритми, які реалізовані в даному фреймворку, є гарно протестованими на помилки
- Даний фреймворк надає зрозумілий код для подальшого його покращення або інших змін

В процесі розробки моделі автоматичного розпізнавання мовлення, згідно поставлених вимог до розроблюваної моделі, було детально описано декілька можливих варіантів реалізації та проаналізовано їх відповідність до плану проекту.

Самі собою глибокі нейронні мережі потребують достатньо якісного та швидкого обладнання для їх тренування, тому було вирішено використати архітектуру яка б не була дуже вимоглива до обладнання на якій вона б тренувалась і щоб вона відповідала вимогам сучасності у дослідженні та використанні.

В ході проведення дослідження моделей автоматичного розпізнавання мовлення було виконано структурний аналіз, описано логіку роботи та протестовано їх якість на тестових даних у трьох доменах: аудіокниги, телефонія та Youtube. За результатами порівняння власна модель показала найкращі показники якості на доменах аудіокниг та Youtube: 16.21% та 16.43% показника WER відповідно, коли на домені телефонії власна модель з показником WER 32.15% поступилась першим місцем моделі від Google з показником WER 30.01%.

Причина різниць показників WER між моделями Google та Silero в різних доменах є приводом до подальших досліджень. Не дивлячись на те, що відсоток даних з Youtube є меншим за відсоток даних аудіокниг, власна модель показує майже однакові результати на тестуванні в цих доменах.

Також з метою проведення техніко-економічного аналізу процесу розробки застосунку, було проведено функціонально-вартісний аналіз (ФВА) розробленого програмного продукту, який мав на меті визначити найкращу альтернативу реалізації системи серед наданих альтернатив та оцінити необхідний бюджет для її створення та подальшого підтримання на будь-якому сервері доступних хмарних сервісів.

Оскільки дана дипломна робота мала на меті розробку системи автоматичного розпізнавання мовлення з точки зору кінцевого користувача та розробника програмного забезпечення, система розроблена за результатами аналізу може знайти застосування в навчальному процесі для ознайомлення з характеристиками наявних рішень щодо синтезу мовлення та процесу інтеграції з ними розроблюваного програмного додатку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Deep Speech: Scaling up end-to-end speech recognition. Baidu Research – Silicon Valley AI Lab [Стаття] - Режим доступу до ресурсу:
<https://arxiv.org/pdf/1412.5567.pdf>
2. Understanding kaldi recipes with mini-librispeech example (part 1 — HMM models) [Електронний ресурс]- Режим доступу до ресурсу:
<https://medium.com/@qianhwan/understanding-kaldi-recipes-with-mini-librispeech-example-part-1-hmm-models-472a7f4a0488>
3. Understanding kaldi recipes with mini-librispeech example (part 2— DNN models) [Електронний ресурс] - Режим доступу до ресурсу:
<https://medium.com/@qianhwan/understanding-kaldi-recipes-with-mini-librispeech-example-part-2-dnn-models-d1b851a56c49>
4. Speech Recognition — Feature Extraction MFCC & PLP. [Електронний ресурс] - Режим доступу до ресурсу:
<https://jonathan-hui.medium.com/speech-recognition-feature-extraction-mfcc-plp-5455f5a69dd9>
5. Acoustic Model Selection for Recognition of Regional Accented Speech. Maryam Najafian [Стаття] - Режим доступу до ресурсу:
https://www.researchgate.net/figure/Structure-of-a-DNN-HMM-system-5_fig8_292365466
6. Ukrainian Open Speech To Text Dataset ~1000 hours [Електронний ресурс] - Режим доступу до ресурсу:
<https://academictorrents.com/details/50f7a8e6157a9c2e38919afee0a11d8145e35556>
7. From gmm-hmm to dnn-hmm. Programmersought. [Електронний ресурс] - Режим доступу до ресурсу:
<https://www.programmersought.com/article/68723391616/>
8. SPEECH RECOGNITION FOR THE DEAF AND HARD OF HEARING - MeMAD [Електронний ресурс] - Режим доступу до ресурсу:

- <https://memad.eu/2019/08/14/speech-recognition-deaf-hard-hearing/>
9. An Introduction to Speech Recognition using WFSTs. Desh Raj
[Электронный ресурс] - Режим доступа до ресурсу
<https://medium.com/explorations-in-language-and-learning/an-introduction-to-speech-recognition-using-wfsts-288b6aeecebe>
 10. NVIDIA Developer Blog [Электронный ресурс] - Режим доступа до ресурсу:
<https://developer.nvidia.com/blog/nvidia-accelerates-speech-text-transcription-3500x-kaldi/>
 11. Speech Recognition. Wikipedia [Электронный ресурс] - Режим доступа до ресурсу:
https://en.wikipedia.org/wiki/Speech_recognition
 12. Speech and Language Processing. Daniel Jurafsky & James H. Martin
[Книга] - Режим доступа до ресурсу:
<https://web.stanford.edu/~jurafsky/slp3/3.pdf>
 13. Speech Recognition — Weighted Finite-State Transducers (WFST)
[Электронный ресурс] - Режим доступа до ресурсу:
<https://jonathan-hui.medium.com/speech-recognition-weighted-finite-state-transducers-wfst-a4ece08a89b7>
 14. Speech Recognition — GMM, HMM. Jonathan Hui [Электронный ресурс] - Режим доступа до ресурсу:
<https://jonathan-hui.medium.com/speech-recognition-gmm-hmm-8bb5eff8b196>
 15. A time delay neural network architecture for efficient modeling of long temporal contexts. Vijayaditya Peddinti , Daniel Povey, Sanjeev Khudanpur [Статья] - Режим доступа до ресурсу:
https://www.danielpovey.com/files/2015_interspeech_multisplICE.pdf
 16. Silero Speech-to-Text. Torch hub [Электронный ресурс] - Режим доступа до ресурсу:
https://pytorch.org/hub/snakers4_silero-models_stt/

17. wav2vec: Unsupervised Pre-training for Speech Recognition. Steffen Schneider, Alexei Baevski, Ronan Collobert, Michael Auli [Электронный ресурс] - режим доступа до ресурсу:
<https://arxiv.org/abs/1904.05862>

ДОДАТОК А

Програмний код підготовки текстових даних для тренування

```
KALDI_ROOT="/home/arturka/Documents/kaldi"
```

```
$KALDI_ROOT/tools/srilm/bin/i686-m64/ngram-count -order 3 wbdiscount  
-write-vocab librispeech-vocab.txt -text corpus.txt -lm lm.arpa
```

```
$KALDI_ROOT/tools/srilm/bin/i686-m64/ngram -lm lm.arpa -prune 1e-8  
-write-lm lm_tglarge.arpa.gz
```

```
$KALDI_ROOT/tools/srilm/bin/i686-m64/ngram -lm lm.arpa -prune 1e-7  
-write-lm lm_tgmed.arpa.gz
```

```
$KALDI_ROOT/tools/srilm/bin/i686-m64/ngram -lm lm.arpa -prune 3e-7  
-write-lm lm_tgsmall.arpa.gz
```

```
python g2p.py --file librispeech-vocab.txt --out librispeech-lexicon.txt
```

ДОДАТОК Б

Програмний код підготовки аудіо даних для тренування

```
import os
import sys

import numpy as np
import pandas as pd
import librosa

import re

from tqdm import tqdm

def preprocess_text(text):
    cleaned_str = re.sub('[^a-яііе\|s|]+', '', text, flags=re.IGNORECASE)
    cleaned_str = re.sub('(\|s|)+', ' ', cleaned_str)
    cleaned_str = cleaned_str.lower()

    return cleaned_str

def create_df(path, text_file, spk=None, gender=None,
              folder_name="none"):
    """
    Creates df from audio data dir
    which have to contains wav and etc
    subfolders
    """
    filenames = os.listdir(path+'wav')
    filepaths = [path+'wav/'+f for f in filenames]
    filesizes = [os.path.getsize(f) for f in filepaths]
```

```

with open(path+'/etc/'+text_file) as f:
    transcriptions = f.readlines()

if not spk:
    speakers = ['anon']*len(filenamees)

if not gender:
    genders = ['m']*len(filenamees)

df = pd.DataFrame({'file_name': sorted(filenamees),
                   'file_path': sorted(filepaths),
                   'file_size': sorted(filesizes),
                   'transcription': sorted(transcriptions),
                   'gender': genders,
                   'speaker': speakers})
    df['transcription'] = df['transcription'].apply(lambda x:
preprocess_text(x).strip())
    df['utt_id'] = folder_name + '_' + df['file_name'].apply(lambda x: x[0:-4])

return df

def spk2gender(df, folder=""):
    res = []
    for s in set(df['speaker']):
        gender = list(df[df['speaker'] == 'anon']['gender'])[0]
        res.append(s+' '+gender)
    with open(folder + 'spk2gender', 'w') as f:

```

```

    for e in res:
        f.write(f'{e}\n')
    return res

```

```

def wavscp(df, folder=""):
    res = []
    for u,p in zip(df['utt_id'], df['file_path']):
        res.append(u+' '+p)
    with open(folder + 'wav.scp', 'w') as f:
        for e in res:
            f.write(f'{e}\n')
    return

```

```

def text_file(df, folder=""):
    res = []
    for u,t in zip(df['utt_id'], df['transcription']):
        res.append(u+' '+t)
    with open(folder+'text', 'w') as f:
        for e in res:
            f.write(f'{e}\n')
    return

```

```

def utt2spk(df, folder=""):
    res = []
    for u,s in zip(df['utt_id'], df['speaker']):
        res.append(u+' '+s)
    with open(folder+'utt2spk', 'w') as f:
        for e in res:
            f.write(f'{e}\n')

```

```

return

if __name__ == '__main__':
    PATH = sys.argv[1]
    test_perc = float(sys.argv[2])
    print("PATH:", PATH)
    print("test_perc:", test_perc)
    df = create_df(PATH, 'txt.withoutdigitdone.data',
folder_name="custom_dataset")

    cut = int(len(df)*test_perc)
    df_train, df_test = df[cut:], df[:cut]

    for df_, folder_ in zip([df_train, df_test], ['train/', 'test/']):
        wavscp(df_, folder_)
        spk2gender(df_, folder_)
        text_file(df_, folder_)
        utt2spk(df_, folder_)

```

ДОДАТОК В

Програмний код скрипту запуску тренування

```
#!/bin/bash

# Change this location to somewhere where you want to put the data.
data=./corpus/

. ./cmd.sh
. ./path.sh

stage=0
. utils/parse_options.sh

set -euo pipefail

mkdir -p $data

if [ $stage -le 0 ]; then
    # utils/fix_data_dir.sh data/ukraine

    # utils/subset_data_dir.sh --first data/ukraine 20000 data/dev_clean_2 # 5hr
    6min
    # n=$(cat data/ukraine/text | wc -l - 20000)
    # utils/subset_data_dir.sh --last data/ukraine $n data/train_clean_5

    for part in dev_clean_2 train_clean_5; do
        utils/fix_data_dir.sh data/$part
        utils/utt2spk_to_spk2utt.pl data/$part/utt2spk > data/$part/spk2utt || exit 1
        utils/validate_data_dir.sh --no-feats data/$part
```

```

done
fi

if [ $stage -le 1 ]; then

    local/prepare_dict.sh --stage 3 --nj 4 --cmd "$train_cmd" \
        data/local/lm data/local/lm data/local/dict_nosp

    utils/prepare_lang.sh data/local/dict_nosp \
        "<UNK>" data/local/lang_tmp_nosp data/lang_nosp

    local/format_lms.sh --src-dir data/lang_nosp data/local/lm
    # Create ConstArpaLm format language model for full 3-gram and 4-gram
    LMs
    utils/build_const_arpa_lm.sh data/local/lm/lm_tglarge.arpa.gz \
        data/lang_nosp data/lang_nosp_test_tglarge
fi

if [ $stage -le 2 ]; then
    mfccdir=mfcc
    # spread the mfccs over various machines, as this data-set is quite large.

    for part in dev_clean_2 train_clean_5; do
        utils/fix_data_dir.sh data/$part
        steps/make_mfcc.sh --cmd "$train_cmd" --nj 4 data/$part
        exp/make_mfcc/$part $mfccdir
        steps/compute_cmvn_stats.sh data/$part exp/make_mfcc/$part $mfccdir
    done

    for part in dev_clean_2 train_clean_5; do

```

```

    utils/fix_data_dir.sh data/$part
done
# Get the shortest 500 utterances first because those are more likely
# to have accurate alignments.
utils/subset_data_dir.sh --shortest data/train_clean_5 15000 data/train_500short
fi

# train a monophone system
if [ $stage -le 3 ]; then
    # TODO(galv): Is this too many jobs for a smaller dataset?
    steps/train_mono.sh --boost-silence 1.25 --nj 4 --cmd "$train_cmd" \
        data/train_500short data/lang_nosp exp/mono

    steps/align_si.sh --boost-silence 1.25 --nj 4 --cmd "$train_cmd" \
        data/train_clean_5 data/lang_nosp exp/mono exp/mono_ali_train_clean_5
    echo "step completed"
fi

# train a first delta + delta-delta triphone system on all utterances
if [ $stage -le 4 ]; then
    steps/train_deltas.sh --boost-silence 1.25 --cmd "$train_cmd" \
        2000 10000 data/train_clean_5 data/lang_nosp exp/mono_ali_train_clean_5
    exp/tri1

    steps/align_si.sh --nj 4 --cmd "$train_cmd" \
        data/train_clean_5 data/lang_nosp exp/tri1 exp/tri1_ali_train_clean_5
    echo "step completed"
fi

# train an LDA+MLLT system.

```

```

if [ $stage -le 5 ]; then
    steps/train_lda_mllt.sh --cmd "$train_cmd" \
        --splice-opts "--left-context=3 --right-context=3" 2500 15000 \
        data/train_clean_5 data/lang_nosp exp/tri1_ali_train_clean_5 exp/tri2b

    # Align utts using the tri2b model
    steps/align_si.sh --nj 4 --cmd "$train_cmd" --use-graphs true \
        data/train_clean_5 data/lang_nosp exp/tri2b exp/tri2b_ali_train_clean_5
    echo "step completed"
fi

# Train tri3b, which is LDA+MLLT+SAT
if [ $stage -le 6 ]; then
    steps/train_sat.sh --cmd "$train_cmd" 2500 15000 \
        data/train_clean_5 data/lang_nosp exp/tri2b_ali_train_clean_5 exp/tri3b
    echo "step completed"
fi

# Now we compute the pronunciation and silence probabilities from training
# data,
# and re-create the lang directory.
if [ $stage -le 7 ]; then
    steps/get_prons.sh --cmd "$train_cmd" \
        data/train_clean_5 data/lang_nosp exp/tri3b
    utils/dict_dir_add_pronprobs.sh --max-normalize true \
        data/local/dict_nosp \
        exp/tri3b/pron_counts_nowb.txt exp/tri3b/sil_counts_nowb.txt \
        exp/tri3b/pron_bigram_counts_nowb.txt data/local/dict

    utils/prepare_lang.sh data/local/dict \

```

```

"<UNK>" data/local/lang_tmp data/lang

local/format_lms.sh --src-dir data/lang data/local/lm

utils/build_const_arpa_lm.sh \
    data/local/lm/lm_tglarge.arpa.gz data/lang data/lang_test_tglarge

steps/align_fmllr.sh --nj 4 --cmd "$train_cmd" \
    data/train_clean_5 data/lang exp/tri3b exp/tri3b_ali_train_clean_5
echo "step completed"
fi

if [ $stage -le 8 ]; then
    # Test the tri3b system with the silprobs and pron-probs.

    # decode using the tri3b model
    utils/mkgraph.sh data/lang_test_tgsmall \
        exp/tri3b exp/tri3b/graph_tgsmall
    for test in dev_clean_2; do
        steps/decode_fmllr.sh --nj 4 --cmd "$decode_cmd" \
            exp/tri3b/graph_tgsmall data/$test \
            exp/tri3b/decode_tgsmall_$test
        steps/lmrescore.sh --cmd "$decode_cmd" data/lang_test_{tgsmall,tgmed} \
            data/$test exp/tri3b/decode_{tgsmall,tgmed}_$test
        steps/lmrescore_const_arpa.sh \
            --cmd "$decode_cmd" data/lang_test_{tgsmall,tglarge} \
            data/$test exp/tri3b/decode_{tgsmall,tglarge}_$test
    done
    echo "step completed"

```

fi

Train a chain model

if [\$stage -le 9]; then

local/chain/tuning/run_tdn_1j.sh --stage 0

fi

local/grammar/simple_demo.sh

Don't finish until all background decoding jobs are finished.

wait

ДОДАТОК Г

Програмний код серверу автоматичного розпізнавання мовлення

```
from vosk import Model, KaldiRecognizer, SetLogLevel
import sys
import os
import wave

SetLogLevel(0)

if not os.path.exists("model"):
    exit (1)

if not os.path.exists("recognizer"):
    exit (1)

wf = wave.open(sys.argv[1], "rb")
if wf.getnchannels() != 1 or wf.getsampwidth() != 2 or wf.getcomptype() !=
"NONE":
    print ("Audio file must be WAV format mono PCM.")
    exit (1)

model = Model("model")
rec = KaldiRecognizer(model, wf.getframerate())
rec.SetWords(True)

while True:
    data = wf.readframes(4000)
    if len(data) == 0:
        break
    if rec.AcceptWaveform(data):
```

```
        print(rec.Result())
    else:
        print(rec.PartialResult())

print(rec.FinalResult())
```