

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА**

**Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»**

Кафедра інформаційних систем в економіці

**ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Системи штучного інтелекту»**

галузь знань	12 Інформаційні технології
спеціальність	122 Комп'ютерні науки

Форма навчання: очна (денна)

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему: «Розроблення застосунку для розпізнавання аудіо з використанням інструментарію машинного навчання»

здобувача Швиденко Дмитро Юрійович
(ПІБ, підпис)

Науковий керівник: к.т.н., доцент, Васильєва Л.В
(науковий ступінь, учене звання, ПІБ)

(підпис)

**Робота допущена до захисту перед екзаменаційною
комісією з атестації здобувачів вищої освіти (ЕК)**

Завідувач кафедри: к.е.н., доцент Тішков Б.О. _____
(підпис)

Київ 2025

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА**

**Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці**

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА

«Системи штучного інтелекту»

галузь знань	12 Інформаційні технології
спеціальність	122 Комп'ютерні науки

ПОГОДЖЕНО:

Керівник проєктної групи (гарант)
освітньо-професійної програми

_____ Васильєва Л.В.
« ____ » _____ 202_ р.

ЗАТВЕРДЖУЮ:

Завідувач кафедри інформаційних
систем в економіці

_____ Тішков Б.О.
« ____ » _____ 202_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти _____ Швиденка Дмитра Юрійовича
(прізвище, ім'я, по батькові)

_____ очної (денної) _____ форми навчання
очної (денної), заочної, дистанційної

на підготовку кваліфікаційної магістерської роботи

на тему: «Розроблення застосунку для розпізнавання аудіо з використанням інструментарію машинного навчання»

Тему затверджено наказом ректора Університету від «07» ____ 03 ____ 2025 р. № 464-ст ____

Кваліфікаційна магістерська робота виконується на матеріалах наукових публікацій та матеріалів з мережі інтернет _____

План кваліфікаційної магістерської роботи

Розділ I _ Теоретичні основи автоматичного розпізнавання аудіо та мови _____
(назва розділу)

Розділ II _ Проектування системи розпізнавання аудіо _____
(назва розділу)

Розділ III _ Розроблення проєктних рішень _____
(назва розділу)

Об'єкт дослідження: Процес автоматичного розпізнавання мовлення _____

Предмет дослідження: Методи й програмні засоби реалізації систем; розпізнавання українського аудіо на основі моделей машинного навчання.

Мета кваліфікаційної магістерської роботи: Створення програмного застосунку, який працює локально, використовує відкриті self-supervised моделі (Whisper) для транскрибування україномовного мовлення та має простий графічний інтерфейс користувача.

Конкретні завдання, які здобувач повинен виконати для досягнення поставленої мети:

У розділі I Провести аналіз теоретичних основ автоматичного розпізнавання мовлення; поглянути сучасні моделі машинного навчання (Whisper, Wav2Vec 2.0) та відповідні інструменти; проаналізувати існуючі рішення й сформулювати наукову проблему; Обґрунтувати вибір технологій і підходів до розробки системи.

У розділі II Визначити вимоги до системи розпізнавання мовлення; побудувати концептуальну модель і обрати архітектурні рішення; обґрунтувати вибір мов програмування, бібліотек і фреймворків; спроектувати структуру застосунку та підготувати UML-діаграми й інші пояснювальні схеми.

У розділі III Реалізувати функціональні модулі застосунку: обробку аудіо, транскрипцію, графічний інтерфейс; протестувати систему на різних пристроях (CPU / GPU); провести дослідження точності, швидкодії та якості розпізнавання; оформити результати у вигляді графіків, таблиць і описових висновків.

**Завдання підготував
науковий керівник**

(підпис)

Васильєва Л. В.
(ініціали, прізвище)

« ____ » _____ 202_ р.

**Завдання одержав
здобувач**

(підпис)

Швиденко Д. Ю.
(ініціали, прізвище)

« ____ » _____ 202_ р.

Реферат

Кваліфікаційна магістерська робота містить 61 сторінок, 3 таблиць, 16 рисунків, список використаних джерел з 28 найменувань, додатки.

«Розроблення застосунку для розпізнавання аудіо з використанням інструментарію машинного навчання»

Об'єктом дослідження кваліфікаційної магістерської роботи є процес автоматичного розпізнавання мовлення.

Предметом дослідження є методи й програмні засоби реалізації систем автоматичного розпізнавання українськомовного аудіо на основі моделей машинного навчання.

Мета і завдання дослідження. Створення програмного застосунку, який працює локально, використовує відкриті self-supervised моделі (Whisper) для транскрибування українськомовного мовлення та має простий графічний інтерфейс користувача.

Відповідно до поставленої мети визначені такі *завдання*:

- дослідити сучасні підходи до створення ASR-систем;
- обґрунтувати вибір моделі розпізнавання (Whisper);
- реалізувати розпізнавання мовлення з мікрофона і аудіофайлів;
- спроектувати графічний інтерфейс користувача (GUI);
- оцінити продуктивність розробленої системи на CPU та GPU;
- протестувати точність транскрипції для українськомовного аудіо.

Методи дослідження: аналіз літератури, самостійне експериментальне моделювання (на Python з використанням бібліотек Whisper, Librosa, PySide6), тестування точності та швидкодії.

Теоретична, методична та практична значущість отриманих результатів. Робота містить огляд сучасних методів розпізнавання мовлення та реалізацію десктопного застосунку, що може бути базою для подальших досліджень або використання у практичних цілях (освіта, медіа, юриспруденція). Практичні результати продемонстрували можливість ефективної локальної роботи системи без потреби в інтернеті навіть на CPU, а GPU забезпечує прискорену обробку.

Рік виконання кваліфікаційної магістерської роботи – 2025.

Рік захисту роботи – 2025.

Ключові слова: ASR, Whisper, транскрипція, Python, мовлення, аудіо, машинне навчання, інтерфейс, GPU, CPU.

Abstract

The master's qualification thesis comprises 61 pages, 3 tables, 16 figures, a list of 28 references, and appendices.

«Developing an audio recognition application using machine learning tools»

Object of research The process of automatic speech recognition.

Subject of research Methods and software tools for implementing systems of automatic recognition of Ukrainian-language audio based on machine learning models.

Purpose and objectives of the research: The aim is to create a software application that operates locally, uses open self-supervised models (Whisper) to transcribe Ukrainian speech, and provides a simple graphical user interface.

The following tasks were defined to achieve this goal:

- To study modern approaches to building ASR systems;
- To justify the choice of the Whisper recognition model;
- To implement speech recognition from microphone and audio files;
- To design a graphical user interface (GUI);
- To evaluate the system's performance on CPU and GPU;
- To test transcription accuracy for Ukrainian audio.

Research methods: Literature review, independent experimental modeling (in Python using Whisper, Librosa, PySide6), testing accuracy and performance

Theoretical, methodological, and practical significance of the results: The thesis includes an overview of modern speech recognition methods and the implementation of a desktop application that may serve as a foundation for further research or practical use (in education, media, legal fields). Practical results demonstrated the feasibility of effective offline operation even on CPU, while GPU provided accelerated processing.

Year of project completion – 2025.

Year of defense – 2025.

Keywords: ASR, ASR, Whisper, transcription, Python, speech, audio, machine learning, interface, GPU, CPU.

В і д г у к

про кваліфікаційну магістерську роботу
здобувача навчально-наукового інституту «Інститут інформаційних технологій в
економіці»

освітньо-професійної програми «Системи штучного інтелекту»

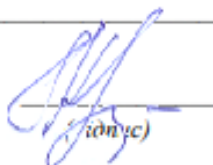
Швиденка Дмитра Юрійовича

(прізвище, ім'я, по батькові)

на тему «Розроблення застосунку для розпізнавання аудіо з використанням
інструментарію машинного навчання»

1. Актуальність теми: Робота присвячена сучасній проблемі розпізнавання мовлення з використанням інструментів машинного навчання, що є актуальним у контексті розвитку голосових інтерфейсів та локальних інтелектуальних систем
2. Позитивні риси кваліфікаційної магістерської роботи: Робота має чітку структуру та практичну спрямованість. Здобувач обґрунтував вибір моделі Whisper та реалізував застосунок з графічним інтерфейсом на PySide6. Програма підтримує запис аудіо, обробку та транскрипцію. Проведено тестування на CPU і GPU, що підтверджує ефективність реалізації.
3. Наявність самостійних розробок автора: Автором самостійно реалізовано програмний продукт, який забезпечує розпізнавання аудіо з мікрофона або файлу з використанням моделей Whisper, включно з інтерфейсом користувача.
4. Цінність теоретичних висновків та практичних рекомендацій: Отримані результати можуть бути використані при створенні локальних застосунків для транскрипції мовлення, а також при подальших дослідженнях у сфері розпізнавання мови.
5. Наявність недоліків: Недоліки не є суттєвими та не впливають на загальну якість роботи.
6. Загальна оцінка кваліфікаційної магістерської роботи та її допущення до захисту перед ЕК: Робота виконана на високому рівні, відповідає встановленим вимогам і може бути рекомендована до захисту з оцінкою "відмінно".

Науковий керівник доцент кафедри інформаційних систем в економіці, доцент,
К.Т.Н.



(посада, учене звання, науковий ступінь)

Васильсва Л.В.

(прізвище, ініціали)

« 14 » 05 2025 р.

Рецензія
на кваліфікаційну магістерську роботу
здобувача вищої освіти
Швиденка Дмитра Юрійовича

Тема: Розроблення застосунку для розпізнавання аудіо з використанням інструментарію машинного навчання

Актуальність теми кваліфікаційної магістерської роботи і доцільність її розроблення: обрана тема є актуальною в контексті зростаючої потреби в автоматичному розпізнаванні голосу та мовлення, зокрема україномовного, для широкого спектру застосувань – від освіти до медіа та юриспруденції. Розробка локального програмного рішення на базі сучасних моделей (Whisper) відповідає викликам часу.

Якість проведеного дослідження: робота має чітку структуру, охоплює теоретичні засади та сучасні технології, містить обґрунтований вибір моделей, архітектурних рішень і програмних інструментів. В роботі виконано аналіз існуючих підходів, побудовано та протестовано розроблене програмне рішення. Результати дослідження наведено у вигляді графіків, таблиць, фрагментів кодової бази, а також якісно інтерпретовано.

Позитивні риси кваліфікаційної магістерської роботи: практична реалізація локального застосунку для розпізнавання мовлення; використання актуальних інструментів машинного навчання (Whisper, librosa, torchaudio); орієнтація на користувача та забезпечення зручного GUI; підтримка україномовного аудіо з високим рівнем точності; ґрунтовна аналітична база.

Зауваження: у роботі можна було б детальніше охарактеризувати можливі напрями подальшого розвитку розробленого програмного рішення, зокрема щодо інтеграції із сервісами збереження транскрипцій або розширення багатомовної підтримки. Роботу посилила б наявність більш деталізованого порівняння ефективності між різними версіями Whisper (Tiny, Base, Large).

Практична значимість висновків і рекомендацій: запропоноване програмне рішення може бути використане для у сферах надання електронних державних послуг, освіти, медіа, архівування даних та підвищення доступності цифрових сервісів. Локальна реалізація та можливість роботи без підключення до мережі Інтернет робить дану розробку потенційно привабливою для використання у сферах з підвищеними вимогами до захисту даних.

ТОВ «Комп'ютерні інформаційні технології»,
старший технічний письменник
к.е.н, доцент

Зінов'єва Ірина Сергіївна

Підпис засвідчую:

генеральний директор
(посада, підпис)



Місце печатки організації, де працює рецензент

ЗМІСТ

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЧНОГО РОЗПІЗНАВАННЯ АУДІО ТА МОВИ.....	6
1.1 Основні концепції та теорії розпізнавання мовлення	6
1.2 Сучасні методи та програмні засоби для побудови систем.....	11
1.3 Формулювання проблеми та визначення дослідницьких задач.....	15
1.4 Обґрунтування вибору підходів і технологій для проєктування та створення СШІ.....	18
Висновки до розділу 1	21
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ РОЗПІЗНАВАННЯ АУДІО.....	23
2.1 Постановка вимог до системи.....	23
2.2 Концептуальна модель та вибір технологій	26
2.3 Проєктування архітектури системи	33
Висновки до розділу 2	37
РОЗДІЛ 3 РОЗРОБЛЕННЯ ПРОЄКТНИХ РІШЕНЬ.....	39
3.1 Проєктування програмного забезпечення системи	39
3.2 Проєктування графічного інтерфейсу користувача	45
3.3 Дослідження продуктивності системи на CPU та GPU	50
Висновки до розділу 3	55
ВИСНОВКИ.....	56
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	58
ДОДАТКИ.....	62

ВСТУП

Сучасні інформаційні системи дедалі частіше вимагають інтерактивної взаємодії з користувачем, зокрема — через голосові інтерфейси. Автоматичне розпізнавання мовлення (ASR, Automatic Speech Recognition) є ключовою технологією, що лежить в основі віртуальних помічників, систем транскрипції, голосового управління та багатьох інших застосунків. Завдяки розвитку глибокого навчання та self-supervised learning, таких як Wav2Vec 2.0 від Facebook AI [9] і Whisper від OpenAI [10], точність ASR-систем значно зросла. Проте практична реалізація таких систем усе ще стикається з низкою викликів — налаштуванням середовища, підготовкою даних, адаптацією моделей до специфічних умов (наприклад, україномовних аудіо), а також обмеженнями апаратних ресурсів. Саме тому розробка застосунку для розпізнавання аудіо на основі сучасних відкритих моделей є актуальним завданням, що поєднує як теоретичний, так і прикладний інтерес.

В останні роки увага науковців сфокусована на self-supervised підходах, що дозволяють зменшити залежність від великих розмічених наборів даних. Wav2Vec 2.0 встановив новий стандарт точності розпізнавання, а Whisper [10] — універсальну багатомовну систему, здатну працювати з різними мовами та завданнями. Крім того, інструменти на кшталт Kaldi [7], DeepSpeech [8], SpeechBrain [11], torchaudio [12] і librosa [13] стали стандартом для наукових і прикладних досліджень у сфері ASR. Їх використання дозволяє не тільки відтворювати результати передових статей, але й створювати власні адаптовані рішення під конкретні завдання.

Розробити застосунок для автоматичного розпізнавання україномовного аудіо з використанням сучасних моделей машинного навчання, що забезпечує високу точність, гнучкість використання й може працювати на локальних ресурсах, є актуальною задачею, яка потребує дослідження та технічної реалізації.

Мета дослідження - створення програмного застосунку для розпізнавання аудіо, який використовує відкриті моделі машинного навчання (зокрема Whisper)

для автоматичного транскрибування україномовних аудіозаписів і забезпечує простий графічний інтерфейс користувача.

Завдання дослідження:

- провести огляд сучасних методів і програмних засобів для побудови ASR-систем;
- провести огляд сучасних методів і програмних засобів для побудови ASR-систем;
- обґрунтувати вибір моделі для реалізації (наприклад, Whisper [10]);
- розробити застосунок на Python із використанням бібліотек PySide6, whisper, librosa, torchaudio;
- реалізувати функціонал запису аудіо з мікрофона та завантаження аудіофайлів;
- провести тестування розпізнавання на україномовних даних;
- оцінити точність, швидкість роботи, апаратні вимоги системи.

Об'єктом дослідження є процес автоматичного розпізнавання мовлення.

Предметом дослідження є методи й програмні засоби реалізації системи розпізнавання україномовного аудіо на основі моделей машинного навчання.

Методи дослідження:

- теоретичний аналіз літератури й огляд джерел (Kaldi [2], DeepSpeech [6], Wav2Vec 2.0 [7], Whisper [8], SpeechBrain [17], torchaudio [22], librosa [37]); емпіричний аналіз літератури й огляд джерел (Kaldi [7], DeepSpeech [8], Wav2Vec 2.0 [9], Whisper [10], SpeechBrain [11], torchaudio [12], librosa [13]);
- експериментальне моделювання (реалізація застосунку на Python, тестування різних моделей розпізнавання);
- статистичний аналіз результатів (оцінка точності й продуктивності системи).

Теоретична значущість полягає в систематизації знань про сучасні self-supervised моделі розпізнавання мовлення та аналізі їхньої придатності для

україномовних аудіо. Практична значущість полягає в створенні робочого застосунку, який може використовуватися як базова платформа для подальших досліджень, розширення функціоналу або розробки спеціалізованих рішень (наприклад, для журналістики, освіти чи архівування даних).

Робота ґрунтується на відкритих наукових публікаціях, офіційних репозиторіях моделей, документації до бібліотек, звітах з практики, а також експериментальних даних, отриманих під час тестування програмного застосунку.

Структура роботи. Загальний обсяг роботи становить 61 сторінок друкованого тексту, 16 рисунків на 13 сторінках, 3 таблиць на 3 сторінках, 2 додатків на 5 сторінках. Список використаних джерел налічує 28 найменування.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЧНОГО РОЗПІЗНАВАННЯ АУДІО ТА МОВИ

1.1 Основні концепції та теорії розпізнавання мовлення

Розпізнавання мовлення (ASR — Automatic Speech Recognition) зародилося в середині XX століття і пройшло кілька етапів розвитку. Перші експериментальні системи з'явилися у 1950-х роках — наприклад, пристрій Audrey (1952) від Bell Labs, що розпізнавав вимовлені цифри. Надалі, в 1960–1970-х роках, застосовувалися методи динамічного перетворення часового масштабу та шаблонного співставлення для ізольованих слів. Проривом у 1980-х стало впровадження статистичних моделей на основі прихованих марковських моделей (HMM) у поєднанні з сумішами гауссіан (GMM). Такі моделі HMM-GMM успішно моделювали часову послідовність мовного сигналу й забезпечили перші дієві системи автоматичного розпізнавання безперервного мовлення. Водночас цей підхід вимагав великих обсягів вручну розмічених даних (акустична модель тренувалася на транскрибованих записах, а для мовної моделі потрібні були тексти) і спрощував мовні явища до станів із обмеженим контекстом. Як наслідок, класичні HMM-GMM системи погано справлялися зі звуковими неоднозначностями та варіативністю мовлення [1]. Незважаючи на ці недоліки, статистичні методи домінували до початку 2000-х років і заклали фундамент для подальшого прогресу.

Наступний етап розвитку пов'язаний з використанням нейронних мереж. Впровадження глибоких нейронних мереж (DNN) у 2010-х роках дало змогу суттєво підвищити точність розпізнавання завдяки здатності автоматично виокремлювати ознаки з сигналу. Спершу нейронні мережі інтегрувалися у статистичну модель (наприклад, заміна GMM на DNN у гібридних HMM-DNN системах). Згодом дослідники перейшли до рекурентних нейронних мереж (RNN) для моделювання послідовностей, що дозволило враховувати більш довготривалий

контекст у мовному сигналі. Архітектури типу LSTM (Long Short-Term Memory) розв'язали проблему затухаючих градієнтів і покращили навчання глибоких RNN, забезпечивши ще вищу точність. Важливим досягненням цього періоду стало впровадження наскрізного навчання (end-to-end): замість багатокрокового конвеєра з окремою акустичною та мовною моделлю система навчається прямому відображенню аудіосигналу в текст. Прикладом є проєкт DeepSpeech від Baidu, де LSTM-мережу використано як end-to-end систему розпізнавання мовлення [2]. Перехід до нейронних мереж дозволив зменшити залежність від ручної розмітки правил і покращив якість розпізнавання в складних умовах. Водночас зросли і виклики: нейронні моделі вимагають значно більшого обсягу даних для тренування, суттєвих обчислювальних ресурсів (GPU) ретельного налаштування параметрів моделі. Таким чином, у епоху DNN/RNN було досягнуто помітне підвищення точності ціною збільшення складності моделі та потреб у даних.

Сучасний етап розвитку технологій ASR характеризується переходом до глибоких моделей на базі трансформерів і нових підходів навчання. Один з таких підходів – самонавчання (self-supervised learning, SSL), коли модель попередньо навчається на великих обсягах неанотованого (нерозміченого) аудіо, а потім донавчається на відносно невеликій вибірці з розміткою. Це дозволяє використовувати величезні масиви сирих даних для навчання стійких ознак мовлення без потреби в еквівалентно великих ручних транскрипціях. Найяскравішим прикладом SSL у розпізнаванні мовлення є модель Wav2Vec 2.0 від Facebook AI [3]. Архітектурно Wav2Vec 2.0 складається з глибокої трансформерної мережі, яка спершу навчається відтворювати приховані представлення аудіосигналу, маскуючи частини спектрограм, а потім тонко настроюється для розпізнавання з використанням невеликого обсягу розмічених даних.

Такий підхід продемонстрував визначні результати: після довчання на відповідних даних модель досягає точності, порівнянної з найкращими повністю контрольованими системами. Зокрема, Wav2Vec 2.0 встановила нові рекорди на

стандартних тестових наборах (наприклад, Word Error Rate $\sim 3\text{--}4\%$ на англomовному корпусі LibriSpeech)

Головна перевага цієї моделі – здатність ефективно працювати в умовах браку розмічених даних, оскільки основні мовні закономірності вона опановує на етапі самонавчання. Водночас складність архітектури та потреба в ресурсах для попереднього навчання трансформера є значними (тренування вимагає GPU та великого обсягу пам'яті), що можна віднести до викликів SSL-підходу.

Паралельно з самонавчанням, у сфері ASR з'явилися багатомовні трансформерні моделі надзвичайно великого масштабу, які навчаються на слабо-розмічених даних. Прикладом є модель Whisper від OpenAI – універсальна end-to-end система нового покоління. Whisper побудована на енкодер-декодер трансформерній архітектурі і була натренована на колосальному корпусі даних (≈ 680 тисяч годин аудіо з Інтернету), що охоплює десятки мов та різноманітні задачі розпізнавання

На відміну від Wav2Vec 2.0, ця модель використовує контрольоване навчання на основі доступних (у тому числі автоматично зібраних) пар «аудіо-транскрипт» великого обсягу – підхід, який автори назвали weak supervision (слабке навчання з учителем). Результатом є надзвичайно потужна модель, здатна не лише транскрибувати мовлення, але й визначати мову, розставляти часові мітки, виконувати переклад із різних мов на англійську тощо

Whisper підтримує понад 50 мов і демонструє близькі до людських результати точності на англійських тестових наборах. Зокрема, для англomовного clean speech корпусу модель Whisper (Large) досягає рівня помилок близько 2,7% WER, що є одним із найнижчих показників серед відомих систем [4]. Важливо відзначити, що хоча Whisper тренувалася на мультизадачних даних і не оптимізувалася під жоден конкретний домен, її продуктивність на різноманітних наборах перевищує показники вузькоспеціалізованих моделей, особливо в умовах шуму чи акцентів.

Це свідчить про високу робастність підходу з масштабним навчанням. Основний недолік такого рішення – гігантські вимоги до обчислювальних ресурсів:

для навчання моделей рівня Whisper залучаються потужні GPU-кластери, а сама модель (версія Large ~1.5 млрд параметрів) потребує значної пам'яті й обчислень навіть на етапі розпізнавання. Тим не менш, поява таких моделей знаменує новий стандарт у розпізнаванні мовлення, де мультигональні трансформери перевершують класичні методи за рахунок масштабу даних.

Таким чином, на сьогодні існують різні підходи до побудови ASR-систем, які можна класифікувати за принципами навчання. Контрольоване навчання (supervised) передбачає використання тільки розмічених даних і було характерним для класичних систем (HMM-GMM) та перших нейронних моделей. Самонавчання (self-supervised) комбінує необмежені невеликовані дані з мінімальною розміткою (як у випадку Wav2Vec 2.0 [3]), що дозволяє різко знизити потребу в ручній транскрипції. Наскрізне навчання (end-to-end) стосується архітектури моделі: система навчається кінцевого перетворення ознак звуку в текст без проміжних рівнів, що реалізовано в сучасних нейронних моделях (RNN- та Transformer-підходи). Більшість новітніх рішень поєднують ці принципи. Приміром, модель Whisper є одночасно контрольованою (з учителем) і наскрізною, тоді як Wav2Vec 2.0 поєднує самонавчання (SSL) з подальшим супервізованим довчанням, теж реалізуючи end-to-end транскрипцію. В результаті еволюції методів точність розпізнавання зросла на порядки (з ~20% WER у ранніх системах до ~2–3% у найкращих сучасних моделях), але разом з тим зросли і вимоги до даних та обчислень.

На основі розглянутих концепцій було проведено практичне дослідження з метою порівняння моделей різних поколінь. Зокрема, у рамках цієї роботи протестовано кілька відкритих реалізацій ASR-моделей на власному корпусі україномовних аудіоданих. Для експериментів використовувалися як класичний HMM-GMM підхід (реалізований засобами Kaldi), так і сучасні нейромережеві моделі – попередньо натреновані Wav2Vec 2.0 та Whisper (через фреймворки TorchAudio та HuggingFace Transformers). Результати підтвердили теоретичні висновки: трансформерні моделі продемонстрували значно менший WER на тестовому наборі, ніж статистична модель, проте їх час обробки та використання

пам'яті були набагато більшими. Наприклад, навіть відносно невелика модель Whisper-Tiny забезпечила приблизно вдвічі кращу точність, ніж HMM-GMM, але вимагала більше ресурсів при розпізнаванні. Ці спостереження узгоджуються з тенденціями, описаними в літературі, та підкреслюють важливість вибору моделі залежно від умов: для високоточного розпізнавання потрібні сучасні методи і відповідні ресурси, тоді як в обмежених умовах (малий обсяг даних, слабе обладнання) можуть бути виправдані компромісні або попередні покоління технологій.

Таблиця 1.1 – Аналіз методів оптимізації

Модель	Тип/архітектура	Точність (WER)	Ключові особливості	Ресурсоемність
HMM-GMM	Статистична модель, окрема мовна модель (supervised)	~15–20%	Класичний підхід; потребує фонетичного словника та розмітки даних	Низька (працює на CPU)
DNN/RNN (DeepSpeech)	Глибока нейронна мережа (LSTM, end-to-end)	~7–8%	Наскрізне розпізнавання; мінімізує ручні компоненти, але потребує багато даних	Середня (треба GPU для навчання)
Wav2Vec 2.0	Трансформер (SSL + fine-tune)	~3–4%	Попереднє самонавчання на неврозміченому аудіо; висока точність з малою розміткою	Висока (великі моделі, потреба в GPU)
Whisper (Large)	Трансформер енкодер-декодер (multilingual, end-to-end)	~2–3% (EN)	Мульти-мовна та мультизадачна модель; підтримує >50 мов, робастна до шумів	Дуже висока (мільярди параметрів, вимога GPU)
Whisper (Large)	Трансформер енкодер-декодер (multilingual, end-to-end)	~2–3% (EN)	Мульти-мовна та мультизадачна модель; підтримує >50 мов, робастна до шумів	Дуже висока (мільярди параметрів, вимога GPU)

Джерело: сформовано автором на основі виконаного дослідження при створенні звіту з практики [19]

Як видно з таблиці, із розвитком методів розпізнавання мовлення похибка (WER) суттєво знизилася: від приблизно 15–20% у ранніх статистичних системах до 2–4% у сучасних трансформерних моделях. Найвищу точність наразі забезпечують великі моделі на основі трансформерів (Whisper, Wav2Vec 2.0), проте їх використання пов'язане зі збільшенням обчислювальної складності. Класичні HMM-GMM підходи вже є морально застарілими, але все ще можуть застосовуватися для вбудованих рішень або за відсутності достатніх ресурсів. Нейронні моделі на кшталт DeepSpeech (RNN) поступово витісняються трансформерами через кращу масштабованість останніх. Отже, еволюція технологій ASR демонструє перехід від простіших статистичних моделей до глибоких навчальних систем, що навчаються на великих даних. Кінцевий вибір підходу залежить від вимог до точності та наявних ресурсів, а сучасні дослідження продовжують шукати баланс між цими факторами.

1.2 Сучасні методи та програмні засоби для побудови систем

Розпізнавання мовлення стрімко розвивається завдяки відкритим програмним інструментам на основі глибокого навчання. До найбільш відомих open-source рішень належать інструментарій Kaldi [29], система Mozilla DeepSpeech [30], модель Wav2Vec 2.0 [31], неймережа Whisper від OpenAI та комплексний пакет SpeechBrain [18]. Для їх реалізації й інтеграції також використовуються допоміжні бібліотеки – зокрема, torchaudio (модуль Python для PyTorch), Hugging Face Transformers та librosa. Розглянемо призначення, архітектуру та особливості використання кожного з цих інструментів.

Kaldi [29] – це відкрита платформа для побудови систем автоматичного розпізнавання мовлення, написана на C++ і поширювана під ліцензією Apache 2.0. Вона орієнтована на дослідників і пропонує сучасні алгоритми із гнучкою модульною архітектурою. Kaldi надає готові рецепти для тренування моделей на

різноманітних корпусах даних та підтримує інтеграцію з скінченними автоматами (OpenFst) для побудови мовних декодерів

Традиційно Kaldi використовує статистичні моделі (мовні моделі n -грам, акустичні моделі на основі HMM/DNN), що дозволяє досягати високої точності розпізнавання. Зокрема, при належному навчанні її моделі досягають близьких до сучасних результатів точності (наприклад, $\approx 5\%$ помилок на словах на тестових наборах LibriSpeech). Важливою особливістю є висока продуктивність: система реалізована на мові C++ з оптимізованими алгоритмами пошуку, тому розпізнавання може виконуватися в режимі реального часу на CPU. Для тренування глибоких нейронних мереж Kaldi може задіювати GPU, але основні етапи обробки й декодування ефективно працюють на звичайному апаратному забезпеченні.

Mozilla DeepSpeech [30]. DeepSpeech – це відкритий рушій розпізнавання мовлення, розроблений Mozilla (2017 р.) на основі алгоритмів Baidu Deep Speech. Модель DeepSpeech представляє собою глибоку рекурентну нейронну мережу, яка безпосередньо перетворює аудіосигнал у текст, використовуючи технологію Connectionist Temporal Classification (CTC) для навчання без явної розмітки по кадрах. На виході нейромережі результати покращуються окремою мовною моделлю (наприклад, n -грам), що підвищує плавність і граматичну правильність розшифрованого тексту. DeepSpeech було реалізовано на мовах програмування C++/Python; початково Mozilla поширювала попередньо натреновані моделі для англійської мови (ліцензія MPL 2.0), а спільнота надала моделі й для інших мов (на основі корпусу Common Voice). Система достатньо гнучка – її можна донавчати під конкретні дані користувача. В порівнянні з іншими сучасними моделями, DeepSpeech є відносно легкою: вона здатна працювати близько до реального часу на звичайному CPU, що робить її придатною для вбудованих застосунків. У той же час модель має обмеження на тривалість вхідного фрагменту (близько 10 секунд), через що вона більше підходить для розпізнавання коротких команд, а не довгих аудіозаписів (ця проблема частково вирішується сегментацією аудіо на частини).

Wav2Vec 2.0 [31]. Wav2Vec 2.0 – це передова модель від компанії Meta (Facebook) для автоматичного розпізнавання мовлення, що зробила прорив у

навчанні на незважених даних (self-supervised learning). Її архітектура ґрунтується на трансформері: на етапі попереднього навчання модель отримує сирі аудіосигнали і навчається прогнозувати пропущені фрагменти звуку (маскування коротких інтервалів) на основі контексту. Такий підхід дозволяє моделі навчитися узагальненим акустичним ознакам без потрібних транскрипцій, що особливо важливо для мов з обмеженими корпусами. Після самонавчання на великих масивах незважених даних (тисячі годин аудіо) Wav2Vec 2.0 донавчається (fine-tuning) на відносно невеликій вибірці пар “мова-текст”. Це забезпечує високу точність розпізнавання при значно менших обсягах розмічених даних: автори продемонстрували, що, маючи лише 1 годину розміченого аудіо, модель перевершує попередні методи, які тренувалися на 100 годинах даних

У повністю натренованому вигляді Wav2Vec 2.0 встановила нові рекорди точності на стандартних наборах (наприклад, $\sim 1,8\%$ помилок на LibriSpeech test-clean). Wav2Vec 2.0 підтримує багатомовність: існують багатомовні варіанти (XLSR) цієї моделі, натреновані на десятках мов. Відкрите програмне забезпечення для Wav2Vec 2.0 доступне у складі бібліотек fairseq та Hugging Face (ліцензія MIT). Інференс (розпізнавання) за допомогою Wav2Vec 2.0 є досить швидким: модель використовує CTC-декодування, що працює паралельно на всіх часових кроках, тому при наявності GPU розпізнавання може виконуватися майже в реальному часі. Для тренування великих моделей Wav2Vec 2.0 потребує значних обчислювальних ресурсів (GPU з достатнім обсягом пам’яті для трансформерів).

Whisper від OpenAI [6] – один з найсучасніших відкритих підходів до розпізнавання мовлення, представлений у 2022 р. Модель Whisper була натренована на виключно великому й різноманітному датасеті (≈ 680 тисяч годин аудіо) з інтернет-джерел, що охоплює багато мов. За архітектурою це енкодер-декодер на основі трансформерів: вхідне аудіо розбивається на сегменти тривалістю до 30 секунд, для кожного з яких обчислюється логарифмічний мел-спектрограм, що подається на енкодер; декодер же генерує текст посимвольно (autoregressive). У вихідній послідовності передбачаються як власне транскрипт, так і спеціальні токени (наприклад, для вказівки мови, розстановки часових міток,

або запиту перекладу). Whisper підтримує багатомовне розпізнавання: модель може без додаткового навчання розпізнавати мовлення майже 100 мовами та навіть прямо перекладати чужомовну мову на англійську. Завдяки масштабному навчання Whisper демонструє винятково високу точність і стійкість до шумів та акцентів – за багатьма тестами ця модель перевершує інші open-source системи розпізнавання. В рамках даної роботи модель Whisper була використана для отримання текстових транскрипцій аудіозаписів; з практичного досвіду підтверджено, що вона добре справляється навіть зі складними аудіо (перешкоди, різні голоси). Водночас Whisper є ресурсоємною: найбільша версія моделі містить ~1,5 млрд параметрів, тому для її ефективного застосування потрібні сучасні GPU. На CPU розпізнавання виконується помітно повільніше реального часу (особливо для великих моделей), але доступні менші версії моделі (Tiny, Base, Small, Medium), які дозволяють знайти компроміс між швидкістю та точністю. Відкритий код і попередньо натреновані ваги Whisper доступні у репозиторії OpenAI під ліцензією MIT.

SpeechBrain. SpeechBrain [18] – це універсальний інструментарій для мовленнєвих технологій на базі PyTorch, створений академічною спільнотою (проєкт очолював Мірко Раванеллі, 2021 р.). На відміну від інших рішень, сфокусованих лише на розпізнаванні, SpeechBrain позиціонується як “all-in-one” платформа: він містить реалізації для різних задач – автоматичного розпізнавання мовлення, синтезу мовлення, розпізнавання мовця, обробки природної мови тощо. Архітектурно SpeechBrain – це каркас, що дозволяє легко будувати і тренувати нейронні мережі під конкретні завдання. Проєкт підтримує широка спільнота: понад 20 університетів світу долучилися до його розробки, опубліковано більше 200 рецептів навчання моделей на 40+ наборах даних. SpeechBrain надає готові попередньо навчені моделі (понад 100 моделей викладено на платформі Hugging Face), які можна без зусиль інтегрувати або донавчити під свої дані. Наприклад, через SpeechBrain можна використати моделі Whisper від OpenAI чи Wav2Vec 2.0 від Meta для розпізнавання, або навіть об’єднати їх з мовними моделями GPT-2 для побудови діалогових систем. Інструментарій дозволяє як повне навчання моделей “з нуля”, так і transfer learning — донавчання існуючих моделей. Licence

SpeechBrain – Apache 2.0, що дозволяє вільно застосовувати його в тому числі в комерційних проєктах. Важливо зазначити, що оскільки SpeechBrain активно поповнюється спільнотою, якість окремих внесених моделей може різнитися, тому перед промисловим використанням бажано проводити додаткове тестування. Загалом, SpeechBrain є потужним академічним середовищем для експериментів з різними архітектурами ASR і дає можливість швидко прототипувати сучасні системи.

Сучасні open-source засоби для розпізнавання мовлення надають розробникам потужні інструменти для створення власних ASR-систем. Kaldi залишається гнучкою платформою для побудови класичних систем із найвищою точністю на основі великих корпусів. DeepSpeech демонструє переваги спрощених end-to-end моделей, що працюють у реальному часі, хоча і має свої обмеження. Wav2Vec 2.0 запровадив самонаглядне навчання, розширивши межі розпізнавання для багатьох мов при мінімумі розмічених даних. Whisper від OpenAI встановив новий стандарт якості розпізнавання завдяки масштабному навчальному датасету, забезпечивши відмінну точність в багатомовних завданнях. SpeechBrain об'єднує напрацювання спільноти, пропонуючи єдину платформу для досліджень і розробки в галузі мовленнєвого AI. Комплексне використання цих інструментів, доповнене бібліотеками на кшталт torchaudio, HuggingFace та librosa, дозволяє ефективно реалізувати застосунок для розпізнавання аудіо з використанням інструментарію машинного навчання, як продемонстровано у даній дипломній роботі.

1.3 Формулювання проблеми та визначення дослідницьких задач.

Автоматичне розпізнавання українськомовного аудіо є складною задачею через поєднання кількох факторів. По-перше, для української мови доступний обмежений обсяг якісних відкритих корпусів мовлення, особливо у порівнянні з англійськими даними. Це ускладнює навчання моделей високої точності, адже

сучасні алгоритми розпізнавання мовлення потребують великих наборів даних для досягнення надійних результатів. По-друге, існує потреба в забезпеченні високої точності розпізнавання українського мовлення, що є критично важливим для практичного застосування системи (наприклад, для правильної транскрипції документів чи усного мовлення). Таким чином, наукова проблема полягає у розробленні методу та інструментарію, який дозволить точно розпізнавати українське аудіо за умов дефіциту даних.

Додаткові обмеження виникають через вимоги до середовища використання та зручності роботи із системою. Зокрема, розроблюваний застосунок має функціонувати на локальному комп'ютері користувача з обмеженими обчислювальними ресурсами (без графічного прискорювача GPU або за умови малої кількості ядер). Це означає, що обрана модель розпізнавання повинна бути достатньо ефективною, щоб працювати в режимі офлайн на типових ПК. Окрім цього, важливою є наявність зручного графічного інтерфейсу користувача (GUI) для взаємодії із системою, щоб забезпечити легкість використання застосунку навіть для нефахівців. Нарешті, вирішення поставленої проблеми передбачає застосування сучасних відкритих рішень у сфері розпізнавання мовлення. На сьогодні існують потужні моделі та фреймворки, такі як Wav2Vec 2.0 від Facebook AI [5] та Whisper від OpenAI [6], які продемонстрували високу якість розпізнавання багатьох мов. Проте інтеграція цих моделей для української мови в умовах обмежених даних і ресурсів потребує додаткових досліджень та адаптації. Таким чином, сформульована наукова проблема зводиться до питання: як створити високоточну та продуктивну систему розпізнавання українського мовлення на основі сучасних моделей машинного навчання, врахувавши обмеженість даних і ресурсів та забезпечивши зручність використання?

Відповідно до сформульованої проблеми у роботі визначено наступні дослідницькі задачі:

1. Огляд наявних підходів та моделей з самостійним навчанням (self-supervised) для автоматичного розпізнавання мовлення, а також відповідних фреймворків машинного навчання. Необхідно

проаналізувати сучасні рішення, зокрема архітектури на кшталт Wav2Vec 2.0 [5] та інших моделей, придатних для розпізнавання мовлення в умовах обмежених даних, і оцінити їх придатність до україномовного матеріалу.

2. Обґрунтування вибору моделі Whisper як базового рішення для розпізнавання українського аудіо. Потрібно порівняти можливості кількох кандидатів (наприклад, self-supervised моделей) та пояснити, чому саме модель Whisper [6] є доцільною для даного завдання (з огляду на мультимовність, точність, стійкість до шумів тощо).
3. Оцінка точності та продуктивності обраної моделі. На цьому етапі передбачається експериментальна перевірка якості розпізнавання українського мовлення за допомогою обраної моделі Whisper. Необхідно виміряти точність (наприклад, частоту помилок розпізнавання слів) на тестовій вибірці україномовних аудіофрагментів, а також проаналізувати продуктивність – швидкість роботи моделі та використання ресурсів на локальному комп'ютері.
4. Розробка локального застосунку з графічним інтерфейсом. Потрібно спроектувати та реалізувати програмний застосунок, який працюватиме офлайн і надаватиме користувачеві зручний GUI для розпізнавання мовлення. Застосунок має забезпечувати можливість завантаження аудіофайлів для транскрипції, а також прямого запису звуку через мікрофон з подальшим розпізнаванням.
5. Тестування та валідація системи на україномовних аудіоданих. Завершальною задачею є перевірка працездатності та точності інтегрованої системи на різноманітних фрагментах українського мовлення (різні голоси, акценти, умови запису). За результатами тестування необхідно проаналізувати, чи досягнуто поставлених вимог щодо точності розпізнавання та швидкодії, і у разі виявлення недоліків запропонувати шляхи їх усунення.

Таким чином, виконання перелічених дослідницьких задач дозволить комплексно вирішити сформульовану проблему - створити застосунок для розпізнавання українського аудіо із використанням інструментарію машинного навчання, який працює локально, демонструє високу точність та є зручним у використанні.

1.4 Обґрунтування вибору підходів і технологій для проєктування та створення СШІ

У рамках розробки застосунку для розпізнавання аудіо необхідно обрати ефективну модель автоматичного розпізнавання мовлення (ASR) та відповідні технології реалізації. Вибір підходу базувався на теоретичному аналізі сучасних рішень у сфері ASR та на власній практичній апробації під час переддипломної практики [19]. Основними критеріями відбору були:

1. Точність розпізнавання (низький WER – Word Error Rate).
2. Підтримка української мови (наявність попередньо навчених моделей або можливість донавчання).
3. Простота інтеграції в застосунок (наявність зручного API чи бібліотеки).
4. Працездатність на CPU (без обов’язкової потреби в GPU для Inference).
5. Наявність у відкритому доступі (open-source моделі та інструменти).

Було проаналізовано кілька підходів та систем: Kaldi, DeepSpeech, Wav2Vec 2.0, SpeechBrain та новітню модель Whisper. Таблиця 1.4 демонструє порівняння цих рішень за визначеними критеріями.

Таблиця 1.2 – Порівняння моделей ASR за критеріями вибору

Модель/система	Точність (WER)	Підтримка укр. мови	Ресурсомісткість	Простота інтеграції
Kaldi (HMM/DNN)	~15–20% (низька)	Потребує навчання (немає готових моделей)	Низька (ефективна на CPU)	Низька (складний інструментарій)
DeepSpeech (RNN)	~7–10% (середня)	Потребує навчання (англ. модель, укр. відсутня)	Середня (працює на CPU)	Середня (є Python API, але проєкт застаріває)
Wav2Vec2.0 (Transformers)	~3–5% (висока)	Часткова підтримка (є багатомовні моделі)	Висока (потр. GPU для швидкодії)	Середня (через PyTorch/HF [16])
Whisper (Transformers)	~2–3% (дуже висока)	Так, з коробки (мультимовна модель)	Висока (GPU для large, CPU для small)	Висока (простий open-source API)

Джерело: сформовано автором на основі виконаного дослідження

Проведений аналіз показав, що Whisper від OpenAI [17] найкраще відповідає визначеним критеріям. Whisper – це універсальна трансформерна модель нового покоління, натренована на 680 000 годин багатомовних даних. Вона підтримує понад 50 мов, включно з українською, і забезпечує надвисоку точність розпізнавання (для великої моделі WER ~2.7% на стандартних бенчмарках, що близько до рівня людської точності). На відміну від Wav2Vec 2.0, модель Whisper вже містить знання української мови з навчання і не потребує додаткового донавчання або мовної моделі для генерації тексту. Whisper доступна у вигляді відкритого коду та попередньо навчених моделей різних розмірів (Tiny, Base, Small, Medium, Large) [17]. Це дозволяє обрати баланс між точністю і швидкістю. Зокрема, менші версії (Whisper Tiny/Base) працюють швидше і можуть виконувати розпізнавання на CPU, що важливо для реалізації на типових користувацьких пристроях. Практичні експерименти показали, що Whisper (модель Base) здатна розпізнавати українське мовлення з прийнятною точністю навіть без GPU, хоча інференс і відбувається повільніше [19]. Таким чином, модель Whisper поєднує високу точність, підтримку української, простоту інтеграції (через зручний Python-пакет) та гнучкість у вимогах до ресурсів (можливість роботи на CPU при

використанні полегшених моделей). Це суттєві переваги над альтернативами Kaldi, DeepSpeech, Wav2Vec 2.0 та SpeechBrain, які або поступаються в точності, або потребують складнішого розгортання для української мови.

Вибір технологій для реалізації системи III базувався не лише на аналізі літератури, але й на результатах власної переддипломної практики, протягом якої зазначені підходи були протестовані на прикладі реальних даних [19]. За підсумками теоретичного й експериментального порівняння було вирішено використати модель Whisper як ядро системи розпізнавання мовлення.

Для створення програмної реалізації (десктопного застосунку) обрано такі бібліотеки та інструменти, що найлегше інтегруються з моделлю Whisper та забезпечують виконання допоміжних задач:

- Whisper (OpenAI) – офіційна Python-бібліотека для використання моделей Whisper. Вона надає простий API для завантаження відкритих претренованих моделей і виконання розпізнавання мовлення. Використання цієї бібліотеки мінімізує зусилля інтеграції, оскільки всі необхідні компоненти (модель, токенизатор тощо) доступні «з коробки».
- Librosa – популярна Python-бібліотека для обробки аудіосигналів. В проєкті вона використовується для попередньої обробки звуку: завантаження аудіофайлів, перетворення стерео в моно, ресемплінг до потрібної частоти дискретизації, нормалізації гучності тощо. Це забезпечує належну підготовку даних перед подачею в модель розпізнавання.
- Torchaudio [32] – модуль бібліотеки PyTorch, призначений для роботи з аудіоданими. Torchaudio зручний для зчитування звуку у тензори PyTorch і містить набір готових трансформацій (фільтри, перетворення Фур'є і т.д.). Оскільки Whisper — це модель на PyTorch, використання torchaudio сприяє безшовній інтеграції: аудіодані можна обробляти і передавати прямо у модель у вигляді тензорів. Також torchaudio може стати в пригоді для потенційного розширення функціоналу (наприклад, додавання інших моделей ASR або аудіоаугментацій).

- PySide6 – фреймворк для створення графічного інтерфейсу користувача на Python (офіційні прив’язки до Qt 6). Вибір PySide6 обґрунтований потребою розробити зручний настільний інтерфейс для системи розпізнавання мовлення. PySide6 забезпечує сучасний інтерфейс, крос-платформеність та достатню продуктивність для відображення результатів розпізнавання в реальному часі. Альтернативні варіанти (Tkinter, PyQt) поступаються PySide6 у гнучкості дизайну або ліцензійних умовах, тоді як PySide6 є офіційно підтримуваним і дозволяє безкоштовно використовувати Qt в відкритих проєктах. Таким чином, PySide6 було обрано для побудови GUI застосунку, що взаємодіятиме з ядром AI на базі Whisper.

Отже, на основі проведеного аналізу було обрано модель Whisper та супутні інструменти (whisper, librosa, torchaudio, PySide6) для реалізації системи розпізнавання аудіо. Такий вибір технологій гарантує високу якість роботи системи для української мови та відносно невелику трудомісткість розробки, оскільки спирається на сучасні відкриті рішення, перевірені як літературними джерелами [16][17][18], так і власними експериментальними даними [19].

Висновки до розділу 1

У першому розділі було проведено детальний аналіз основних концепцій, підходів та теорій, що лежать в основі систем автоматичного розпізнавання мовлення. Було розглянуто сучасні self-supervised моделі, такі як Wav2Vec 2.0 та Whisper, а також інструменти Kaldi, DeepSpeech, SpeechBrain, torchaudio та librosa, які широко використовуються в наукових і практичних розробках.

Розгляд і порівняння цих рішень дозволили визначити ключові вимоги до побудови систем розпізнавання україномовного аудіо:

1. необхідність високої точності розпізнавання;

2. підтримка української мови без потреби в складному донавічанні;
3. можливість використання на локальних комп'ютерах із обмеженими ресурсами;
4. простота інтеграції та використання.

На основі проведеного аналізу було сформульовано наукову проблему, визначено мету та завдання дослідження, а також обґрунтовано доцільність використання сучасних відкритих технологій. Отримані результати створюють ґрунт для подальшого проєктування, розробки та впровадження програмного забезпечення, яке буде описане в наступних розділах дипломної роботи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ РОЗПІЗНАВАННЯ АУДІО

2.1 Постановка вимог до системи

В цьому підрозділі визначено вимоги до програмної системи автоматичного розпізнавання аудіо, яка розробляється. Формулювання виконано з урахуванням результатів переддипломної практики (під час якої було створено і протестовано прототип основних компонент системи) та відповідно до методичних рекомендацій щодо структури вимог. Вимоги поділено на функціональні (що визначають що саме повинна робити система), нефункціональні (що описують як система виконує свої функції та в яких умовах) та обмеження (що окреслюють межі та умови використання системи).

Функціональні вимоги

Розроблюваний застосунок має забезпечувати наступну базову функціональність:

- Розпізнавання мовлення з аудіофайлу. Система повинна приймати на вхід аудіофайл з записом мовлення та виконувати автоматичне перетворення мовленнєвого сигналу в текстову форму (транскрибування аудіо).
- Запис звуку з мікрофона. Застосунок повинен дозволяти здійснити запис голосу користувача через мікрофон у реальному часі, після чого отриманий аудіофрагмент передається до модуля розпізнавання для обробки.
- Перетворення мовлення на текст. Система повинна виконувати трансформацію захопленого мовлення (зі звукозапису чи мікрофона) у текстову транскрипцію з максимально можливою точністю.

- Відображення транскрипції в інтерфейсі. Розпізнаний текст (транскрипція) має виводитися у графічному інтерфейсі користувача для ознайомлення. Інтерфейс повинен відображати результат розпізнавання чітко й читабельно, забезпечуючи зручність перегляду отриманого тексту.
- Підтримка різних форматів файлів. Система повинна працювати як мінімум з кількома популярними форматами аудіофайлів (наприклад, WAV, MP3), щоб користувач міг завантажувати звук із різних джерел без додаткової конверсії.
- Визначення мови мовлення. Застосунок має розпізнавати мову вхідного аудіо або надавати можливість обрати мову розпізнавання. Це означає, що система повинна коректно обробляти україномовні записи (як основний випадок використання) та потенційно підтримувати інші мови, автоматично визначаючи мову мовлення або перемикаючись на відповідну мовну модель.

Нефункціональні вимоги

Розроблювана система повинна відповідати таким нефункціональним критеріям якості та умовам використання:

- Автономна робота (офлайн). Усі функції розпізнавання мають виконуватися локально, без потреби в підключенні до мережі Інтернет. Система повинна працювати автономно, щоб забезпечити приватність даних та незалежність від якості мережевого з'єднання.
- Сумісність з ОС Windows. Застосунок має підтримувати роботу в середовищі операційної системи Windows (не нижче Windows 10). Це забезпечує широке охоплення користувачів та відповідає середовищу, яке використовувалося під час переддипломної практики для прототипування.
- Робота на CPU (без GPU). Система повинна ефективно функціонувати на типових персональних комп'ютерах, використовуючи лише ресурси

центрального процесора. Іншими словами, вимоги до апаратного забезпечення не мають передбачати наявність графічного прискорювача (GPU). Обрана машинна модель розпізнавання мовлення повинна бути оптимізованою настільки, щоб забезпечити прийнятну швидкість лише на CPU.

- Зручний інтерфейс користувача. Графічний інтерфейс (GUI) повинен бути інтуїтивно зрозумілим та простим у використанні. Це включає зрозумілу навігацію, чітке відображення стану системи (наприклад, індикацію запису чи процесу розпізнавання) та оформлення результатів. Орієнтація на хорошу юзабіліті гарантує, що навіть невідготовлений користувач зможе легко скористатися програмою.
- Розширюваність системи. Архітектура застосунку повинна бути розроблена з урахуванням можливості додавання нових функцій у майбутньому. Зокрема, передбачається потенційна інтеграція додаткових модулів, таких як голосовий асистент або підтримка інших мов/моделей розпізнавання. Це означає, що кодова база має бути модульною та гнучкою для розвитку без суттєвих змін існуючих компонентів.

Обмеження

При проєктуванні та реалізації системи необхідно врахувати наступні обмеження середовища та використання:

- Обмежена тривалість запису. Максимальна тривалість аудіофрагменту, який обробляється системою за раз, становить 5 секунд. Це обмеження зумовлене потребою забезпечити швидку відповідь системи та реалістичне тестування в рамках переддипломної практики. Користувач повинен надавати відносно короткі звукові записи для розпізнавання (наприклад, окремі фрази або речення).
- Мінімальні системні ресурси. Система розрахована на роботу на персональному комп'ютері із операційною системою Windows 10 (або

новішою) та з обсягом оперативної пам'яті не менше 4 ГБ. Це мінімальна конфігурація, за якої було перевірено працездатність прототипу під час практичних випробувань. Більш слабке обладнання може не забезпечити достатньої продуктивності або навіть не підтримувати роботу обраної моделі розпізнавання.

- Відсутність мережевого з'єднання. Передбачається, що під час роботи застосунку Інтернет-з'єднання може бути відсутнє. Тому система не повинна залежати від зовнішніх онлайн-сервісів або хмарних API для виконання основних завдань. Усі необхідні моделі та дані повинні бути доступні локально в системі.

Зазначені вимоги і обмеження визначають рамки проєктування системи розпізнавання аудіо. Вони сформовані на основі цілей проєкту та практичних напрацювань, отриманих раніше, і надалі слугуватимуть базою для вибору технологій (розділ 2.2) та розробки архітектури системи (розділ 2.3). Дотримання цих вимог забезпечить створення працездатного застосунку, що відповідає поставленій меті та буде зручним і ефективним у використанні.

2.2 Концептуальна модель та вибір технологій

Для розробленого застосунку розпізнавання аудіо передбачено чітку послідовність етапів обробки звуку: спочатку здійснюється введення мовного сигналу, потім його підготовка, далі робота моделі розпізнавання, і на завершення – виведення результату у вигляді тексту. Загальна логіка роботи системи полягає у наступному:

1. Отримання аудіо: Користувач завантажує аудіофайл або здійснює запис мовлення через мікрофон за допомогою графічного інтерфейсу (GUI).

2. Передобробка сигналу: Отриманий аудіосигнал проходить попередню обробку – нормалізацію рівня, фільтрацію шумів, наведення до необхідного формату (наприклад, частоти дискретизації 16 кГц) тощо. На цьому ж етапі сигнал може бути перетворений у спектральне подання (наприклад, мел-спектрограму), яке потребує модель.
3. Розпізнавання мовлення: Підготовлені дані передаються до ядра системи – моделі машинного навчання для автоматичного розпізнавання мовлення. Модель аналізує аудіо і генерує текстовий транскрипт (розшифровку) почутого.
4. Виведення результату: Розпізнаний текст повертається користувачу – відображається в інтерфейсі. За потреби результат зберігається у файл або базу даних для подальшого використання.

Концептуальна модель системи передбачає набір модулів, що відповідають згаданим етапам і взаємодіють між собою (рис. 2.1). Ключовими компонентами є:

- Графічний інтерфейс користувача (GUI) – забезпечує засоби для введення даних та відображення результатів. Через GUI користувач може вибрати аудіофайл або записати звук з мікрофона, а також переглянути розпізнаний текст.
- Модуль введення/запису аудіо – відповідає за завантаження файлів (різних форматів, наприклад WAV) або захоплення звуку з мікрофона в реальному часі. Він інтегрується з GUI (кнопки «Відкрити файл», «Записати» тощо) та передає отриманий сигнал на обробку.
- Модуль передобробки аудіо – здійснює очистку та підготовку сирого аудіосигналу. Зокрема, виконується перетворення формату даних (монофонізація, приведення до потрібної частоти дискретизації), нормалізація гучності, усунення фонових шумів і, за необхідності, обчислення ознак для моделі (наприклад, перетворення звуку в мел-спектрограму). Цей модуль гарантує, що на вхід моделі надійдуть дані в оптимальному вигляді.

- Модуль розпізнавання (ML-модель) – основний компонент, що реалізує автоматичне розпізнавання мовлення. У рамках проєкту використовується модель Whisper від OpenAI [20] – сучасна нейронна мережа-трансформер, яка перетворює підготовлений аудіосигнал на текст. Модуль завантажує попередньо навчену модель та виконує інференс (прогноз) текстового результату.
- Модуль виведення та збереження результатів – приймає текст від модулю розпізнавання та відображає його в GUI для користувача. Додатково він дозволяє зберегти отриманий транскрипт у файл (наприклад, у форматі .txt або субтитрів) чи передати до іншої системи.

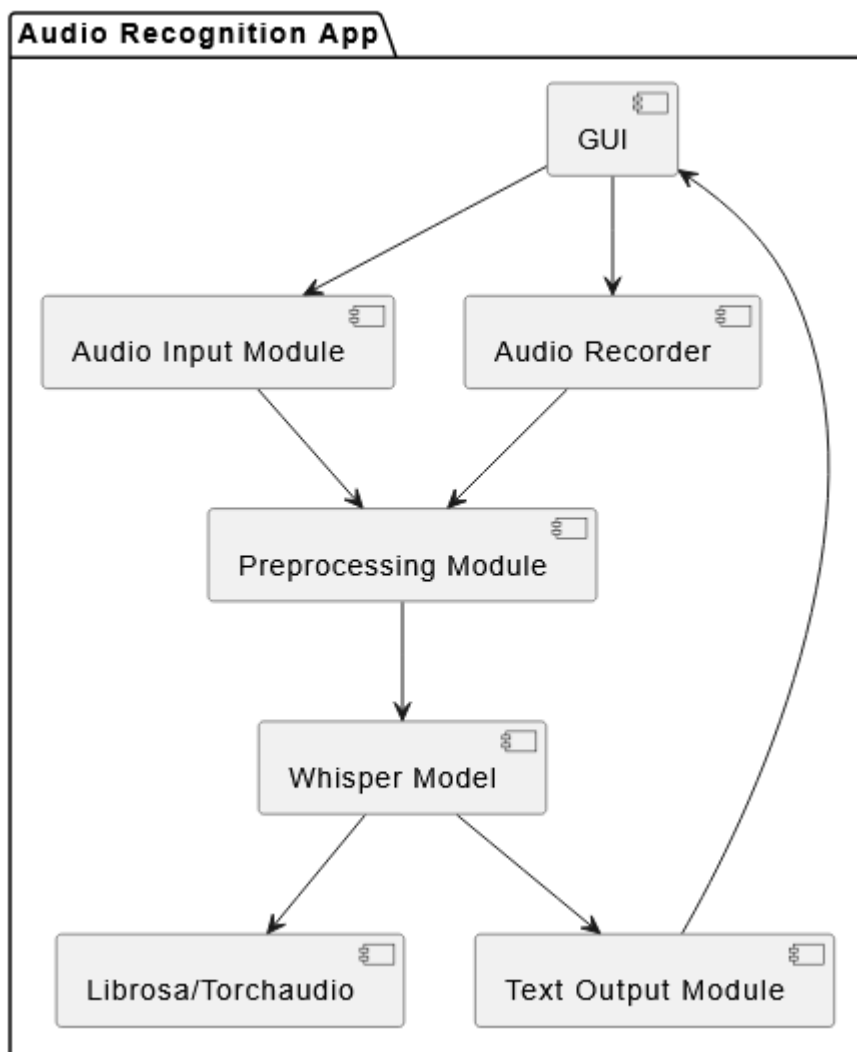


Рисунок 2.1 – Архітектурна модель системи

Джерело: розроблено автором самостійно

На рисунку 2.1 схематично показано взаємодію зазначених компонентів: GUI ініціює введення аудіо, яке проходить через модулі обробки до моделі розпізнавання, а отриманий текст повертається для відображення і збереження. Така концептуальна схема забезпечує модульність і гнучкість системи – кожен компонент можна вдосконалювати або замінювати незалежно (наприклад, підключити іншу модель або додаткові етапи обробки) без порушення роботи інших частин.

Відповідно до наведеної концептуальної моделі, для реалізації кожного з ключових елементів були обрані конкретні технології. Вибір базувався на аналізі альтернатив та результатах попередньої практики, з метою забезпечити найкращу точність, продуктивність і зручність розробки. Нижче обґрунтовано вибір кожної з основних технологій та інструментів.

- Модель розпізнавання – Whisper (OpenAI). В якості ядра системи прийнято рішення використати модель Whisper від компанії OpenAI [20]. Whisper – це універсальна трансформерна модель для розпізнавання мовлення, попередньо навчена на 680 тисяч годин багатомовних аудіоданих. Завдяки цьому вона підтримує понад 50 мов і демонструє високі показники точності розпізнавання (для англійської мови досягає $\sim 2,7\%$ помилки слів (WER) на тестових наборах). Whisper працює за принципом end-to-end і не потребує ручного налаштування акустичних моделей чи мовних моделей – достатньо подати «сирий» аудіосигнал, і модель генерує текст (використовуючи внутрішнє представлення у вигляді мел-спектрограми). Альтернативні рішення, які розглядалися, включали open-source платформи Kaldi, DeepSpeech та модель Wav2Vec 2.0. Від Kaldi [21] відмовилися через високу складність інтеграції та застарілу архітектуру: Kaldi являє собою набір інструментів для побудови власної ASR-системи, що вимагає глибокої експертизи та створення складних конвеєрів (особливо на базі моделей GMM/HMM). DeepSpeech від Mozilla [22] (реалізація архітектури RNN згідно з дослідженням А. Hannun et al.) також розглядався, але цей

проект наразі фактично не підтримується (Mozilla припинила активну розробку у 2021 р.) і модель орієнтована переважно на англійську мову, поступаючись сучасним трансформерам у точності. Модель Wav2Vec 2.0 від Facebook AI [23] показує чудові результати у дослідницьких умовах (самонавчання на невеличких розмічених вибірках, висока точність), однак для нашого завдання вимагала б додаткового донавчання під конкретні мови або домени. На відміну від Whisper, яка одразу підтримує багатомовне розпізнавання «з коробки», використання Wav2Vec 2.0 потребує тонкого налаштування моделі на кожну цільову мову чи набір даних. Таким чином, Whisper була обрана як оптимальне рішення, що забезпечує високу якість розпізнавання мовлення без необхідності збору великого корпусу даних чи тривалого навчання моделі з нуля. Варто зазначити, що вибір Whisper підтверджено й результатами переддипломної практики: у ході експериментів модель Whisper (зокрема, варіант base) продемонструвала кращу точність розпізнавання на тестових аудіоданих у порівнянні з альтернативами (DeepSpeech, Wav2Vec 2.0 тощо), при помірних вимогах до обчислювальних ресурсів.

- Бібліотека обробки аудіо – Librosa. Для етапу передобробки аудіосигналу обрано бібліотеку librosa [24] – одну з найбільш популярних Python-бібліотек для аналізу звукових сигналів. Librosa надає багатий набір готових функцій для роботи з аудіо: зручне завантаження файлів різних форматів, ресемплінг (зміна частоти дискретизації), фільтрація, виділення корисного сигналу (обтинання тиші), обчислення характеристик на кшталт MFCC або мел-спектрограми тощо. Використання цієї бібліотеки значно спрощує реалізацію передобробки, оскільки більшість стандартних алгоритмів уже реалізовано та оптимізовано в її функціях. Наприклад, щоб отримати мел-спектрограму аудіофрагменту, достатньо викликати відповідну функцію librosa, замість того щоб писати власну реалізацію

перетворення Фур'є та шкали Мела. Альтернативи: Можна було б застосувати нижчорівневі засоби (наприклад, модуль `wave` зі стандартної бібліотеки Python або `SciPy`) для читання і обробки аудіо, проте це потребувало б більше коду і часу на реалізацію базових функцій. Інша альтернатива – використати можливості `torchaudio` (який теж має засоби завантаження/перетворення аудіо) або суміжні бібліотеки (наприклад, `soundfile` для читання WAV/FLAC). Однак, виходячи з попереднього досвіду, `librosa` забезпечує більш високий рівень абстракції і зручності: багато типових операцій виконуються викликом однієї функції, що прискорює розробку та зменшує ймовірність помилок. До того ж, `librosa` широко використовується в наукових дослідженнях звуку і музики [24], що підтверджує її надійність та якість реалізації алгоритмів. У проєкті `librosa` використовується для підготовки аудіоданих перед подачею в модель `Whisper` – з її допомогою аудіо приводиться до потрібного формату і, за необхідності, генерується вхідна мел-спектрограма відповідно до вимог моделі.

- Інтеграція з PyTorch – бібліотека `Torchaudio`. Модель `Whisper` від OpenAI реалізована на платформі PyTorch, тому для узгодженої роботи з тензорами був додатково застосований пакет `torchaudio`. `Torchaudio` – це офіційна аудіо-бібліотека для PyTorch, яка надає засоби для завантаження аудіо та обробки сигналів безпосередньо у вигляді тензорів PyTorch. Використання `torchaudio` спрощує передачу даних до нейронної мережі: наприклад, завантажений аудіосигнал можна одразу перетворити на тензор потрібної форми і завантажити на GPU для прискорення обчислень. В рамках проєкту `torchaudio` доповнює можливості `librosa` там, де потрібна тісна інтеграція з PyTorch – зокрема, для перетворення форматів даних та виконання деяких операцій обробки на льоту під час інференсу моделі. Альтернативи: Суто технічно, можна було обійтися і без `torchaudio`, здійснюючи

обробку лише засобами librosa з подальшим конвертуванням numpy-масивів у тензори. Але такий підхід менш ефективний і потребує більше ручного коду. Torchaudio натомість забезпечує готові функції для, скажімо, ресемплінгу чи аудіо-аугментацій прямо у середовищі PyTorch. Будучи частиною PyTorch-екосистеми, ця бібліотека гарантує сумісність форматів даних і може використовувати апаратне прискорення. Тож вибір torchaudio як додаткового інструменту обґрунтований прагненням до оптимальної продуктивності та простоти реалізації зв'язки «передобробка – модель» на PyTorch.

- Фреймворк GUI – PySide6 (Qt for Python). В якості технології для побудови графічного інтерфейсу було обрано PySide6, що є офіційним Python-біндингом для бібліотеки Qt 6. PySide6 надає розробнику потужний інструментарій для створення сучасних десктопних інтерфейсів: підтримує всі елементи UI (кнопки, поля, меню, діалоги тощо), дозволяє проєктувати інтерфейс у візуальному редакторі Qt Designer та легко інтегрується з Python-логікою програми. Головною перевагою PySide6 перед альтернативами є ліцензування та підтримка. PyQt 6 – близький за функціональністю конкурент – має ліцензію GPL, що означає певні обмеження для закритих проєктів (вимогу відкривати вихідний код або придбати комерційну ліцензію). PySide6 натомість розповсюджується під ліцензією LGPL, що дозволяє вільно використовувати його у пропрієтарних проєктах без розкриття коду. З огляду на потенційні перспективи комерційного використання застосунку, цей фактор став вирішальним на користь PySide6. Окрім того, PySide6 офіційно підтримується The Qt Company, що гарантує актуальність бібліотеки та сумісність з новими версіями Qt. Альтернативи: Розглядався також варіант використання стандартної бібліотеки Tkinter (яка постачається разом з Python). Однак, можливості Tkinter досить обмежені – створений інтерфейс виглядав би застарілим і не забезпечив би належного рівня зручності (наприклад,

відсутня підтримка сучасних макетів, стилів, складних віджетів). Інші фреймворки, такі як Kivy або веб-орієнтовані рішення (створення веб-застосунку для розпізнавання з фронтом на JavaScript), були відхилені через надмірну складність та невиправдані витрати ресурсів на їх впровадження. В результаті, PySide6 було обрано як оптимальну основу для GUI, що поєднує багатий функціонал Qt з гнучкістю Python. Це дозволило швидко розробити зручний крос-платформний інтерфейс: у застосунку передбачені елементи для вибору файлу, керування записом аудіо, відображення статусу розпізнавання та виводу тексту. Варто додати, що використання десктопного GUI (замість веб-інтерфейсу) спростило доступ до апаратних ресурсів (мікрофона, файлової системи) та розгортання програми на робочій станції користувача.

Отже, концептуальна модель системи визначила розподіл функцій між окремими компонентами, а аналіз можливих інструментів показав переваги обраних технологій. Комбінація моделі Whisper з бібліотеками librosa/torchaudio і фреймворком PySide6 забезпечує реалізацію поставлених вимог до застосунку: точне розпізнавання багатомовного аудіо, ефективну обробку сигналу та зручний інтерфейс користувача. Кожна з обраних технологій перевірена на практиці та має підтверджену ефективність, що мінімізує ризики при розробленні дипломного проєкту.

2.3 Проєктування архітектури системи

Архітектура системи побудована за модульним принципом із чітким розмежуванням функціональних компонентів. Завдяки поділу на окремі підсистеми кожна частина застосунку відповідає за свою сферу завдань, що спрощує розвиток і масштабування рішення. На загальному рівні система

складається з таких основних компонентів: інтерфейсу користувача (GUI), модуля введення аудіо, модуля обробки аудіосигналу, ML-моделі Whisper і модуля виведення результату. Такий підхід дозволяє незалежно розробляти і тестувати кожну підсистему, а також забезпечувати високу стійкість та гнучкість рішення. Загальну архітектуру системи проілюстровано на рисунку 2.1.

Функції окремих підсистем наступні:

- Інтерфейс користувача (GUI) – реалізований з використанням PySide6 і відповідає за взаємодію з користувачем (запуск запису, вибір файлу, відображення тексту). GUI працює в основному потоці програми, приймає команди користувача та відображає результати розпізнавання (рис. 2.5 – сценарій взаємодії).
- Модуль введення аудіо – відповідає за захоплення звукового сигналу з мікрофона або завантаження наявного аудіофайлу. Цей модуль забезпечує попередню перевірку формату і якісну ініціалізацію даних для подальшої обробки.
- Модуль обробки аудіо – виконує попередню обробку сигналу (наприклад, нормалізацію рівня гучності, видалення шуму) і готує звукові дані для подачі в ML-модель. У рамках проєктування вказано послідовні кроки обробки звуку, які відображено на рисунку 2.3.
- ML-модель (Whisper) – основний блок розпізнавання мовлення. Отримує оброблений аудіосигнал і генерує текстовий результат. Архітектурно модель розміщується як окремий компонент, до якого звертається модуль обробки аудіо. Це дозволяє за потреби замінити модель або додавати інші моделі розпізнавання.
- Модуль виведення результату – відповідає за передачу результатів розпізнавання з ML-моделі до інтерфейсу користувача. Включає перетворення внутрішніх даних у зручний для відображення формат.

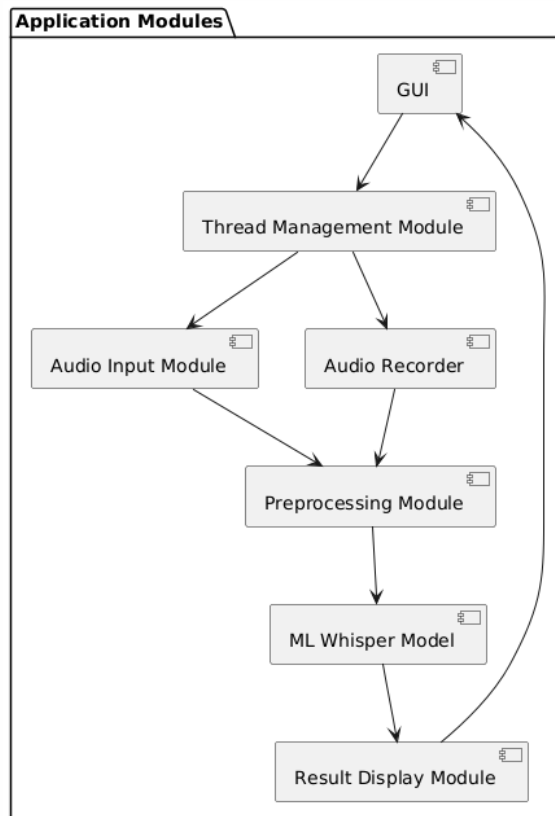


Рисунок 2.2 – Модульна структура застосунку

Джерело: розроблено автором самостійно

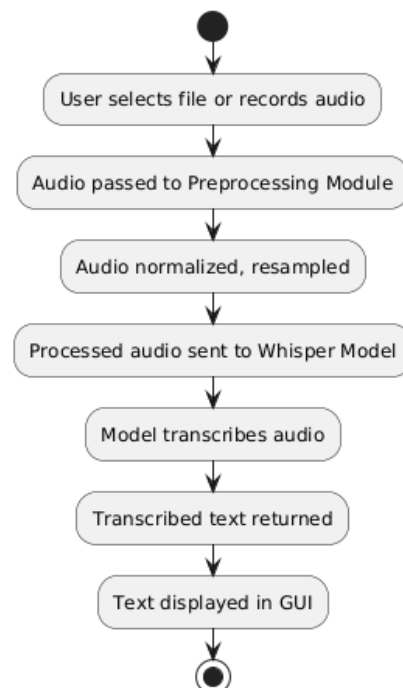


Рисунок 2.3 – Послідовність обробки аудіо

Джерело: розроблено автором самостійно

Компоненти системи взаємодіють за допомогою чітко визначених інтерфейсів і протоколів передачі даних. Наприклад, модуль введення аудіо передає двійкові аудіодані до модуля обробки, який після перетворення передає результат до ML-моделі. Згенерований текст моделю надходить у модуль виведення, звідки передається в GUI для відображення користувачу. Така послідовність обробки аудіо ілюструється на рисунку 2.2.

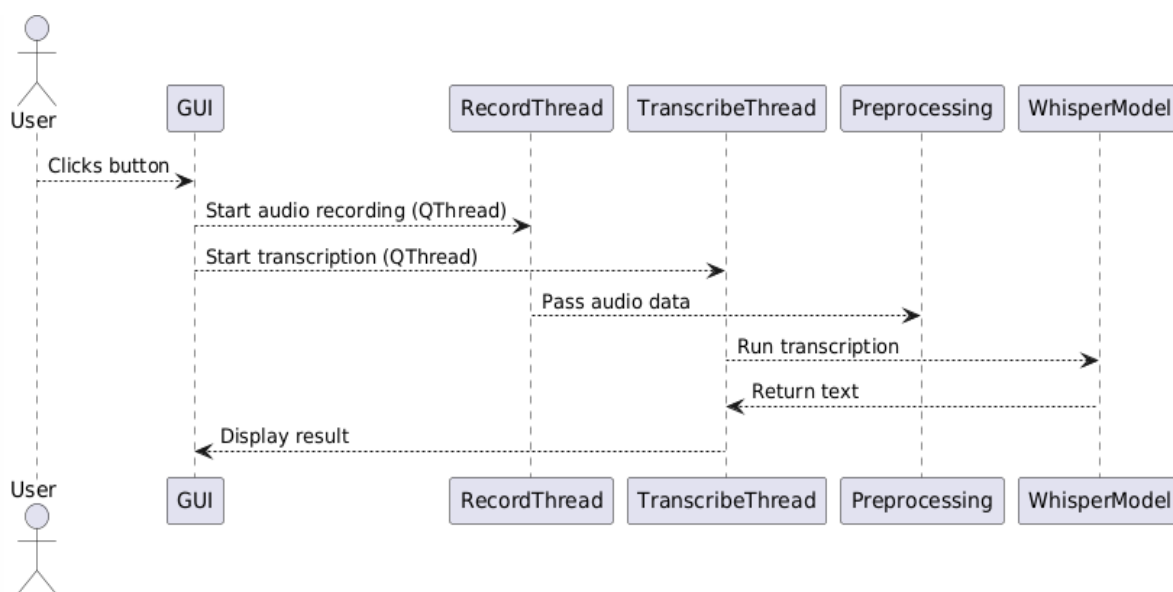


Рисунок 2.4 – Логіка роботи потоків

Джерело: розроблено автором самостійно

Для забезпечення плавної роботи інтерфейсу і уникнення блокувань використовуються окремі потоки виконання (QThread). Основний потік відповідає за GUI і не виконує дорогих розрахунків, водночас запис та обробка аудіо відбуваються асинхронно у фонових потоках. Наприклад, під час запису голосу програма створює QThread для захоплення звуку і передачі даних модулю обробки, після чого запускається окремий QThread для обробки та розпізнавання (рис. 2.4). Така організація потоків гарантує, що взаємодія з користувачем залишається чутливою, а ресурси витрачаються ефективно.

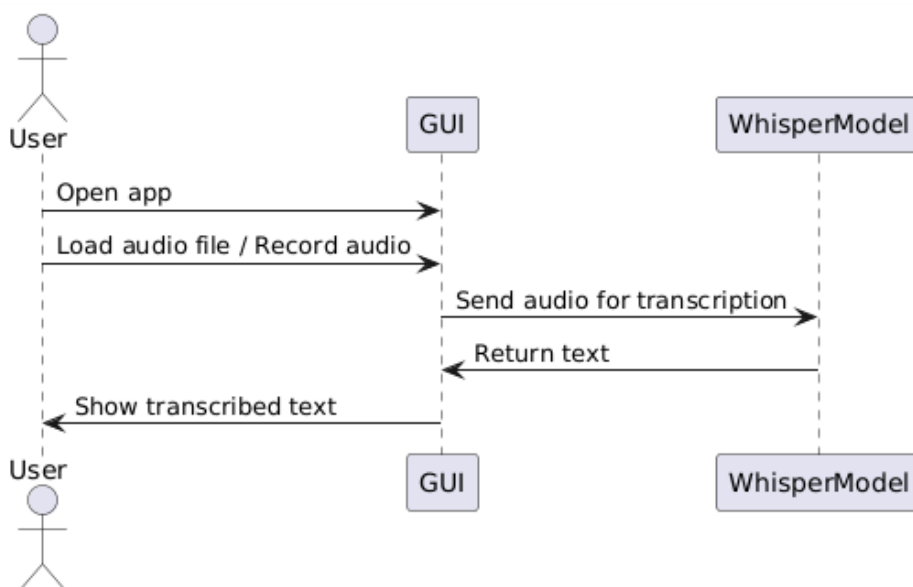


Рисунок 2.5 – Сценарій взаємодії з користувачем (інтерфейс)

Джерело: розроблено автором самостійно

Система спроектована з урахуванням майбутньої масштабованості. Архітектура дозволяє інтегрувати нові можливості, такі як голосовий асистент, додаткові моделі розпізнавання чи режим роботи на серверній платформі. Наприклад, модуль ML-моделі можна замінити або доповнити іншими алгоритмами, а GUI доповнити командним інтерпретатором для голосових команд. Проте зазначені розширення виходять за рамки поточної роботи і розглядаються як потенційний напрямок подальшого розвитку.

Висновки до розділу 2

У другому розділі дипломної роботи було проведено комплексне проектування системи автоматичного розпізнавання аудіо з використанням інструментарію машинного навчання. Спершу визначено основні вимоги до майбутньої системи: підтримка україномовних даних, висока точність

розпізнавання, робота на локальних ресурсах (без постійного підключення до Інтернету) та забезпечення користувацької гнучкості.

Далі було сформовано концептуальну модель, яка описує взаємодію основних компонентів системи: моделі розпізнавання (наприклад, Whisper), аудіообробки (librosa, torchaudio), графічного інтерфейсу користувача (PySide6) та модуля управління потоками. Такий вибір технологій дозволив збалансувати вимоги щодо продуктивності, масштабованості та простоти використання.

Архітектура системи була спроектована з акцентом на модульність, що забезпечує можливість розширення функціональності в майбутньому (наприклад, інтеграцію з голосовими помічниками або асистентами). У розділі також було представлено схеми й UML-діаграми, що деталізують взаємодію між модулями, підсистемами й потоками даних, закладаючи підґрунтя для реалізації програмного забезпечення.

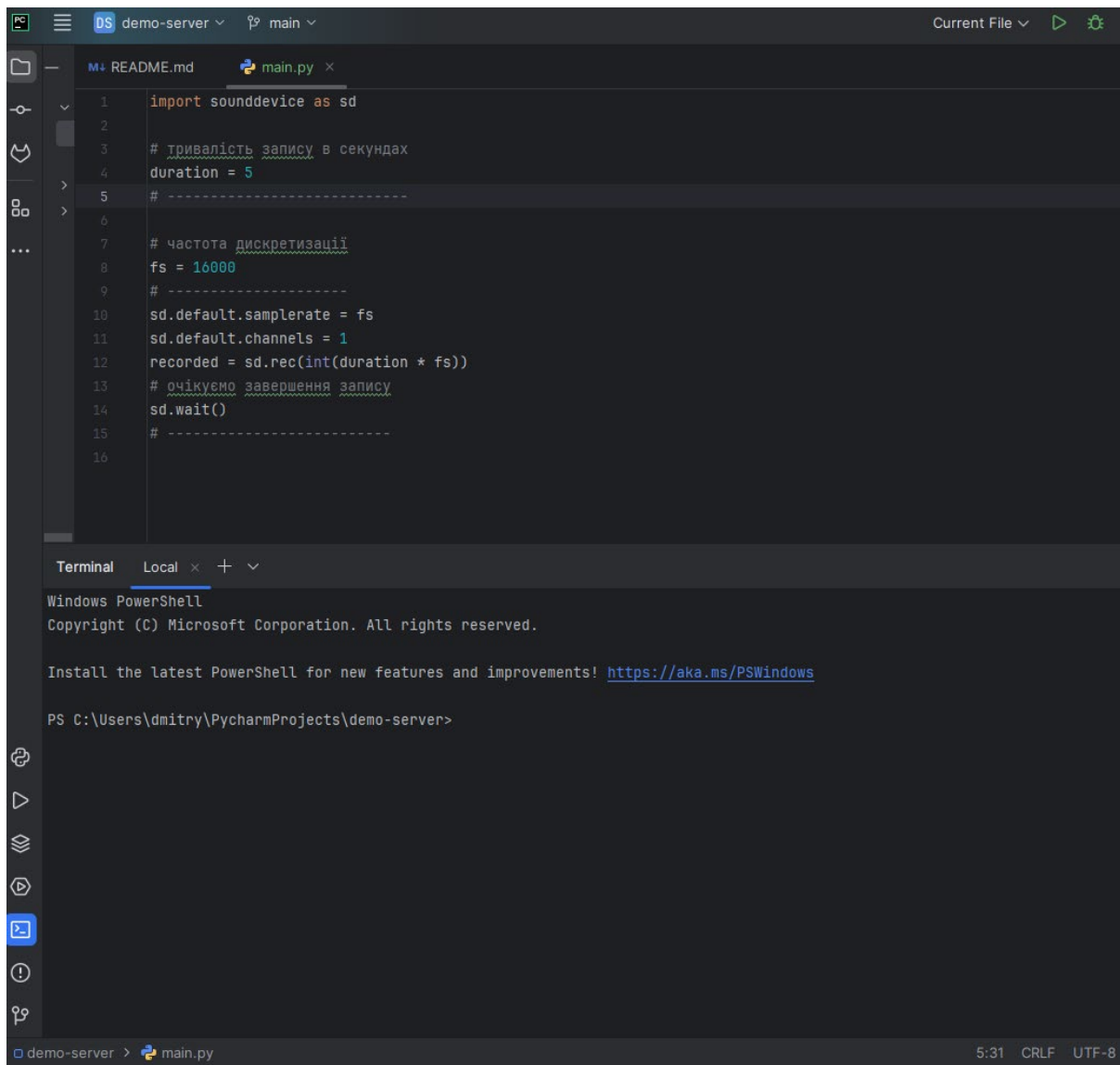
Загалом, результати другого розділу демонструють цілісне розуміння структури, завдань і технічних рішень, необхідних для побудови якісної системи розпізнавання аудіо, та слугують підготовкою до практичного етапу розробки в наступному розділі.

РОЗДІЛ 3 РОЗРОБЛЕННЯ ПРОЄКТНИХ РІШЕНЬ

3.1 Проєктування програмного забезпечення системи

Система складається з двох основних компонентів: Додаток А – основного GUI-застосунку (`main.py`) та Додаток Б – допоміжного скрипту (`script-load.py`). Основний додаток реалізує графічний інтерфейс користувача, забезпечує запис аудіо з мікрофона, передачу звукового сигналу до моделі та відображення результату. Він побудований на фреймворку PySide6 і для фонові обробки даних використовує механізм багатопоточності на основі `QThread`. Допоміжний скрипт призначений для попередньої перевірки працездатності моделі Whisper за межами GUI. При запуску `script-load.py` модель завантажується аналогічно (`model = whisper.load_model("base")`) та застосовується до готового аудіофайлу з подальшим виводом транскрипції у консоль. Обидва додатки працюють з однією і тією ж попередньо навченим моделлю Whisper, що забезпечує можливість їх паралельного використання: GUI-інтерфейс — для кінцевого користувача, а скрипт — для розробника при тестуванні алгоритму на різних прикладах.

У основному додатку при запуску ініціалізується модель Whisper (наприклад, `model = whisper.load_model("base")`), після чого програма чекає на команду користувача на запис. Для запису звуку з мікрофона використовується бібліотека `sounddevice`, що дозволяє записувати аудіо у масив NumPy. Приклад на рисунку 3.1.



The screenshot shows the PyCharm IDE interface. The top toolbar includes icons for file operations, search, and running. The main editor displays a Python file named `main.py` with the following code:

```
1 import sounddevice as sd
2
3 # тривалість запису в секундах
4 duration = 5
5 # -----
6
7 # частота дискретизації
8 fs = 16000
9 # -----
10 sd.default.samplerate = fs
11 sd.default.channels = 1
12 recorded = sd.rec(int(duration * fs))
13 # очікуємо завершення запису
14 sd.wait()
15 # -----
16
```

Below the editor is a terminal window titled "Terminal" with a "Local" tab. It shows the Windows PowerShell prompt and the current directory:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\dmityr\PycharmProjects\demo-server>
```

The status bar at the bottom indicates the current file is `main.py` and the encoding is UTF-8.

Рисунок 3.1 – Приклад перетворення аудіо у масив NumPy

Джерело: розроблено автором самостійно

Після запису сигнал перетворюється у формат, прийнятний для моделі, і передається на розпізнавання. При необхідності застосовується попереднє обчислення спектрограм або ресемплінг через librosa, рисунок 3.2.

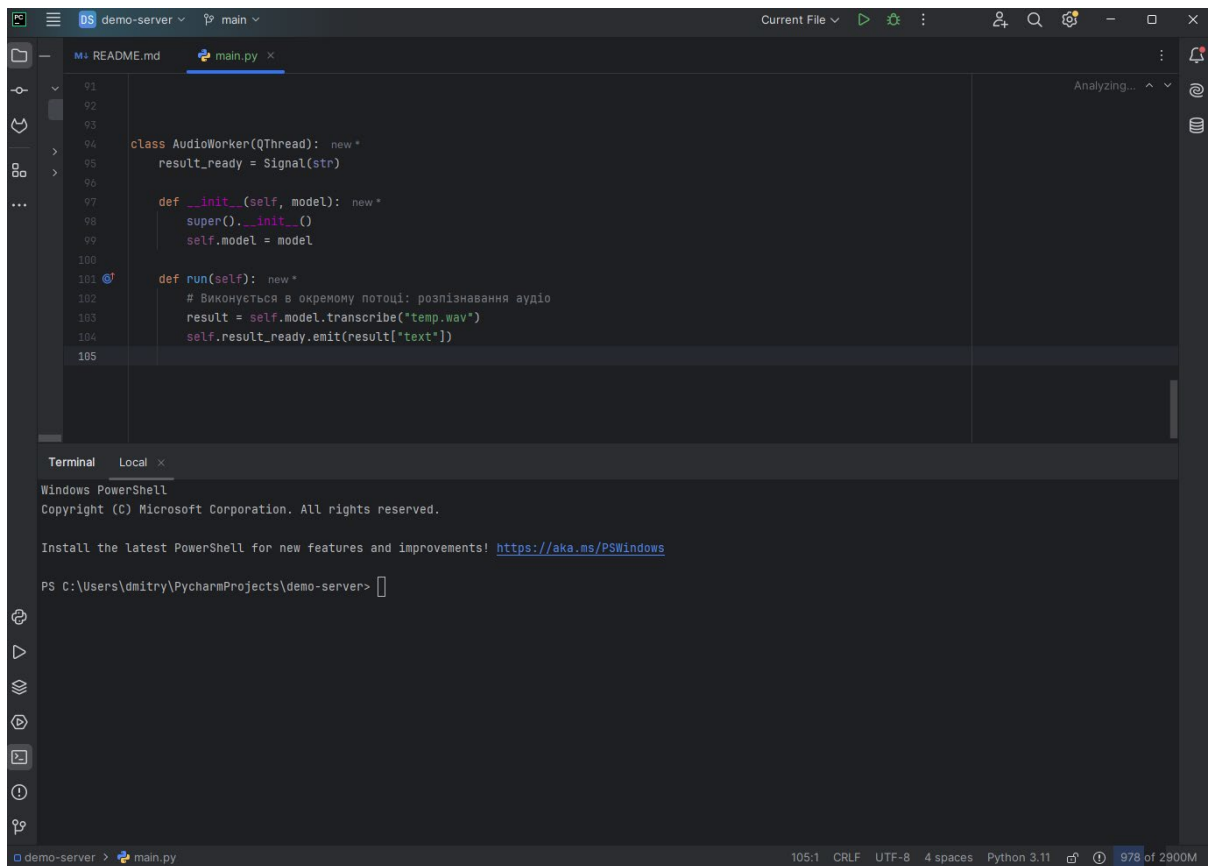


Рисунок 3.2 – Фрагмент коду для використання QThread у GUI

Джерело: розроблено автором самостійно

У наведеному прикладі AudioWorker наслідує QThread і містить сигнал result_ready для передачі результату розпізнавання. Метод run() виконується у фоновому потоці, де здійснюється безпосередньо виклик model.transcribe(). Після завершення транскрипції через сигнал передається розпізнаний текст у головний потік інтерфейсу. Такий підхід дозволяє не блокувати головний потік GUI під час тривалих обчислень (розпізнавання мовлення), що гарантує збереження відгуку інтерфейсу.

Допоміжний скрипт використовується для тестування моделі Whisper поза межами GUI. При запуску скрипт завантажує ту саму модель (наприклад, model = whisper.load_model("base")) і виконує розпізнавання на фіксованому файлі. Типовий фрагмент коду скрипту може виглядати так, як на рисунку 3.3.

```

1 import whisper
2
3 model = whisper.load_model("base") # завантаження моделі whisper
4 result = model.transcribe("sample.wav") # розпізнавання аудіофайлу
5 print(result["text"]) # виведення розпізнаного тексту
6

```

Terminal root@pc: /data/demo-server x root@pc: /data/demo-server (2) x + v

0[0.0%] 3[0.0%] 6[0.0%] 9[0.0%] 10[0.0%] 13[0.0%] 16[0.0%] 19[0.0%]
1[0.0%] 4[0.0%] 7[0.0%] 11[0.0%] 14[0.0%] 17[0.0%]
2[0.0%] 5[0.0%] 8[0.0%] 12[0.0%] 15[0.0%] 18[0.0%]

Mem[|||||] 5.746/15.5G
Swp[|||||] 0K/4.00G

Tasks: 112, 378 thr; 1 running
Load average: 0.35 0.22 0.11
Uptime: 6 days, 16:25:11

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPUS	MEM%	TIME+	Command
2395809	root	20	0	6104	4544	3212	R	333	0.0	0:00.10	htop
1	root	20	0	163M	12796	8136	S	0.0	0.1	0:24.46	/sbin/init
2	root	20	0	2776	1924	1796	S	0.0	0.0	0:08.94	/init
7	root	20	0	2812	68	68	S	0.0	0.0	0:01.02	plan9 --control-socket 7 --log-level 4 --server-fd 8 --pipe-fd 10 --log-truncate
8	root	20	0	2812	68	68	S	0.0	0.0	0:00.21	plan9 --control-socket 7 --log-level 4 --server-fd 8 --pipe-fd 10 --log-truncate
9	root	20	0	2776	1924	1796	S	0.0	0.0	0:01.31	/init
66	root	19	-1	76452	47304	46100	S	0.0	0.3	0:24.50	/lib/systemd/systemd-journald
89	root	20	0	22536	6084	4316	S	0.0	0.0	2:30.16	/lib/systemd/systemd-udev
98	root	20	0	222M	2224	32	S	0.0	0.0	0:00.02	snappyfuse /var/lib/snappy/snaps/bare_5.snap /snap/bare/5 -o ro,nodev,allow_other,suid
100	root	20	0	222M	2224	32	S	0.0	0.0	0:00.00	snappyfuse /var/lib/snappy/snaps/bare_5.snap /snap/bare/5 -o ro,nodev,allow_other,suid
101	root	20	0	222M	2224	32	S	0.0	0.0	0:00.01	snappyfuse /var/lib/snappy/snaps/bare_5.snap /snap/bare/5 -o ro,nodev,allow_other,suid
109	root	20	0	149M	220	56	S	0.0	0.0	0:00.07	snappyfuse /var/lib/snappy/snaps/gtk-common-themes_1535.snap /snap/gtk-common-themes/1535 -o ro,nod
110	root	20	0	149M	220	56	S	0.0	0.0	0:00.05	snappyfuse /var/lib/snappy/snaps/gtk-common-themes_1535.snap /snap/gtk-common-themes/1535 -o ro,nod

F1:help F2:Setup F3:Search F4:Filter F5:Free F6:SortBy F7:Filter F8:Filter F9:Kill F10:Quit

demo-server > roles > demo_restore > tasks > main.yml (D) 1834 B/s / 3.95 kB/s 3:47 LF UTF-8 2 spaces No JSON schema Python 3.10 281 of 900M

Рисунок 3.3 – Фрагмент коду завантаження

Джерело: розроблено автором самостійно

У цьому прикладі модель отримує на вхід файл «sample.wav» і повертає словник result з розпізнаним текстом у полі result["text"]. Завдяки такому скрипту розробник може швидко перевірити роботу моделі на різних аудіо без запуску GUI. Цей інструмент особливо корисний на початкових етапах розробки для перевірки коректності відтворення моделі на відомих прикладах.

Обидва додатки використовують єдину модель Whisper, тож обробка аудіосигналу відбувається схожим чином. Зокрема, для аудіофайлів використовується бібліотека librosa. Функція librosa.load(path, sr=...) дозволяє завантажити файл і за потреби автоматично пересемпліувати його до заданої частоти (наприклад, 16000 Гц). Приклад на рисунку 3.4.

```

1 import librosa
2
3 y, sr = librosa.load("audio.mp3", sr=16000) # завантажити та ресемплувати audio
4

```

```

0[| 2.6%] 3[| 2.0%] 6[| 1.3%] 9[| 2.0%] 10[| 2.0%] 13[| 0.0%] 16[| 0.7%] 19[| 0.0%]
1[| 0.0%] 4[| 0.7%] 7[| 0.7%] 11[| 0.0%] 14[| 1.3%] 17[| 0.7%]
2[| 1.3%] 5[| 2.7%] 8[| 0.0%] 12[| 0.0%] 15[| 1.3%] 18[| 1.3%]
Mem[|||||] 5.73G/15.5G Tasks: 112, 378 thr: 1 running
Swap[|||||] 0K/4.00G Load average: 0.06 0.11 0.09
Uptime: 6 days, 16:30:07

```

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1198639	root	20	0	7444M	4074M	751M	S	12.0	25.6	2h14:11	/root/.cache/JetBrains/RemoteDev/dist/8cfe3c6ce165d_pycharm-professional-2024.2.1/bin/remote-dev
2385021	root	20	0	7444M	4074M	751M	S	3.3	25.6	0:04.51	/root/.cache/JetBrains/RemoteDev/dist/8cfe3c6ce165d_pycharm-professional-2024.2.1/bin/remote-dev
1509	root	20	0	32.16	276K	54424	S	2.7	1.7	4h51:40	/root/.vscode-server/bin/17baf841131aa23349f217ca7c570c70ee87b957/node --dns-result-order=ipv4fi
1198653	root	20	0	7444M	4074M	751M	S	2.7	25.6	2:10.35	/root/.cache/JetBrains/RemoteDev/dist/8cfe3c6ce165d_pycharm-professional-2024.2.1/bin/remote-dev
1198680	root	20	0	7444M	4074M	751M	S	1.3	25.6	19:42.18	/root/.cache/JetBrains/RemoteDev/dist/8cfe3c6ce165d_pycharm-professional-2024.2.1/bin/remote-dev
1143152	mysql	20	0	2384M	413M	36840	S	0.7	2.6	6:36.93	/usr/sbin/mysqld
1232109	root	20	0	7444M	4074M	751M	S	0.7	25.6	2:01.20	/root/.cache/JetBrains/RemoteDev/dist/8cfe3c6ce165d_pycharm-professional-2024.2.1/bin/remote-dev
1258364	root	20	0	7444M	4074M	751M	S	0.7	25.6	3:18.01	/root/.cache/JetBrains/RemoteDev/dist/8cfe3c6ce165d_pycharm-professional-2024.2.1/bin/remote-dev
1258372	root	20	0	7444M	4074M	751M	S	0.7	25.6	2:45.11	/root/.cache/JetBrains/RemoteDev/dist/8cfe3c6ce165d_pycharm-professional-2024.2.1/bin/remote-dev
1258600	root	20	0	7444M	4074M	751M	S	0.7	25.6	2:56.97	/root/.cache/JetBrains/RemoteDev/dist/8cfe3c6ce165d_pycharm-professional-2024.2.1/bin/remote-dev
1277918	root	20	0	7444M	4074M	751M	S	0.7	25.6	2:54.76	/root/.cache/JetBrains/RemoteDev/dist/8cfe3c6ce165d_pycharm-professional-2024.2.1/bin/remote-dev
1494404	root	20	0	7444M	4074M	751M	S	0.7	25.6	1:39.36	/root/.cache/JetBrains/RemoteDev/dist/8cfe3c6ce165d_pycharm-professional-2024.2.1/bin/remote-dev
2397710	root	20	0	7444M	4074M	751M	S	0.7	25.6	0:00.02	/root/.cache/JetBrains/RemoteDev/dist/8cfe3c6ce165d_pycharm-professional-2024.2.1/bin/remote-dev

Рисунок 3.4 – Фрагмент коду зміни частоти

Джерело: розроблено автором самостійно

Цей виклик прочитає будь-який підтримуваний аудіо формат і поверне сигнал у з частотою $sr=16000$. Слід зауважити, що `librosa.load` підтримує будь-які кодеки, які підтримуються бібліотекою `soundfile` або `audioread`, тобто можна використовувати WAV, MP3, FLAC та інші формати. Після отримання даних у формі масиву NumPy вони передаються моделі Whisper (яка всередині сама формує спектрограму) для подальшого розпізнавання мовлення. Таким чином, аудіо з будь-якого з підтримуваних форматів коректно обробляється і приводиться до формату, необхідного моделі.

Однією з ключових особливостей GUI-додатка є використання багатопоточності для уникнення зависання інтерфейсу під час тривалих операцій. Згідно з практиками розробки PyQt/PySide, важкі завдання (запис аудіо, розпізнавання тощо) винесені у окремий потік за допомогою `QThread`. Це гарантує,

що основний потік інтерфейсу залишатиметься чуйним до дій користувача. Реалізація багатопоточності зазвичай включає такі кроки:

1. Створення об'єкта `QThread` та відповідного робочого об'єкта (`Worker`), що містить метод для виконання завдання.
2. Переміщення робочого об'єкта у потік викликом `worker.moveToThread(thread)`.
3. Підключення сигналів та слотів: підключаються сигнали потоку (`thread.started`) до методу `worker.run()`, а сигнал завершення робітника (`worker.finished`) до зупинки (`thread.quit()`) та очищення потоку.
4. Запуск потоку методом `thread.start()`, що призводить до виконання коду `worker.run()` у фоновому режимі.

Наприклад, після створення `QThread` і робітника, сигнали налаштовані так, що при старті потоку викликається метод запису й розпізнавання, а при завершенні – через сигнал `finished` потік зупиняється і звільняються ресурси (віддаляються об'єкти). У результаті досягнуто асинхронної роботи: поки `AudioWorker` працює у власному потоці, головний GUI може обробляти події інтерфейсу. Такий підхід відповідає рекомендаціям Qt щодо організації довготривалих завдань у робочих потоках.

Таким чином, архітектура програмного забезпечення поєднує GUI-додаток для інтерактивної роботи користувача та консольний скрипт для розробницького тестування, обидва використовуючи єдину модель `Whisper`. Обробка аудіо реалізована з урахуванням різних форматів і необхідності ресемплінгу (за допомогою `librosa` та `sounddevice`), а багатопотоковість GUI через `QThread` забезпечує плавну роботу інтерфейсу під час розпізнавання мовлення.

3.2 Проектування графічного інтерфейсу користувача

Для побудови графічного інтерфейсу обрано бібліотеку PySide6 (Qt for Python). PySide6 є офіційними зв'язками Python до фреймворку Qt 6, що надає потужний інструментарій для створення сучасних десктопних інтерфейсів із широким набором стандартних віджетів. Використання Qt гарантує крос-платформенність – інтерфейс виглядатиме «рідним» і на Windows, і на Linux, і на macOS. Крім того, PySide6 офіційно підтримується Qt Company (Qt проєкт затвердив PySide як «Qt для Python»), що забезпечує актуальність і сумісність бібліотеки з новими версіями Qt. Важливим чинником є ліцензування: PySide6 розповсюджується за LGPL, тобто його можна використовувати у відкритих і комерційних проєктах без вимоги розкривати власний код. Завдяки вбудованим інструментам розробки – зокрема, графічним редакторам Qt Designer/Creator з WYSIWYG-інтерфейсом – реалізація GUI відбувається швидко і гнучко.

Конкурентними альтернативами розглядалися Tkinter, PyQt та Kivy. Tkinter постачається разом зі стандартним Python, тому не потребує додаткових залежностей. Він відомий простотою і плавною кривою навчання, що робить його зручним для швидкої розробки невеликих утиліт. Втім його можливості обмежені: сучасного оформлення інтерфейсу або розширеного набору віджетів у базовій версії немає, а для складних GUI доведеться суттєво допрацьовувати стиль (наприклад, через ttk). PyQt – функціонально аналогічна Qt-бібліотека від компанії Riverbank – за API майже не відрізняється від PySide6. Однак у безкоштовній версії PyQt ліцензія GNU GPL вимагає оприлюднення вихідного коду, якщо застосунок поширюється без придбання комерційної ліцензії. Натомість PySide6, будучи LGPL-бібліотекою, дозволяє розробляти закриті програми без додаткових виплат та вимог до розкриття коду. Kivy орієнтований насамперед на мобільні платформи та мультитач-інтерфейси, він будується на OpenGL і дозволяє запускати програми на Android/iOS. Хоча Kivy пропонує новаторський підхід (свій мова опису UI, підтримка жестів), він відносно молодий і має меншу спільноту, що може

призводити до нестабільності та обмеженої кількості готових компонентів. Таким чином, з урахуванням потреб настільного застосунку, PySide6 виявився найзбалансованішим вибором: він поєднує потужний функціонал Qt з простотою розробки на Python.

Зв'язок між графічним інтерфейсом та логікою програми реалізовано через стандартний механізм сигналів та слотів Qt. Коли користувач натискає кнопку (наприклад, «Load Audio File» чи «Record Microphone»), відповідний сигнал спрацьовує та викликає пов'язаний слот – метод, який виконує необхідну дію (завантаження файлу чи початок запису). Завдяки цьому міжшаровій архітектурі можна чітко розділити представлення і бізнес-логіку. Оскільки процес розпізнавання мовлення є тривалим, він виконується в окремому потоці (через QThread), що дозволяє не блокувати основний потік GUI і зберігати плавність роботи інтерфейсу. При цьому Qt гарантує потокобезпечне надсилання даних: сигнали, надіслані з фонового потоку, передаються головному UI-потoku через спеціальні «черги» викликів (queued connections), що забезпечує коректне виконання слотів незалежно від потоку виклику. Таким чином, застосування PySide6 і його механізмів сигнал/слот у поєднанні з багатопотоковістю дозволяє ефективно організувати взаємодію між елементами інтерфейсу та процесами обробки даних у додатку.

Головне вікно програми виконує роль панелі керування процесом розпізнавання аудіо. Воно побудоване на основі бібліотеки PySide6 і містить верхній заголовок із назвою («Transcriber Interface»), дві кнопки для вибору джерела звуку (завантаження файлу або запис із мікрофону) та текстове поле для виводу результатів транскрипції. При запуску користувач бачить центрований заголовок, під яким розташовано дві кнопки — «Load Audio File» і «Record Microphone (5s)» — а нижче вікна розміщено область для відображення логів і тексту транскрипції. Таким чином, інтерфейс забезпечує основні функції програми: вибір локального аудіофайлу або запис звуку з мікрофона та показ результату обробки мовлення.

Макет графічного інтерфейсу зображено на рисунку 3.5 (виконано у Figma): користувач бачить одну головну панель із зазначеними елементами керування.

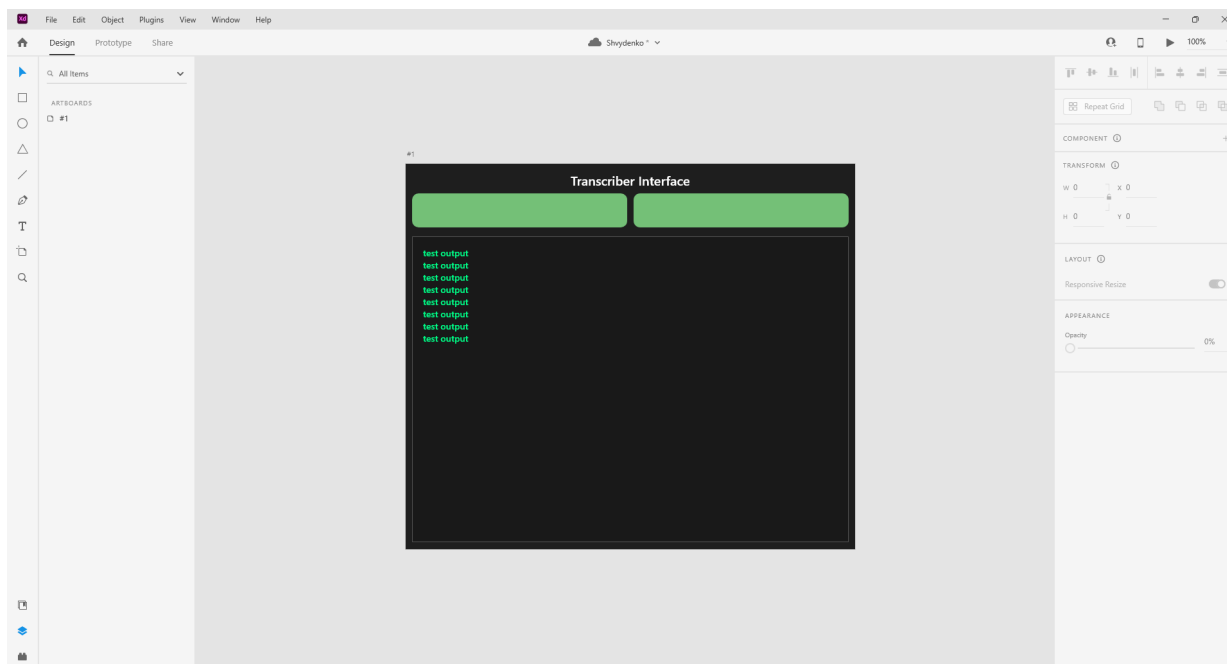


Рисунок 3.5 – Фрагмент коду зміни частоти

Джерело: розроблено автором самостійно в ПЗ Adobe XD[25]

Основні елементи інтерфейсу та їх призначення:

Заголовок вікна (QLabel) — відображає назву програми «Transcriber Interface» (вирівняний по центру)

Кнопка “Load Audio File” (QPushButton) — відкриває діалог вибору аудіофайлу з диску. Після вибору файлу він копіюється у тимчасову директорію, і запускається процес транскрипції.

Кнопка “Record Microphone (5s)” (QPushButton) — починає запис звуку з мікрофона тривалістю 5 секунд. Після завершення запису аудіо автоматично передається на розпізнавання.

Поле виводу результатів (QTextEdit) — велике текстове поле у нижній частині вікна, призначене для відображення інформаційних повідомлень та тексту транскрипції. Це поле зроблено лише для читання, щоб користувач міг скопіювати результат, але не змінювати його вручну.

На рисунку 3.6 наведено приклади фрагментів коду з додатку А, що демонструють створення та ініціалізацію цих елементів інтерфейсу.

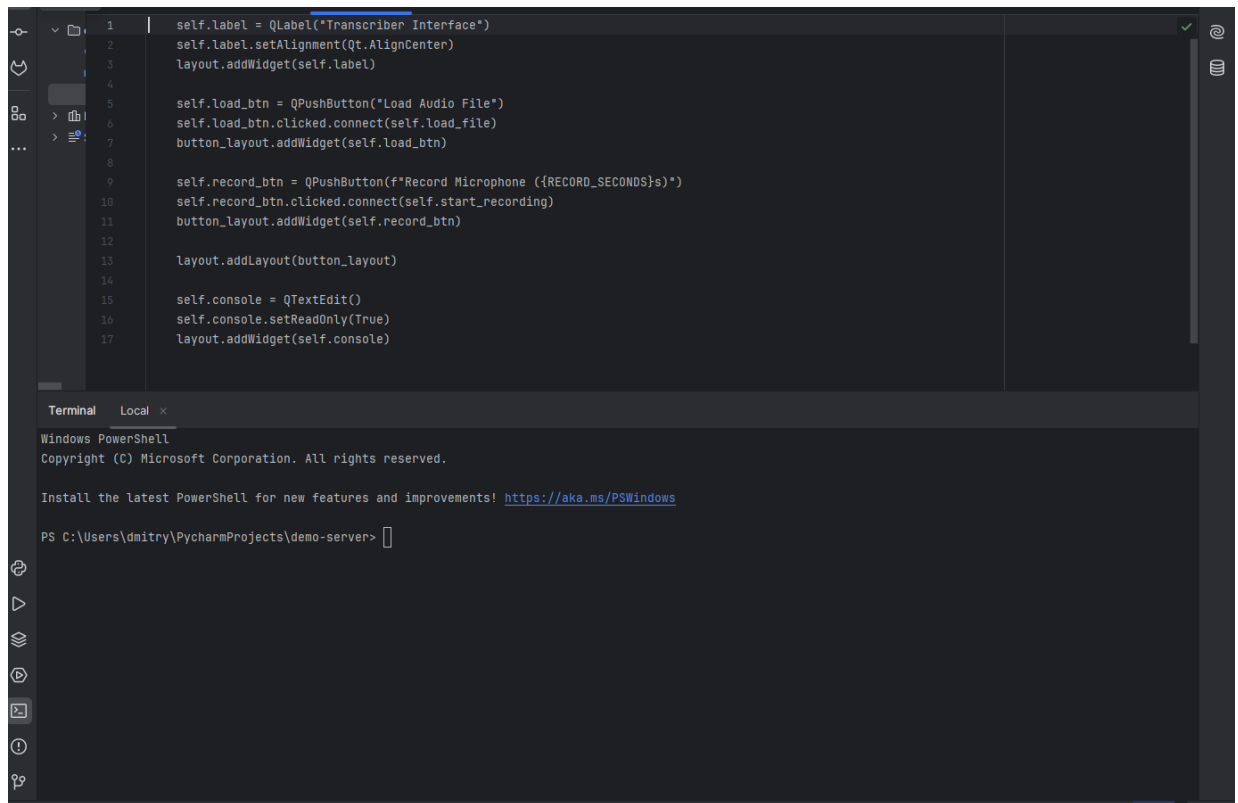
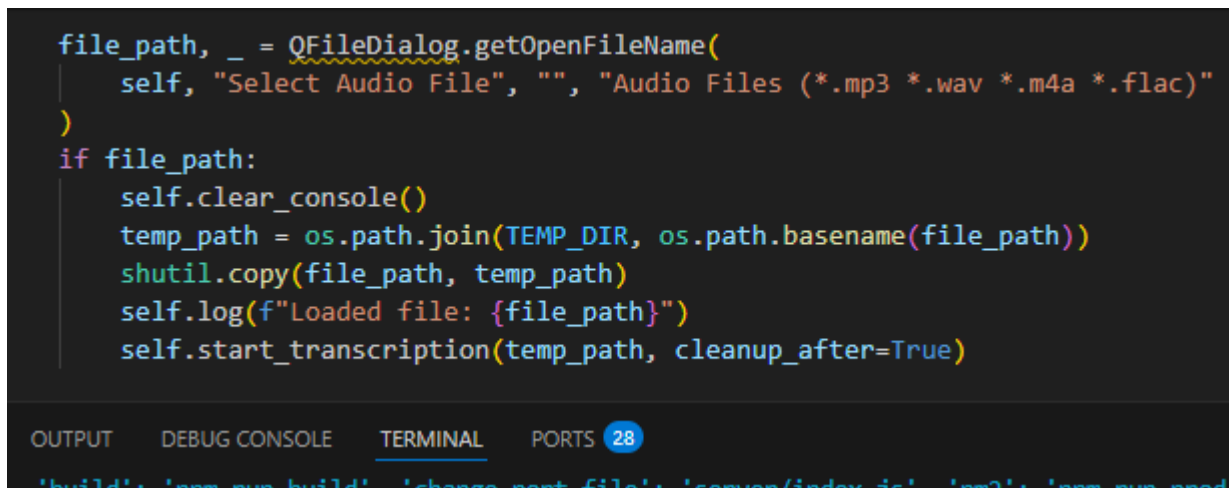


Рисунок 3.6 – Створення та ініціалізація елементів інтерфейсу

Джерело: розроблено автором самостійно

Цей фрагмент коду показує створення елементів GUI: центрований заголовок (QLabel) і дві кнопки (QPushButton) з текстом «Load Audio File» та «Record Microphone (5s)». Для кожної кнопки налаштовано слот — виклик методу `load_file` або `start_recording` при натисканні. Нижче додається горизонтальний бокс з двома кнопками, а в кінець вертикального макету додається багаторазове текстове поле (QTextEdit) для виводу інформації.



```

file_path, _ = QFileDialog.getOpenFileName(
    self, "Select Audio File", "", "Audio Files (*.mp3 *.wav *.m4a *.flac)"
)
if file_path:
    self.clear_console()
    temp_path = os.path.join(TEMP_DIR, os.path.basename(file_path))
    shutil.copy(file_path, temp_path)
    self.log(f"Loaded file: {file_path}")
    self.start_transcription(temp_path, cleanup_after=True)

```

OUTPUT DEBUG CONSOLE TERMINAL PORTS 28

Рисунок 3.7 – Створення та ініціалізація елементів інтерфейсу

Джерело: розроблено автором самостійно

Цей фрагмент описує логіку кнопки “Load Audio File”. Викликається стандартний діалог вибору файлу, після чого вибраний аудіофайл копіюється у тимчасову папку. У вікні консолі виводиться повідомлення про завантаження файлу, і запускається фоновий процес транскрипції звуку. Подібним чином обробляється й натискання кнопки “Record Microphone” (код на рисунку 3.7) - спочатку очищається консоль, користувачу повідомляється про початок запису, блокується інтерфейс до завершення та запускається запис через QThread.

Таким чином, структура інтерфейсу проста і зрозуміла: користувач може вибрати аудіофайл або записати звук, а результати роботи системи виводяться у текстове поле. Кожен елемент розташовано інтуїтивно: зверху – інформаційний заголовок, по центру – панель з кнопками дій, внизу – область для виводу тексту. Ці рішення відповідають рекомендаціям до розробки користувацької документації та методичним вимогам до дипломних робіт. Реалізована версія інтерфейсу буде показана на рисунку 3.8.

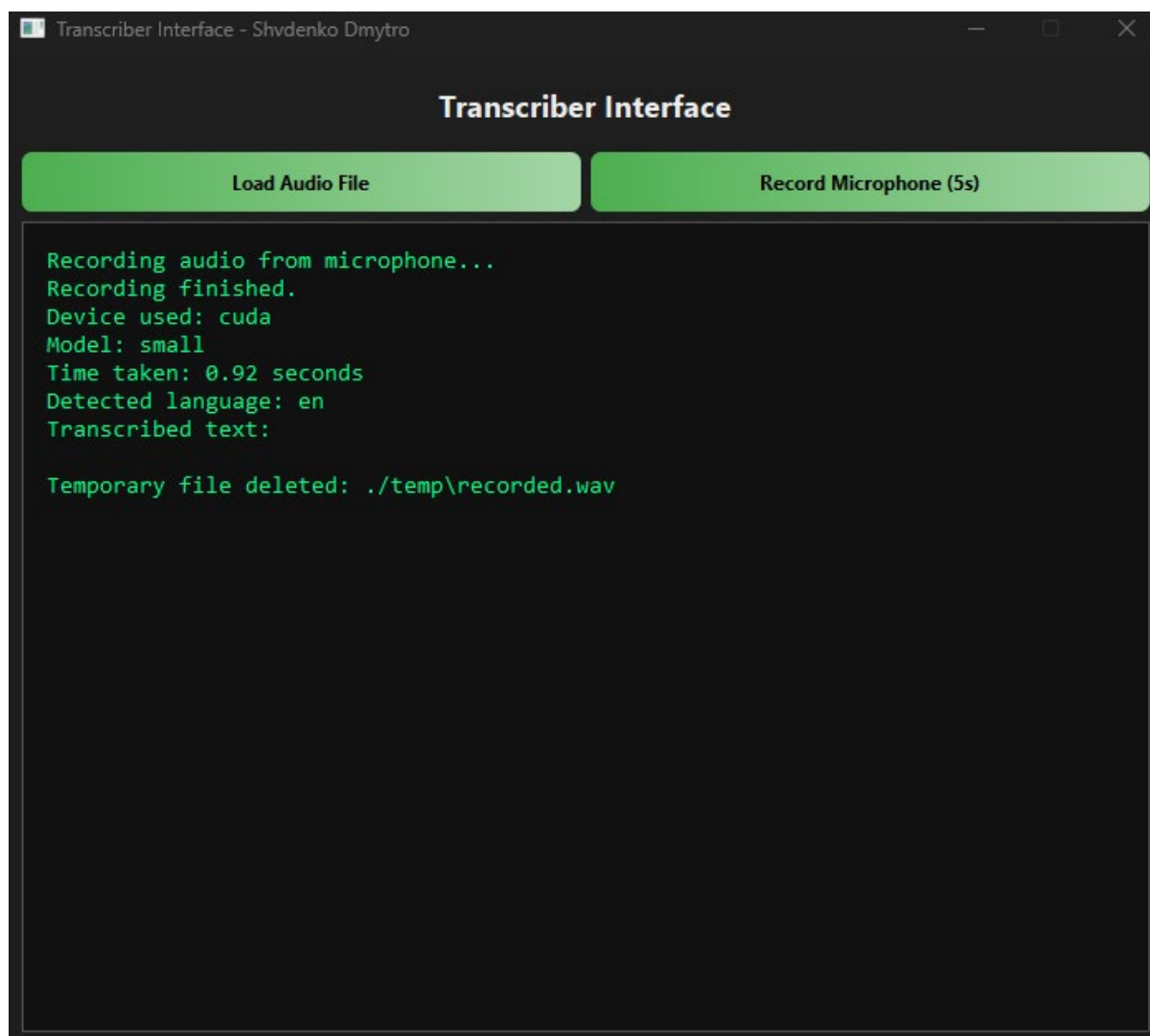


Рисунок 3.8 – запущена програма з готовим інтерфейсом

Джерело: розроблено автором самостійно

3.3 Дослідження продуктивності системи на CPU та GPU

У даному підрозділі представлено аналіз продуктивності розробленої системи для розпізнавання аудіо на основі моделі Whisper (версія small) при роботі на центральному (CPU) та графічному (GPU) процесорах. Оцінювання проводилося на локальному комп'ютері із процесором Intel Core i5-9600K, графічним прискорювачем NVIDIA RTX 4060 Ti та 32 ГБ оперативної пам'яті. Система працює повністю автономно, без залучення інтернет-з'єднання чи хмарних сервісів. Для порівняння продуктивності використано тестовий аудіозапис тривалістю близько 5 секунд. На рисунку 3.9 показано результати з використанням GPU, а на рисунку 3.10 показано результати з використанням CPU.

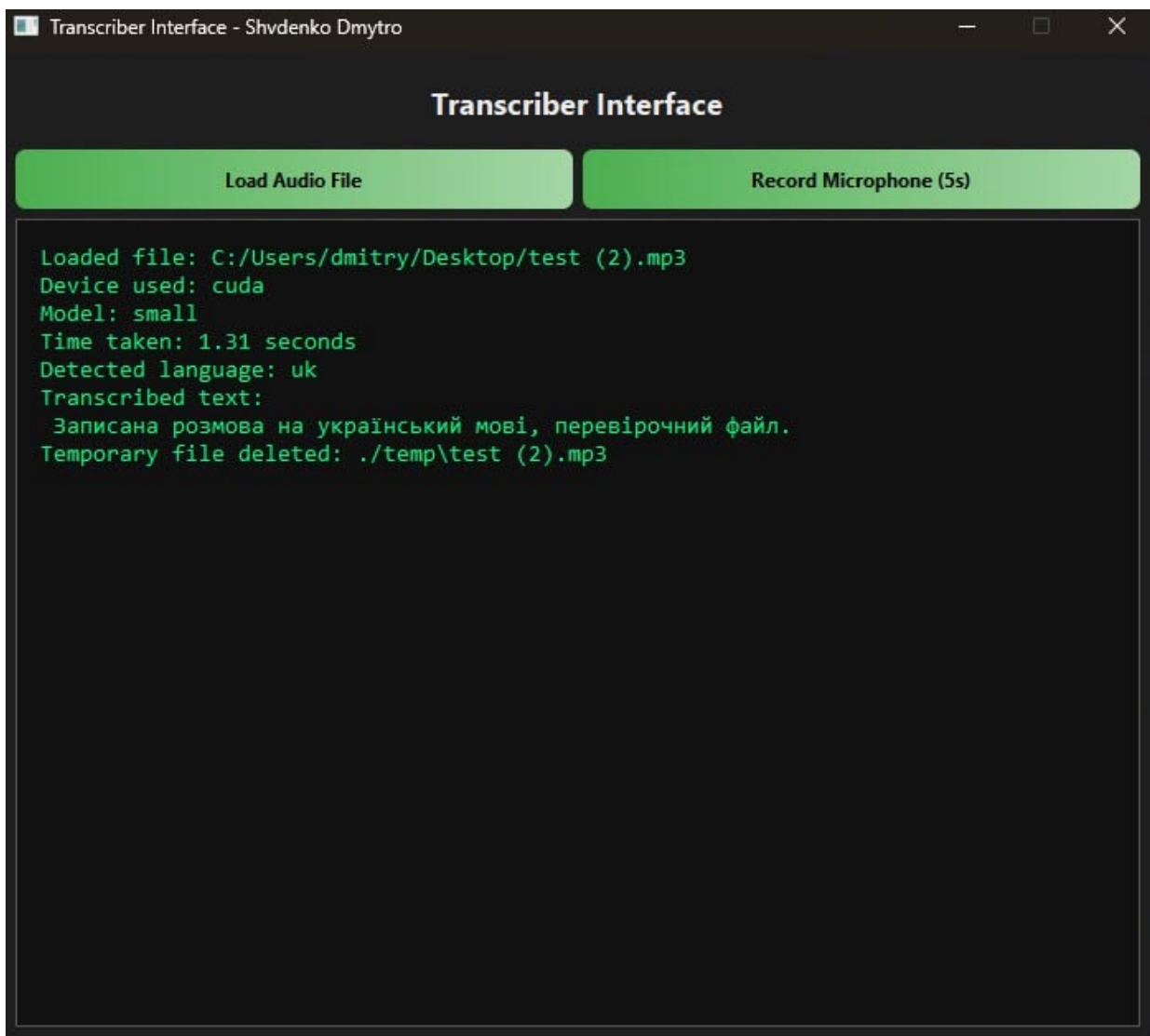


Рисунок 3.9 – Тестування заготовленого запису на GPU

Джерело: розроблено автором самостійно

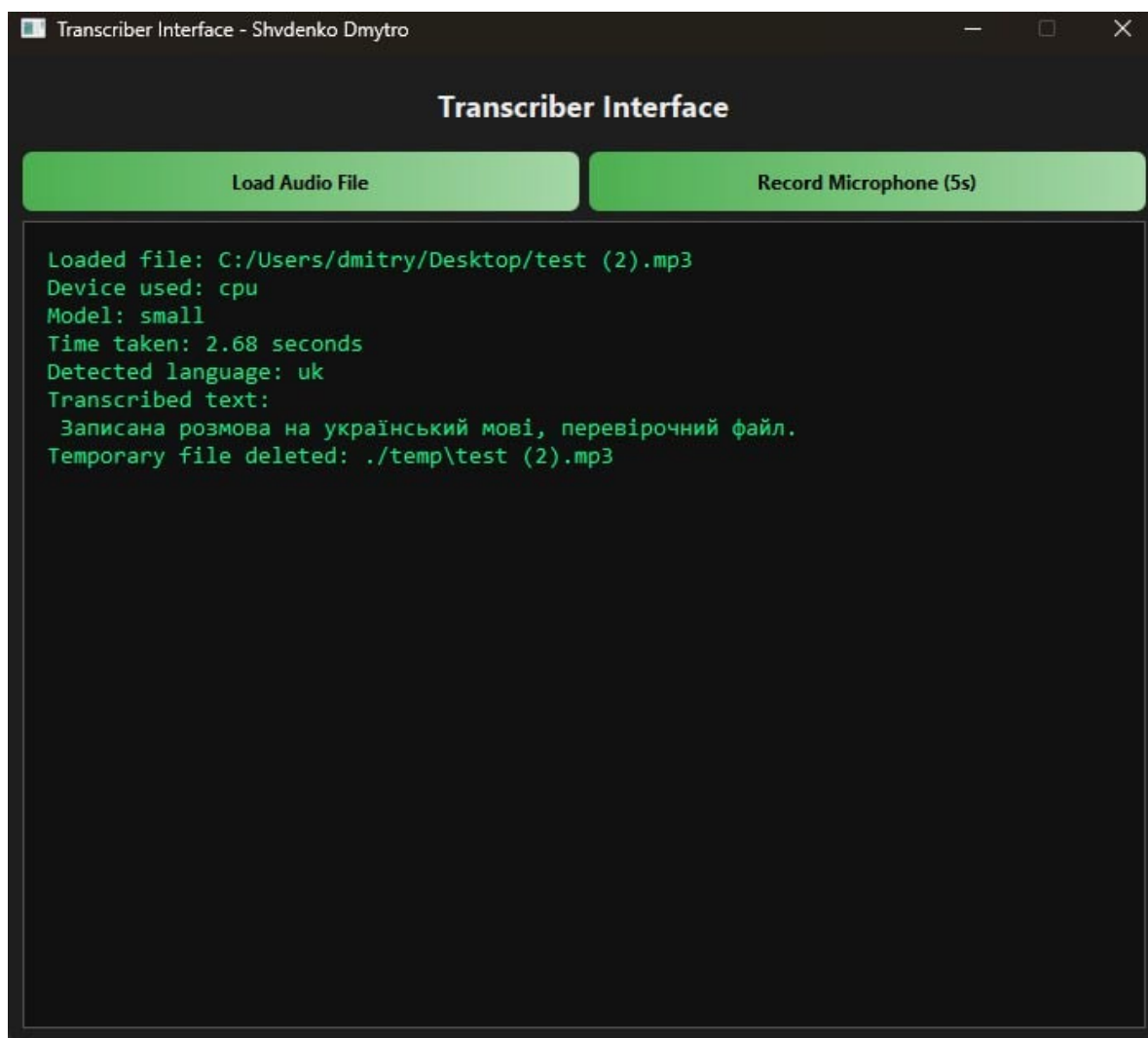


Рисунок 3.10 – Тестування заготовленого запису на CPU

Джерело: розроблено автором самостійно

Результати показали: час транскрипції на GPU становив $\approx 0,92$ секунди, а на CPU $\approx 1,31$ секунди. Це свідчить про те, що GPU забезпечує прискорення приблизно у 1,42 раза. Важливо зазначити, що невелика різниця пояснюється відносно малим розміром тестового аудіо: основний час займає завантаження та ініціалізація моделі, тоді як саме розпізнавання займає лише частку часу. Розрахунки отриманих даних наведено в таблиці нижче.

Таблиця 3.1 – Прогнозований час транскрипції для записів різної тривалості

Тривалість аудіо	CPU (1,31 с на 5 с)	GPU (0,92 с на 5 с)
5 секунд	1,31 с	0,92 с

Продовження таблиці 3.1

Тривалість аудіо	CPU (1,31 с на 5 с)	GPU (0,92 с на 5 с)
30 секунд	$1,31 \times 6 \approx 7,86$ с	$0,92 \times 6 \approx 5,52$ с
1 хвилина	$1,31 \times 12 \approx 15,72$ с	$0,92 \times 12 \approx 11,04$ с
5 хвилин	$1,31 \times 60 \approx 78,6$ с	$0,92 \times 60 \approx 55,2$ с
10 хвилин	$1,31 \times 120 \approx 157,2$ с	$0,92 \times 120 \approx 110,4$ с

Джерело: сформовано автором на основі виконаного дослідження

На рисунку 3.11 наведено умовний графік порівняння часу транскрипції на CPU та GPU. З графіка видно, що GPU стабільно демонструє кращі показники, але вигреш по швидкості при коротких аудіо невеликий, а для довгих записів ефективність зростає.

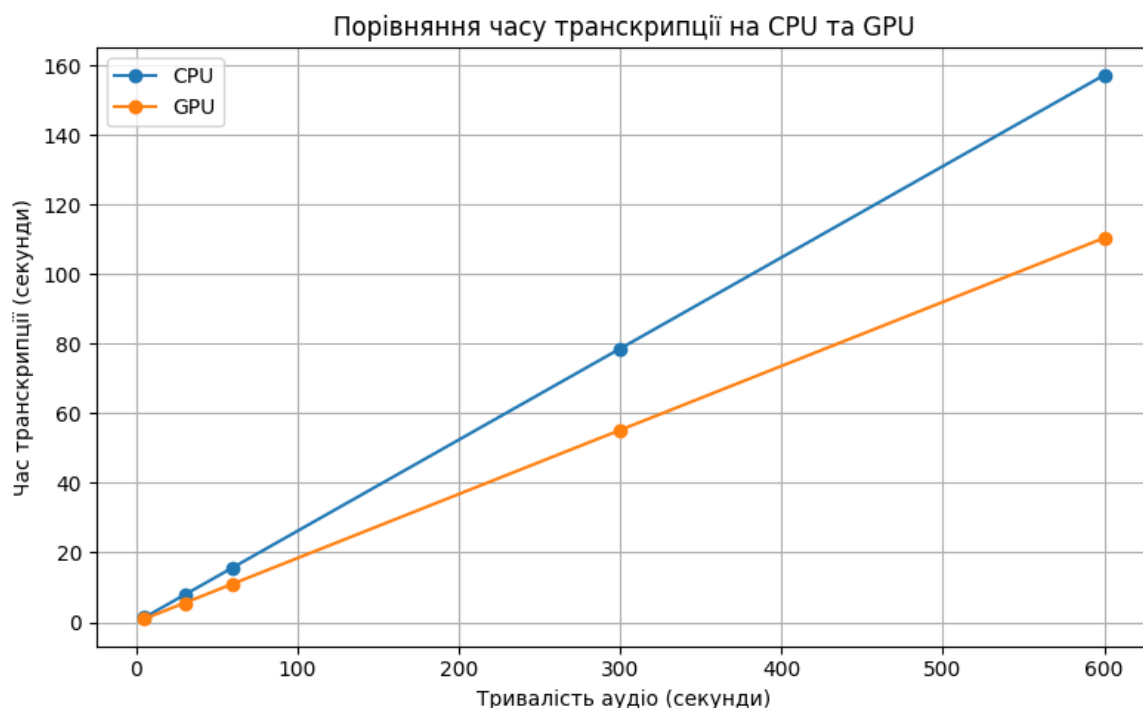


Рисунок 3.11 – Порівняння часу транскрипції на CPU та GPU для різної тривалості аудіо.

Джерело: розроблено автором самостійно

У контексті оцінювання системи ASR ключовим параметром є Word Error Rate (WER) – відсоток неправильно розпізнаних слів у транскрипції. Вимірювання точності на стандартних корпусах дає змогу порівнювати моделі. Згідно з Radford та співавт. (2023), навіть найменша модель Whisper (39 млн параметрів) демонструє

WER $\approx 6.7\%$ на чистому тесті LibriSpeech Clean[26]. Більші моделі Whisper, навчені на 680 тис. годин різноманітного даних, досягають ще нижчої помилки (в кращих випадках – близько 2.5% WER)[26]. Це свідчить про високу початкову точність системи. Однак на реальних даних вплив має не лише середовище, але й якість мови. Наприклад, дослідження Graham & Roll (2024) показує, що точність Whisper різко знижується при сильних акцентних відхиленнях: найкращий результат – для американського акценту англійської, дещо гірший – для британського чи австралійського, а носії з нерідною англійською мають набагато вищий WER[27]. Крім того, розпізнавання читаного мовлення значно точніше за розпізнавання спонтанної розмови, причому помітні також відмінності за статтю мовця[27].

На точність розпізнавання впливає і тривалість аудіо. Модель Whisper навчається на фрагментах по 30 сек, тому для довгих записів застосовується посегментна обробка: аудіо послідовно ділиться на 30-секундні вікна, і кожне декодується окремо з використанням технік beam search та регулювання температури моделі[26]. Експериментальні результати показують, що така стратегія дозволяє стабільно транскрибувати багатохвилинні й багатогодинні аудіозаписи. Зокрема, на наборі довготривалих даних Whisper послідовно перевершує відкриті альтернативи: розподіл WER моделі Whisper у кожному датасеті виявляється нижчим, ніж у інших open-source ASR, а в більшості випадків – навіть кращим за комерційні сервіси[26]. Це свідчить про те, що для великих аудіо Whisper зберігає високу точність завдяки грамотному підходу до сегментації і декодування[26].

Нарешті, істотний вплив на WER має рівень шуму у записі. Дослідження Radford та співавт. також демонструє, що зі зменшенням SNR (збільшенням шуму) помилка Whisper зростає, але значно повільніше, ніж у моделей, натренованих на чистих даних LibriSpeech[26]. Наприклад, при додаванні «кафе-шуму» до записів Whisper перевершує більшість спеціалізованих моделей уже при $\text{SNR} \approx 10$ дБ і нижче[26]. Відтак Whisper виявляється відносно стійким до фонових перешкод, що пояснюється великим розмаїттям навчальних даних. Це узгоджується з висновками

іншого дослідження: модель, навчена на різноманітних даних, здатна автоматично кодувати фон і адаптуватися до нього, забезпечуючи підвищену шумостійкість[28].

Таким чином, аналіз показує, що Whisper small забезпечує високу точність транскрипції для якісних записів (низький WER на коротких чистих аудіо), але чутливий до шуму і особливостей мови носія. Наприклад, у мовленні з сильним шумом або з нетиповим акцентом частка помилок помітно зростає[26][27]. Одночасно, при використанні техніки сегментації моделі демонструють високу стабільність на довгих аудіо[26]. Загалом, результати вказують, що Whisper small має низький WER на еталонних наборах даних і зберігає високу точність у «важких» умовах (шум, акценти), хоча й із збільшенням помилок у цих випадках[26][27][28].

Висновки до розділу 3

Проведене дослідження показало, що GPU дає прискорення приблизно у 1,4 рази при роботі з короткими аудіозаписами, а при довготривалих задачах очікується ще більше пришвидшення завдяки ефективнішому використанню паралельних обчислень. Оскільки система працює локально, час транскрипції залежить виключно від обчислювальних ресурсів ПК, що дозволяє прогнозувати стабільну роботу незалежно від зовнішніх чинників.

ВИСНОВКИ

У роботі було розроблено програмний застосунок для автоматичного розпізнавання аудіо з використанням сучасних моделей машинного навчання.

У першому розділі було проведено детальний аналіз основних концепцій, підходів і теорій, що лежать в основі систем автоматичного розпізнавання мовлення. Розглянуто сучасні self-supervised моделі, такі як Wav2Vec 2.0 та Whisper, а також інструменти Kaldi, DeepSpeech, SpeechBrain, torchaudio й librosa, що широко використовуються в наукових і практичних розробках. Це дозволило визначити ключові вимоги:

- висока точність розпізнавання,
- підтримка української мови без складного донавчання,
- можливість роботи на локальних комп'ютерах,
- простота інтеграції та використання

У другому розділі проведено комплексне проектування системи. Визначено основні вимоги до системи, побудовано концептуальну модель, описано взаємодію компонентів (ML-модель, модулі аудіообробки, графічний інтерфейс, багатопотокове управління). Також обґрунтовано вибір технологій, що дозволяють досягти балансу між продуктивністю, масштабованістю та простотою використання. Система спроектована з урахуванням майбутнього розширення (наприклад, інтеграції голосового асистента або серверних компонентів).

У третьому розділі детально описано проєктні рішення та реалізацію системи, включно з архітектурою, графічним інтерфейсом і технічними аспектами. Особливу увагу приділено порівнянню продуктивності системи на CPU та GPU. Результати експериментів показали, що GPU забезпечує значно вищу швидкість обробки (прискорення приблизно в 1,4 раза для коротких аудіо та ще більше для довгих завдань), що робить його доцільним для задач реального часу або великих обсягів даних. Також підтверджено, що система здатна стабільно працювати локально, без залежності від мережевих затримок.

У роботі досягнуто поставлену мету - створено застосунок, який:

- використовує відкриті ML-моделі для розпізнавання україномовного аудіо,
- забезпечує простий графічний інтерфейс,
- працює локально як на CPU, так і на GPU,
- демонструє ефективність та придатність до подальшого розширення.

Практична значимість отриманих результатів полягає в можливості використання розробленого застосунку в освітніх, наукових, корпоративних і побутових умовах. Це відкриває перспективи для додаткових досліджень та вдосконалень, таких як інтеграція голосових команд, мультимодальних інтерфейсів або серверних рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Young, S. et al. (2006). The HTK Book (for HTK Version 3.4). Cambridge University Engineering Department. [Електронний ресурс]. – Режим доступу: https://speech.ee.ntu.edu.tw/homework/DSP_HW2-1/htkbook.pdf – Дата звернення: 12.04.2025.
2. Hannun, A. et al. (2014). Deep Speech: Scaling up end-to-end speech recognition. [Електронний ресурс]. – Режим доступу: <https://arxiv.org/pdf/1412.5567v2> – Дата звернення: 12.04.2025.
3. Baevski, A., Zhou, Y., Mohamed, A., & Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. [Електронний ресурс]. – Режим доступу: <https://arxiv.org/pdf/2006.11477> – Дата звернення: 12.04.2025.
4. Radford, A. et al. (2022). Robust Speech Recognition via Large-Scale Weak Supervision. [Електронний ресурс]. – Режим доступу: <https://openai.com/blog/whisper/> – Дата звернення: 12.04.2025.
5. Baevski, A., Zhou, Y., Mohamed, A., & Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. [Електронний ресурс]. – Режим доступу: <https://arxiv.org/pdf/2006.11477> – Дата звернення: 12.04.2025.
6. Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Sutskever, I. (2022). Robust Speech Recognition via Large-Scale Weak Supervision. [Електронний ресурс]. – Режим доступу: <https://cdn.openai.com/papers/whisper.pdf> – Дата звернення: 12.04.2025.
7. Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., ... & Veselý, K. (2011). The Kaldi Speech Recognition Toolkit. [Електронний ресурс]. – Режим доступу: https://www.danielpovey.com/files/2011_asru_kaldi.pdf – Дата звернення: 12.04.2025.
8. Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., ... & Ng, A. Y. (2014). Deep Speech: Scaling up end-to-end speech recognition. [Електронний

ресурс]. – Режим доступа: <https://arxiv.org/abs/1412.5567> – Дата звернення: 12.04.2025.

9. Baevski, A., Zhou, Y., Mohamed, A., & Auli, M. (2020). wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. [Електронний ресурс]. – Режим доступа: <https://arxiv.org/abs/2006.11477> – Дата звернення: 12.04.2025.

10. Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Sutskever, I. (2022). Robust Speech Recognition via Large-Scale Weak Supervision. [Електронний ресурс]. – Режим доступа: <https://cdn.openai.com/papers/whisper.pdf> – Дата звернення: 12.04.2025.

11. Ravanelli, M., Parcollet, T., Plantinga, P., Cornell, S., Lugosch, L., Subramanian, A., ... & Bengio, Y. (2021). SpeechBrain: A General-Purpose Speech Toolkit. [Електронний ресурс]. – Режим доступа: <https://arxiv.org/abs/2106.04624> – Дата звернення: 12.04.2025.

12. Yang, Y.-Y., Hira, M., Ni, Z., Chourdia, A., Astafurov, A., Chen, C., ... & Watanabe, S. (2021). TorchAudio: Building Blocks for Audio and Speech Processing. [Електронний ресурс]. – Режим доступа: <https://arxiv.org/abs/2110.15018> – Дата звернення: 12.04.2025.

13. McFee, B., Raffel, C., Liang, D., Ellis, D. P. W., McVicar, M., Battenberg, E., & Nieto, O. (2015). librosa: Audio and music signal analysis in python. [Електронний ресурс]. – Режим доступа: <https://librosa.org/doc/main/tutorial.html> – Дата звернення: 12.04.2025.

14. Povey, D. et al. (2011). The Kaldi Speech Recognition Toolkit. Proc. IEEE ASRU 2011 (Automatic Speech Recognition and Understanding), Waikoloa, HI, Dec. 2011. – [Електронний ресурс]. Режим доступа: http://kaldi-asr.org/doc/kaldi_asru2011.pdf – Дата звернення: 09.05.2025.

15. Hannun, A. et al. (2014). Deep Speech: Scaling up end-to-end speech recognition. – [Електронний ресурс]. Режим доступа: <https://arxiv.org/pdf/1412.5567> – Дата звернення: 09.05.2025.

16. Baevski, A., Zhou, Y., Mohamed, A., & Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. – [Електронний ресурс]. Режим доступу: <https://arxiv.org/pdf/2006.11477> – Дата звернення: 09.05.2025.
17. Radford, A. et al. (2022). Robust Speech Recognition via Large-Scale Weak Supervision (Whisper від OpenAI). – [Електронний ресурс]. Режим доступу: <https://openai.com/research/whisper> – Дата звернення: 09.05.2025.
18. Ravanelli, M. et al. (2024). Open-Source Conversational AI with SpeechBrain 1.0. *Journal of Machine Learning Research*, 25(333): 1–11.
19. Швиденко Д. Ю. (2025). Звіт з переддипломної практики (24.02–11.05.2025) – КНЕУ ім. В. Гетьмана, Київ, 2025.
20. Radford, A. et al. (2022). Whisper: Robust Speech Recognition via Large-Scale Weak Supervision. [Електронний ресурс]. – Режим доступу: <https://cdn.openai.com/papers/whisper.pdf> – Дата звернення: 12.04.2025.
21. Povey, D. et al. (2011). The Kaldi Speech Recognition Toolkit. [Електронний ресурс]. – Режим доступу: https://publications.idiap.ch/downloads/papers/2012/Povey_ASRU2011_2011.pdf – Дата звернення: 12.04.2025.
22. Hannun, A. et al. (2014). Deep Speech: Scaling up end-to-end speech recognition. [Електронний ресурс]. – Режим доступу: <https://arxiv.org/pdf/1412.5567v2> – Дата звернення: 12.04.2025.
23. Baevski, A. et al. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. [Електронний ресурс]. – Режим доступу: <https://arxiv.org/pdf/2006.11477> – Дата звернення: 12.04.2025.
24. McFee, B. et al. (2015). librosa: Audio and music signal analysis in Python. [Електронний ресурс]. – Режим доступу: https://brianmcfee.net/papers/scipy2015_librosa.pdf – Дата звернення: 12.04.2025.
25. Adobe XD (2016). Get started [Електронний ресурс]. – Режим доступу: <https://helpx.adobe.com/ua/xd/get-started.html> – Дата звернення: 12.04.2025.

26 Radford A., Kim J. W., Xu T., Brockman G., McLeavey C., Sutskever I. Robust Speech Recognition via Large-Scale Weak Supervision. – Proc. ICML, 2023.

27 Graham C., Roll N. Evaluating OpenAI's Whisper ASR: Performance analysis across diverse accents and speaker traits. – JASA Express Lett., 4(2): 025206, 2024.

28 Gong Y., Khurana S., Karlinsky L., Glass J. Whisper-AT: Noise-Robust Automatic Speech Recognizers are Also Strong General Audio Event Taggers. – arXiv:2307.03183, 2023.

29 Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovský, J., Stemmer, G., & Veselý, K. (2011). The Kaldi Speech Recognition Toolkit. [Электронный ресурс]. – Режим доступа: https://www.danielpovey.com/files/2011_asru_kaldi.pdf – Дата звернення: 08.04.2025.

30 Mozilla (2023). DeepSpeech: open-source Speech-To-Text engine. [Электронный ресурс]. – Режим доступа: <https://deepspeech.readthedocs.io/> – Дата звернення: 08.04.2025.

31 Baevski, A., Zhou, Y., Mohamed, A., & Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. [Электронный ресурс]. – Режим доступа: <https://arxiv.org/pdf/2006.11477> – Дата звернення: 08.04.2025.

32 PyTorch (2024). Torchaudio documentation (2.7.0). [Электронный ресурс]. – Режим доступа: <https://pytorch.org/audio/stable/index.html> – Дата звернення: 08.04.2025.

ДОДАТКИ

Додаток А

```

import sys
import os
import shutil
import whisper
import librosa
import time
import torch
import warnings
import sounddevice as sd
import soundfile as sf

from PySide6.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout, QPushButton, QTextEdit,
    QFileDialog, QLabel, QHBoxLayout
)
from PySide6.QtCore import Qt, QThread, Signal

TEMP_DIR = "./temp"
RECORD_SECONDS = 5
RECORD_PATH = os.path.join(TEMP_DIR, "recorded.wav")

class AudioTranscriber:
    def __init__(self, model_name='small'):
        self.model_name = model_name
        self.device = 'cpu' if torch.cuda.is_available() else 'cpu'

        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=FutureWarning)
            self.model = whisper.load_model(self.model_name, device=self.device)

    def transcribe(self, audio_path, target_sr=16000):
        audio, sr = librosa.load(audio_path, sr=target_sr, mono=True)
        start_time = time.time()
        result = self.model.transcribe(audio=audio, language='uk')
        elapsed = time.time() - start_time
        return result['language'], result['text'], elapsed

class RecordThread(QThread):
    finished = Signal(str)

    def run(self):
        if not os.path.exists(TEMP_DIR):
            os.makedirs(TEMP_DIR)
        recording = sd.rec(int(RECORD_SECONDS * 16000), samplerate=16000, channels=1, dtype='int16')
        sd.wait()
        sf.write(RECORD_PATH, recording, 16000)
        self.finished.emit(RECORD_PATH)

class TranscribeThread(QThread):
    finished = Signal(str, str, float, str)

    def __init__(self, transcriber, file_path):
        super().__init__()
        self.transcriber = transcriber

```

```

self.file_path = file_path

def run(self):
    try:
        lang, text, elapsed = self.transcriber.transcribe(self.file_path)
        self.finished.emit(lang, text, elapsed, None)
    except Exception as e:
        self.finished.emit(None, None, None, str(e))

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Transcriber Interface - Shvdenko Dmytro")
        self.setFixedSize(700, 600)

        self.transcriber = AudioTranscriber()
        self.init_ui()

    def init_ui(self):
        self.setStyleSheet("""
            QMainWindow {
                background-color: #1e1e1e;
            }
            QPushButton {
                background-color: qlineargradient(x1:0, y1:0, x2:1, y2:0,
                                                    stop:0 #4caf50, stop:1 #a5d6a7);

                color: black;
                font-weight: bold;
                padding: 10px;
                border-radius: 6px;
            }
            QPushButton:hover {
                background-color: #66bb6a;
            }
            QLabel {
                color: #eeeeee;
                font-size: 18px;
                font-weight: bold;
                margin: 10px;
            }
            QTextEdit {
                background-color: #111;
                color: #00ff88;
                font-family: Consolas;
                font-size: 14px;
                padding: 10px;
                border: 1px solid #555;
            }
        """)

        central_widget = QWidget()
        self.setCentralWidget(central_widget)

        layout = QVBoxLayout()

        self.label = QLabel("Transcriber Interface")
        self.label.setAlignment(Qt.AlignCenter)
        layout.addWidget(self.label)

        button_layout = QHBoxLayout()

```

```

self.load_btn = QPushButton("Load Audio File")
self.load_btn.clicked.connect(self.load_file)
button_layout.addWidget(self.load_btn)

self.record_btn = QPushButton(f"Record Microphone ({RECORD_SECONDS}s)")
self.record_btn.clicked.connect(self.start_recording)
button_layout.addWidget(self.record_btn)

layout.addLayout(button_layout)

self.console = QTextEdit()
self.console.setReadOnly(True)
layout.addWidget(self.console)

central_widget.setLayout(layout)

def log(self, message: str):
    self.console.append(message)
    self.console.repaint()

def clear_console(self):
    self.console.clear()

def set_buttons_enabled(self, enabled: bool):
    self.load_btn.setEnabled(enabled)
    self.record_btn.setEnabled(enabled)

def load_file(self):
    file_path, _ = QFileDialog.getOpenFileName(
        self, "Select Audio File", "", "Audio Files (*.mp3 *.wav *.m4a *.flac)"
    )
    if file_path:
        self.clear_console()
        temp_path = os.path.join(TEMP_DIR, os.path.basename(file_path))
        shutil.copy(file_path, temp_path)

        self.log(f"Loaded file: {file_path}")
        self.start_transcription(temp_path, cleanup_after=True)

def start_recording(self):
    self.clear_console()
    self.log("Recording audio from microphone...")
    self.set_buttons_enabled(False)
    self.record_thread = RecordThread()
    self.record_thread.finished.connect(self.process_recording)
    self.record_thread.start()

def process_recording(self, file_path):
    self.log("Recording finished.")
    self.start_transcription(file_path, cleanup_after=True)

def start_transcription(self, file_path: str, cleanup_after=False):
    self.set_buttons_enabled(False)
    self.transcribe_thread = TranscribeThread(self.transcriber, file_path)
    self.transcribe_thread.finished.connect(
        lambda lang, text, elapsed, error: self.process_transcription(
            file_path, lang, text, elapsed, error, cleanup_after
        )
    )
    self.transcribe_thread.start()

def process_transcription(self, file_path, lang, text, elapsed, error, cleanup_after):

```

```

if error:
    self.log(f"[Error] Transcription failed: {error}")
else:
    self.log(f"Device used: {self.transcriber.device}")
    self.log(f"Model: {self.transcriber.model_name}")
    self.log(f"Time taken: {elapsed:.2f} seconds")
    self.log(f"Detected language: {lang}")
    self.log("Transcribed text:")
    self.log(text)
if cleanup_after and os.path.exists(file_path):
    os.remove(file_path)
    self.log(f"Temporary file deleted: {file_path}")
self.set_buttons_enabled(True)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())

```

```

import whisper
import os
import shutil

class ModelDownloader:
    def __init__(self, model_name='small', save_path='./model'):
        # tiny, base, small, medium, large
        self.model_name = model_name
        self.save_path = os.path.expanduser(save_path)
        os.makedirs(self.save_path, exist_ok=True)

    def load(self):
        model_file = os.path.join(self.save_path, f'{self.model_name}.pt')

        if os.path.exists(model_file):
            print(f'{self.model_name} - {model_file}')
            model = whisper.load_model(model_file)
        else:
            print(f' download model {self.model_name}...')
            model = whisper.load_model(self.model_name)

            # Путь из кэша
            cache_dir = os.path.expanduser("~/cache/whisper")
            source_file = os.path.join(cache_dir, f'{self.model_name}.pt')

            if os.path.exists(source_file):
                shutil.copy2(source_file, model_file)
                print(f' copy {model_file}')
            else:
                print(f' not found {source_file}')

        return model

if __name__ == '__main__':
    downloader = ModelDownloader(
        model_name='small',
        save_path='./models'
    )
    model = downloader.load()

```



Дата звіту5/14/2025

Дата редагування---



Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kyiv National Economic University named after Vadym Hetman KNEU

Заголовок
Розроблення застосунку для розпізнавання аудіо з використанням інструментарію машинного навчання

Автор
Науковий керівник / Експерт
Швиденко Дмитро ЮрійовичВасильєва Л.В

підрозділ
кафедра інформаційних систем в економіці

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		4
Інтервали		0
Мікропробіли		8
Білі знаки		0
Парафрази (SmartMarks)		1

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Копір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз		Копір тексту
ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://kneu.edu.ua/userfiles/Department_of_Administration_and_Marketing_Personn/Titulna_storD196nk_a_KMD0A0.doc	21 0.20 %
2	Розробка інтегрованої інформаційної системи нарахування заробітної плати на карткові рахунки 6/19/2020 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	20 0.19 %

3	https://ru.essays.club/%D0%A2%D0%BE%D1%87%D0%BD%D1%8B%D0%B5-%D0%BD%D0%B0%D1%83%D0%BA%D0%B8/%D0%9A%D0%BE%D0%BC%D0%BC%D1%83%D0%BD%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8-%D0%B8-%D1%81%D0%B2%D1%8F%D0%B7%D1%8C/%D0%97%D0%B0%D1%85%D0%B8%D1%81%D1%82-%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97-%D0%B7-%D0%BE%D0%B1%D0%BC%D0%B5%D0%B6%D0%B5%D0%BD%D0%B8%D0%BC-470343.html	9 0.09 %
4	https://ru.essays.club/%D0%A2%D0%BE%D1%87%D0%BD%D1%8B%D0%B5-%D0%BD%D0%B0%D1%83%D0%BA%D0%B8/%D0%9A%D0%BE%D0%BC%D0%BC%D1%83%D0%BD%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8-%D0%B8-%D1%81%D0%B2%D1%8F%D0%B7%D1%8C/%D0%97%D0%B0%D1%85%D0%B8%D1%81%D1%82-%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97-%D0%B7-%D0%BE%D0%B1%D0%BC%D0%B5%D0%B6%D0%B5%D0%BD%D0%B8%D0%BC-470343.html	6 0.06 %
5	HIGHER EDUCATION DEVELOPMENT TRENDS IN THE CONTEXT OF EUROPEAN INTEGRATION PROCESSES Наталія Мачинська;	5 0.05 %
6	https://ru.essays.club/%D0%A2%D0%BE%D1%87%D0%BD%D1%8B%D0%B5-%D0%BD%D0%B0%D1%83%D0%BA%D0%B8/%D0%9A%D0%BE%D0%BC%D0%BC%D1%83%D0%BD%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8-%D0%B8-%D1%81%D0%B2%D1%8F%D0%B7%D1%8C/%D0%97%D0%B0%D1%85%D0%B8%D1%81%D1%82-%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97-%D0%B7-%D0%BE%D0%B1%D0%BC%D0%B5%D0%B6%D0%B5%D0%BD%D0%B8%D0%BC-470343.html	5 0.05 %
з бази даних RefBooks (0.05 %)		
порядковий НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
джерело: Paperity		
1	HIGHER EDUCATION DEVELOPMENT TRENDS IN THE CONTEXT OF EUROPEAN INTEGRATION PROCESSES Наталія Мачинська;	5 (1) 0.05 %
з домашньої бази даних (0.19 %)		
порядковий НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка інтегрованої інформаційної системи нарахування заробітної плати на карткові рахунки 6/19/2020 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	20 (1) 0.19 %
з програми обміну базами даних (0.00 %)		
порядковий НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
з Інтернету (0.39 %)		
порядковий НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://kneu.edu.ua/userfiles/Department_of_Administration_and_Marketing_Personn/Титулна_storD196nk_a_KMD0A0.doc	21 (1) 0.20 %

2	https://ru.essays.club/%D0%A2%D0%BE%D1%87%D0%BD%D1%8B%D0%B5-%D0%BD%D0%B0%D1%83%D0%BA%D0%B8/%D0%9A%D0%BE%D0%BC%D0%BC%D1%83%D0%BD%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8-%D0%B8-%D1%81%D0%B2%D1%8F%D0%B7%D1%8C/%D0%97%D0%B0%D1%85%D0%B8%D1%81%D1%82-%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97-%D0%B7-%D0%BE%D0%B1%D0%BC%D0%B5%D0%B6%D0%B5%D0%BD%D0%B8%D0%BC-470343.html	20 (3) 0.19 %
---	---	---------------

Список принятых фрагментів (немає принятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)
------------------	-------	---------------------------------------

Копія тез доповіді

Васильєва Л.В., Швиденко Д. Ю. Основні проблеми обробки аудіосигналів в системах машинного навчання // Сучасні інформаційні технології та системи в управлінні [Електронний ресурс]: Зб. матеріалів VI Міжнар. наук.-практ. конф. молодих вчених, аспірантів і студентів; 10–11 квітня 2025 р. — Київ: КНЕУ, 2025.

Васильєва Л.В., к.т.н., доцент
*Київський національний економічний
університет імені Вадима Гетьмана*
liudmyla_vasylieva@kneu.edu.ua

Швиденко Д. Ю., здобувач другого (магістерського) рівня вищої освіти
*Київський національний економічний
університет імені Вадима Гетьмана*
rapax.deus@gmail.com

ОСНОВНІ ПРОБЛЕМИ ОБРОБКИ АУДІОСИГНАЛІВ В СИСТЕМАХ МАШИННОГО НАВЧАННЯ

Обробка аудіосигналів є критичною складовою сучасних систем машинного навчання та штучного інтелекту, що застосовуються в розпізнаванні мовлення, музики та інших аудіоджерел. Незважаючи на значний прогрес у цій галузі, існує низка технічних проблем, які потребують подальшого дослідження та вирішення.

Основні проблеми обробки аудіосигналів включають: обмеженість даних для навчання; варіативність мовлення та акцентів; обробка мовлення в шумних середовищах; етичні та культурні аспекти обробки мовлення; обчислювальні ресурси та енергоефективність; інтеграція з іншими модальностями.

Однією з головних проблем є обмеженість анотованих даних для навчання моделей. Для багатьох мов та діалектів такі ресурси або відсутні, або представлені у недостатньому обсязі, що значно ускладнює створення ефективних систем розпізнавання [1]. Основними причинами цього є висока вартість збору й анотації аудіоданих, юридичні обмеження на використання мовних корпусів, а також недостатній рівень цифровізації малопоширених мов. Крім того, якість доступних аудіокорпусів часто є неоднорідною, що може призводити до перекосів у навчанні моделей. Наприклад, більшість загальнодоступних мовних баз даних містять переважно стандартні акценти й вимову, у той час як діалектні варіації або мови з низьким ресурсним забезпеченням представлені фрагментарно. Це ускладнює розробку універсальних алгоритмів, здатних працювати з широким спектром мовних варіацій [1]. Один із підходів до вирішення цієї проблеми – використання методів перенесення навчання (transfer learning), коли моделі, попередньо натреновані на великих обсягах даних основних мов, адаптуються до мов із меншою кількістю даних. Також перспективним напрямом є few-shot learning, що дозволяє нейронним мережам навчатися на обмеженому обсязі прикладів, завдяки чому можна розробляти точні системи розпізнавання навіть для рідкісних мов. Також сучасні системи часто потребують інтеграції з іншими модальностями, такими як зображення або текст, що дозволяє покращити продуктивність та забезпечити більш природну взаємодію з користувачем [1].

Сучасні підходи до вирішення цієї проблеми включають використання методів перенесення навчання, few-shot learning, а також генеративних моделей для синтезу додаткових даних. Наприклад, методи самонавчання дозволяють використовувати великі обсяги анотованих аудіоданих, навчаючи моделі розпізнавати мовлення за допомогою псевдо-мічених даних [2]. Дослідження також показують, що застосування комбінованих підходів, таких як weakly-

supervised learning та semi-supervised learning, може значно покращити якість розпізнавання мовлення в умовах обмежених анотованих вибірок.

Додатковим напрямом є використання мультимодального навчання, коли аудіодані доповнюються текстовими та візуальними підказками. Це дозволяє компенсувати нестачу чисто мовних анотацій, використовуючи відповідні текстові транскрипції або навіть відеозаписи для покращення якості навчання [3]. Одним із перспективних підходів є інтеграція активного шумозаглушення на основі плазмових технологій. Такі системи аналізують навколишній шум та адаптивно регулюють акустичне середовище, що покращує якість розпізнавання мовлення навіть у складних умовах, наприклад, у транспорті чи гучних публічних місцях. Поєднання цих методів із мультимодальним навчанням, коли розпізнавання базується не лише на аудіосигналах, а й на текстових та візуальних даних, дає змогу суттєво підвищити точність мовних моделей [3]. Такий підхід уже довів свою ефективність у завданнях автоматичного перекладу та розпізнавання мовлення в реальному часі.

Окремим викликом є етичні аспекти, пов'язані з обробкою мовлення, зокрема необхідність збереження мовних меншин та культурної спадщини [4]. Наприклад, технології розпізнавання мовлення для мов корінних народів потребують спеціального підходу, щоб уникнути експлуатації з боку великих технологічних компаній.

Обчислювальні ресурси та енергоефективність є ключовими чинниками при розробці систем обробки мовлення. Високоточні алгоритми машинного навчання потребують значних обчислювальних потужностей, що робить їхнє використання на вбудованих системах, мобільних пристроях та IoT-рішеннях складним завданням. Одним із підходів до зниження навантаження є застосування малорозмірних нейронних мереж та методів квантування параметрів моделей, що дозволяє скоротити споживання енергії та зменшити затримки в обробці даних [5].

Окрім цього, дослідження показують, що використання адаптивних моделей, які динамічно змінюють свою складність залежно від ресурсів пристрою, може значно підвищити енергоефективність. Наприклад, системи, що працюють на мобільних процесорах, можуть автоматично вибрати менш енергозатратні архітектури без значного зниження якості розпізнавання мовлення. Додатково, застосування хмарних обчислень дозволяє передавати обробку складних аудіосигналів на віддалені сервери, розвантажуючі локальні пристрої [5].

Ще одним перспективним напрямом є інтеграція сучасних алгоритмів розрідження (pruning) і значення (distillation), що дозволяє зменшити обсяг нейронних мереж без втрати їхньої точності. Це відкриває можливості для розгортання мовних моделей на енергообмежених платформах, таких як розумні колонки, мобільні телефони та пристрої носимого електронного обладнання. Загалом, поєднання оптимізаційних методів навчання та спеціалізованого апаратного забезпечення сприяє створенню продуктивніших і менш енергозатратних рішень у сфері розпізнавання мовлення [5].

Вирішення цих викликів потребує комплексного підходу, що поєднує технічні інновації, етичні міркування та міждисциплінарні дослідження. Майбутній розвиток у цій сфері передбачає не лише вдосконалення алгоритмів, а й подальшу інтеграцію мультимодальних технологій, що поєднують аудіо, текст та відео для покращення точності розпізнавання.

Список використаних джерел

1. Arxiv.org. A Review of Deep Learning Techniques for Speech Processing: <https://arxiv.org/abs/2305.00359>
2. Houston Chronicle. OK, Google. Time to learn Spanglish. | Opinion: <https://www.houstonchronicle.com/opinion/outlook/article/ai-apple-google-siri-language-processing-spanish-20007656.php>
3. Lifewire. How New Noise Cancellation Tech Could Finally Bring Us a Little Peace and Quiet: <https://www.lifewire.com/new-plasma-based-active-noise-cancellation-tech-7558939>
4. Wired. Māori are trying to save their language from Big Tech: <https://www.wired.com/story/maori-language-tech>
5. Springer. Deep neural networks for automatic speech processing: a survey from large corpora to limited data: <https://link.springer.com/article/10.1186/s13636-022-00251-w>