МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ТРАНСФОРМАЦІЇ ГОЛОСУ

Здобувача вищої освіти другого (магістерського) рівня,
ІІ курсу, групи КМІ-24-2, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології в економіці»

(підпис) Бектімиров Абдуль Алим Маратович
(прізвище, ім'я, по батькові)

Науковий керівник (підпис) Гладченко Оксана Вікторівна, канд. пед. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-професійної програми (підпис) Омельчук Антон Анатолійович, канд. техн. наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри протокол від «02» січня 2026 № 10

Завідувач кафедри (підпис) Філоненко Михайло Миколайович, канд. фіз.-мат. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь – 2026

Реєстрація №19 21.01.2026 _____
(дата) (підпис відповідальної особи)

Результати перевірки до захисту 02.01.2026 _____
(до захисту, на доопрацювання) (дата) (підпис)

Результат захисту 4 (добре) 04.02.2026 _____
(оцінка) (дата) (підпис)

Голова комісії: _____ О.С. Дюбанов
(підпис)

Члени комісії: _____ А.Ю. Горбовий
(підпис)

_____ М.М. Філоненко
(підпис)

_____ І.Д. Погорєловська
(підпис)

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст кваліфікаційної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	24		04.02.2026 Дюбанов О.С. Горбовий А.Ю.
2. Дотримання вимог щодо оформлення	від 0 до 10	8		Філоненко М.М.
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	4		Погорєловська І.Д. Суховецька Л.П.
4. Рецензент	від 0 до 5	4		
РАЗОМ БАЛІВ	від 0 до 50	40		

Оцінювання кваліфікаційної роботи науковим керівником – 40 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
80	B	4	04.02.2026	Дюбанов О.С. Горбовий А.Ю. Філоненко М.М. Погорєловська І.Д. Суховецька Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань 12 «Інформаційні технології»
Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Інформаційні управляючі системи і технології в економіці»

Затверджую
Завідувач кафедри _____ Філоненко М.М.
«11» грудня 2024

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Бектімирову Абдуль Алиму Маратовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи Розробка інформаційної системи для трансформації голосу.
Затверджена наказом ректора від «10» грудня 2024 № 1852
2. Строк здачі закінченої роботи «02» січня 2026
3. Мета роботи: Створення мобільного застосунку для локальної трансформації голосу на Android.
Об'єкт дослідження: Процес перетворення голосу на мобільних пристроях.
Предмет дослідження: Методи оптимізації нейромереж для мобільних платформ.
Орієнтовна база дослідження: Документація PyTorch, NCNN, Android SDK.
4. Вихідні дані до роботи: Відкриті реалізації архітектури RVC та її компонентів (HuBERT, RMVPE) на мові Python з використанням бібліотеки PyTorch. Попередньо навчені ваги моделей (.pth).
5. Перелік графічного матеріалу: Архітектурні блок-схеми, скриншоти інтерфейсу та таблиці результатів тестування.
6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)
Вступ; Розділ 1. Аналіз технологій конверсії голосу та проблематика їх інтеграції на мобільних платформах; Розділ 2. Проєктування архітектури системи локальної трансформації аудіопотоку; Розділ 3. Програмна реалізація мобільного застосунку та експериментальна перевірка; Висновки; Список використаних джерел; Додатки.

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 15.12.2024	До 15.12.2024
2	**І розділ** (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	30.05.2025	30.05.2025
3	**ІІ розділ** (аналіз експериментальної частини про методики досліджень)	10.09.2025	10.09.2025
4	**ІІІ розділ** (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	20.11.2025	20.11.2025
5	**Висновки**	11.12.2025	11.12.2025
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	25.12.2025	25.12.2025
7	Отримання відгуку керівника	16.01.2026	16.01.2026
8	Отримання зовнішньої рецензії	19.01.2026	19.01.2026

8. Дата видачі завдання «11» грудня 2024.

9. Керівник кваліфікаційної роботи _____ (підпис) Гладченко О.В. (прізвище, ініціали)

10. Керівник освітньо-професійної програми _____ (підпис) Омельчук А.А. (прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти _____ (підпис) Бектімиров А.М (прізвище, ініціали)

АНОТАЦІЯ

Розробка інформаційної системи для трансформації голосу

Мета роботи полягає в створенні мобільного застосунку для локальної трансформації голосу на Android. *Завданнями роботи* є оптимізація моделей RVC, реалізація гібридного інференсу (NCNN/ONNX), розробка ПЗ та тестування. *Предметом дослідження* є методи оптимізації нейромереж для мобільних платформ. *Об'єктом дослідження* виступає процес перетворення голосу на мобільних пристроях. *Базою дослідження* виступає документація PyTorch, NCNN, Android SDK. *За результатами дослідження* сформульовані методика адаптації важковагових моделей для смартфонів. *Одержані результати* можуть бути використані в медіаіндустрії та для анонімізації.

Кваліфікаційна робота містить 39 сторінок, 2 таблиці, 7 рисунків, список використаних джерел з 40 найменувань, 2 додатки.

Ключові слова: ANDROID, RVC, NCNN, Голос, ШІ.

Рік виконання кваліфікаційної роботи: 2026.

Рік захисту роботи: 2026.

ABSTRACT

Development of an information system for voice transformation

The purpose of the work is to create a mobile application for local voice transformation on Android. *The tasks of the work* are RVC model optimization, hybrid inference implementation (NCNN/ONNX), software development and testing. *The subject of research* is optimization methods of neural networks for mobile platforms. *The object of research* is the voice conversion process on mobile devices. *The basis of the research* is PyTorch, NCNN, Android SDK documentation. *Based on the results of the research*, a methodology for adapting heavy models for smartphones was formulated. *The obtained results* can be used in the media industry and for anonymization.

The qualification work contains 39 pages, 2 tables, 7 figures, a list of references with 40 items, 2 appendices.

Keywords: voice transformation, ANDROID, RVC, NCNN, Voice, AI.

Year of completion: 2026.

Year of defense: 2026

ЗМІСТ

ВСТУП **3**

РОЗДІЛ 1. АНАЛІЗ ТЕХНОЛОГІЙ КОНВЕРСІЇ ГОЛОСУ ТА ПРОБЛЕМАТИКА ЇХ ІНТЕГРАЦІЇ НА МОБІЛЬНИХ ПЛАТФОРМАХ **7**

1.1. Огляд наявних підходів до задачі клонування голосу та їх класифікація 7

1.2. Аналіз обмежень мобільних платформ при роботі з нейромережами та огляд методів оптимізації 9

1.3. Обґрунтування вибору архітектури RVC та засобів кросплатформної розробки 12

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ЛОКАЛЬНОЇ ТРАНСФОРМАЦІЇ АУДІОПОТОКУ **15**

2.1. Архітектура та алгоритми функціонування моделі RVC 15

2.2. Розробка методики конвертації та оптимізації моделей для середовищ ONNX Runtime та NCNN 20

2.3. Розробка алгоритмічного забезпечення та схеми потокової обробки даних 23

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ ТА ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА **26**

3.1. Особливості реалізації аудіопайплайну та інтеграції нейромереж на платформі Android 26

3.2. Аналіз результатів тестування продуктивності системи та якості вихідного сигналу 29

3.3. Перспективи масштабування системи та сфери її практичного застосування 31

ВИСНОВКИ **34**

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ **36**

ДОДАТКИ **40**

ВСТУП

Актуальність теми дослідження. Останні роки відзначилися стрімким розвитком генеративних моделей штучного інтелекту, зокрема в галузі синтезу та обробки аудіо. Технології трансформації голосу (Voice Conversion, VC) відкривають нові можливості для створення контенту, індустрії розваг, персоналізованих асистентів та інструментів для акторів озвучування.

Однак більшість наявних високоякісних рішень для клонування голосу функціонують як хмарні сервіси. Такий підхід має два суттєві недоліки: по-перше, висока затримка (latency), що унеможливлює використання в режимі реального часу, оскільки вимагає надсилання аудіо на сервер та очікування відповіді; по-друге, проблема конфіденційності, адже користувач змушений завантажувати зразки свого голосу (біометричні дані) на сторонні сервери.

Таким чином, розробка локальних інформаційних систем, здатних виконувати складні обчислення для трансформації голосу безпосередньо на пристрої користувача (on-device), є надзвичайно актуальною науково-технічною задачею. Це дозволяє одночасно вирішити проблеми затримки та приватності. Проте, запуск великовагових моделей, таких як RVC, на ресурсно обмежених мобільних платформах, як-от Android, становить значний технічний виклик, що потребує глибокої оптимізації моделей та архітектурних рішень.

Стан і ступінь розробки проблеми в спеціальній літературі. Питання цифрової обробки сигналів та синтезу мовлення є предметом дослідження широкого кола науковців. Фундаментальні основи сучасних методів голосової конверсії (Voice Conversion) та нейронного аудіосинтезу закладені у працях закордонних дослідників. Зокрема, архітектури VITS та HiFi-GAN, що лежать в основі сучасних генераторів, детально описані у роботах Дж. Кіма (J. Kim), Дж. Конга (J. Kong) та ін. [19, 21]. Проблематика самокерованого навчання (Self-Supervised Learning) для екстракції семантичних ознак (модель HuBERT) розкрита у дослідженнях В.-Н. Сю (W.-N. Hsu) та команди Facebook AI Research [13]. Вагомий внесок у розвиток методів оцінки висоти тону (F0

estimation), таких як CREPE та RMVPE, зробили Дж. В. Кім (J. W. Kim) та Х. Вей (H. Wei) [18, 39].

Вагомий внесок у розвиток вітчизняних технологій обробки мовленнєвих сигналів та розпізнавання звукових образів зробили такі вчені, як Т. К. Вінцюк, М. М. Сажок, а також М. І. Шлезінгер, чиї розробки у сфері структурного розпізнавання стали теоретичною базою для систем цифрової обробки сигналів. Проте, у вітчизняній науковій літературі основна увага приділяється переважно задачам розпізнавання мовлення (ASR) або класичному синтезу (TTS), тоді як проблематика нейромережевої трансформації голосу (Voice Conversion) в контексті мобільних обчислень залишається недостатньо висвітленою.

Аналіз наявних IT-проєктів (RVC-Project [31], AI Voice Changer, VoiceMod) свідчить про високий рівень розвитку десктопних та серверних рішень. Водночас сегмент мобільних застосунків представлений переважно:

1. хмарними рішеннями (MagicCall), які залежать від інтернет-з’єднання та мають високу затримку;
2. застарілими DSP-рішеннями (зміна пітчу без зміни тембру), які не забезпечують реалістичності звучання.

Таким чином, попри значний теоретичний доробок, низка практичних питань залишається невирішеною. Зокрема, відсутні комплексні методики адаптації архітектури RVC для запуску на мобільних процесорах (NPU/GPU) з використанням фреймворків NCNN, а також недостатньо досліджені алгоритми оптимізації потокової генерації аудіо в умовах обмежених енергоресурсів смартфонів. Саме цим аспектам: створенню автономної, оптимізованої системи локальної трансформації голосу буде приділена увага у кваліфікаційній роботі.

Метою кваліфікаційної роботи полягає в створенні мобільного застосунку для локальної трансформації голосу на Android. Застосунок має виконувати локальну трансформацію голосу в режимі, близькому до реального часу, на основі оптимізованої нейронної мережі RVC.

Завданнями роботи є оптимізація моделей RVC, реалізація гібридного інференсу (NCNN/ONNX), розробка ПЗ та тестування.

Розглянувши більш детально можна виділити наступні пункти.

1. Провести аналіз сучасних методів та технологій конверсії голосу, їхніх переваг та недоліків.

2. Дослідити обмеження мобільних платформ та обрати оптимальний інструментарій для запуску моделей (інференсу) на пристрої.

3. Обґрунтувати вибір архітектури RVC та її компонентів (HuBERT, RMVPE) як основи для системи.

4. Розробити методику адаптації та конвертації моделей з формату PyTorch у формат для мобільного інференсу.

5. Спроєктувати та програмно реалізувати архітектуру мобільного застосунку на Kotlin, включаючи аудіопайплайн для обробки даних.

6. Провести експериментальне дослідження продуктивності розробленої системи (виміряти затримку, RTF, використання ресурсів).

Об'єктом дослідження виступає процес перетворення голосу на мобільних пристроях.

Предметом дослідження є методи оптимізації нейромереж для мобільних платформ. А саме методи оптимізації моделі RVC (зокрема, конвертація у формат ONNX та NCNN) та алгоритми інтеграції в аудіопайплайн мобільного застосунку на платформі Android.

Методи дослідження. Для досягнення поставленої мети та вирішення завдань кваліфікаційної роботи використано комплекс загальнонаукових та спеціальних методів:

1. методи аналізу та синтезу – для дослідження сучасного стану технологій голосової конверсії, порівняння наявних архітектур нейронних мереж та вибору оптимального стеку технологій;

2. системний аналіз та моделювання – для розробки загальної архітектури інформаційної системи, проєктування схеми взаємодії компонентів (HuBERT, RMVPE, Генератор) та алгоритмів потокової обробки даних;

3. об'єктноорієнтоване проєктування – для розробки структури програмного забезпечення, класів та інтерфейсів мобільного застосунку на

платформі Android;

4. експериментальний метод – для оцінки ефективності розробленої системи, вимірювання часових характеристик (затримок, показника RTF) та споживання системних ресурсів;

5. методи спектрального аналізу – для оцінки якості вихідного аудіосигналу та верифікації коректності роботи алгоритмів після квантизації.

Базою дослідження виступає документація PyTorch, NCNN, Android SDK. Інформаційним та технологічним підґрунтям дослідження виступили відкриті репозиторії програмного коду (GitHub) проєктів RVC-Project, NCNN та ONNX Runtime; технічна документація бібліотек машинного навчання PyTorch та засобів розробки Android SDK (Google Developers); офіційні специфікації графічного API Vulkan (The Khronos Group). У якості емпіричної бази використано набори попередньо навчених ваг моделей (Checkpoints), розміщених на платформі Hugging Face, та апаратне середовище сучасних мобільних пристроїв на базі архітектури ARM (SoC Qualcomm Snapdragon) для проведення натурних випробувань системи.

Наукова значущість одержаних результатів полягає у розробці комплексної методики адаптації архітектури RVC для локального інференсу на мобільних пристроях. На відміну від наявних десктопних та хмарних рішень, у роботі запропоновано та реалізовано пайплайн, що поєднує конвертацію PyTorch-моделей RVC в NCNN, HuBERT та RMVPE в ONNX та їх інтеграцію в аудіоконвеєр на Kotlin, що дозволяє досягти роботи системи в режимі, близькому до реального часу, на пристрої кінцевого користувача.

Практична значущість одержаних результатів полягає у створенні готового до використання програмного рішення – мобільного застосунку, який надає користувачам інструмент для швидкої та конфіденційної зміни голосу. Результати роботи можуть бути впроваджені в індустрії розваг, а також слугувати інструментом для творців контенту та акторів озвучування.

РОЗДІЛ 1. АНАЛІЗ ТЕХНОЛОГІЙ КОНВЕРСІЇ ГОЛОСУ ТА ПРОБЛЕМАТИКА ЇХ ІНТЕГРАЦІЇ НА МОБІЛЬНИХ ПЛАТФОРМАХ

1.1. Огляд наявних підходів до задачі клонування голосу та їх класифікація

Задачі, пов’язані з генерацією та модифікацією людського голосу, є одними з ключових напрямків у сучасному машинному навчанні. Клонування голосу, або ж конверсія голосу (Voice Conversion, VC), є специфічною підзадачею, метою якої є перетворення аудіосигналу однієї особи (джерела) таким чином, щоб він звучав, ніби його вимовила інша особа (ціль). При цьому, на відміну від простого синтезу мовлення (Text-to-Speech, TTS), ключовим завданням є збереження лінгвістичного змісту, а також емоційного забарвлення та інтонації (просодії) мовця-джерела.

Історично, підходи до VC можна класифікувати за двома основними критеріями: вимога до даних та архітектура моделі.

Розглянемо класифікацію стосовно вимог до даних [32], зокрема у контексті змагань Voice Conversion Challenge [23].

Підходи паралельної конверсії голосу (Parallel VC) вимагали наявності паралельних наборів даних, це означає, що для навчання моделі потрібно було мати аудіозаписи, де і мовець-джерело, і мовець-ціль начитують абсолютно однакові речення.

Найбільш поширеним методом у цій категорії був підхід на основі Гаусових Сумішей (Gaussian Mixture Models, GMM). Модель GMM вивчала статистичне відображення (мапінг) між акустичними ознаками (наприклад, кепстральними коефіцієнтами, MFCC) джерела та цілі.

В цьому випадку до недоліків відносять збір паралельних даних, який є вкрай трудомістким, дорогим і часто неможливим у реальних умовах, та якість згенерованого аудіо, яка часто була низькою, з роботизованим відтінком та втратою природності.

Справжній прорив у галузі відбувся з появою методів непаралельної

конверсії голосу (Non-Parallel VC), здатних навчатися на непаралельних даних – тобто, маючи просто набір аудіозаписів мовця А та окремий набір записів мовця Б, без жодного спільного речення. Це стало можливим завдяки глибокому навчанню та ідеї роз'єднання ознак (feature disentanglement).

Ця ідея полягає в тому, щоб навчити модель автоматично розділяти аудіосигнал на його складові:

1) лінгвістичний зміст (що сказано);
2) ідентичність мовця (хто говорить);
3) просодію (як говориться: інтонація, ритм).

Після такого розділення, модель може взяти зміст від джерела, ідентичність від цілі та згенерувати новий аудіосигнал.

Сучасні непаралельні методи конверсії голосу можна розділити на кілька архітектурних родин.

Однією з них є автокодувальники (AE/VAE), прикладами яких є моделі AutoVC [30] та VQ-VAE. Принцип їх роботи полягає у стисканні вхідного аудіосигналу в компактний латентний простір за допомогою кодувальника з подальшим відновленням через декодувальник. Ключовою особливістю є примусове «забування» інформації про ідентичність мовця у латентному просторі при збереженні змісту, де для конверсії декодер отримує код змісту та вектор цільового мовця. Цей підхід вирізняється чіткою архітектурою та добрим контролем над роз'єднанням ознак.

Альтернативним напрямком є використання генеративно-змагальних мереж (GANs), зокрема моделей StarGAN-VC [16] або CycleGAN-VC [17]. Вони функціонують на основі змагання двох мереж: генератора, який намагається виконати конверсію, та дискримінатора, що намагається відрізнити синтезований голос від справжнього. Хоча GANs здатні генерувати дуже чіткий та високоякісний звук, вони залишаються складними та нестабільними у навчанні, що вимагає використання специфічних методів адаптивної оптимізації, таких як Adam [20] або AdamW [24], та ретельного налаштування гіперпараметрів для уникнення колапсу моди.

Новітнім підходом у цій галузі є дифузійні моделі (наприклад, DiffVC) [22]. Процес генерації в них відбувається шляхом поступового «очищення» (denoising) сигналу зі звичайного шуму під керуванням векторів змісту та ідентичності. Цей метод забезпечує найвищу якість синтезу на сьогодні, однак має суттєвий недолік – дуже повільний інференс, який вимагає сотень ітеративних кроків. Це робить дифузійні моделі фактично непридатними для завдань реального часу на мобільних пристроях.

Окремий клас становлять гібридні підходи та методи на основі пошуку (Retrieval-based), яскравим представником яких є архітектура RVC [29]. Цей підхід інтегрує переваги генеративних моделей та механізмів пошуку релевантних зразків. Основу системи становить високопродуктивний генератор, що синтезує мовлення на базі семантичних токенів, тоді як модуль пошуку слугує допоміжним елементом для підвищення деталізації тембру шляхом залучення ознак із бази даних цільового мовця. Перевагами методу є модульність, висока природність синтезу завдяки сучасним вокодерам та ефективність навчання на обмежених даних. Водночас використання модуля пошуку створює високе навантаження на оперативну пам’ять та обчислювальні ресурси через необхідність обробки великих індексних файлів, що ускладнює його використання на мобільних пристроях у базовій конфігурації.

Варто зазначити, що ранні авторегресійні моделі, такі як WaveNet [37] та потокові моделі типу WaveGlow [29], забезпечували високу якість, але були надто повільними. Сучасні генератори використовують принципи GAN [8] для прискорення синтезу.

1.2. Аналіз обмежень мобільних платформ при роботі з нейромережами та огляд методів оптимізації

Перенесення сучасного пайплайну конверсії голосу, що складається з кількох моделей глибокого навчання – зокрема, самого генератора RVC, семантичного енкодера HuBERT та детектора висоти тону RMVPE – з серверного середовища на мобільні пристрої є нетривіальною задачею. Це

пов’язано з низкою фундаментальних апаратних та програмних обмежень, властивих мобільним платформам, таким як Android.

Розглянемо обмеження мобільних платформ.

Мобільні процесори (CPU) на архітектурі ARM, хоч і стали значно потужнішими, все ще суттєво поступаються десктопним та серверним CPU (x86) і особливо графічним процесорам (GPU), що використовуються для навчання. Хоча багато смартфонів мають власні мобільні GPU (наприклад, Adreno, Mali) та спеціалізовані NPU (Neural Processing Units), їхня продуктивність та програмна підтримка дуже фрагментовані [10].

Моделі для клонування голосу можуть вимагати завантаження ваг об’ємом у сотні мегабайт, а іноді й гігабайти. На мобільному пристрої обсяг RAM (наприклад, 4-8 ГБ) є спільним ресурсом між операційною системою, фоновими службами та іншими програмами. Перевищення ліміту пам’яті призводить до негайного крашу застосунку.

Інтенсивні та тривалі обчислення, необхідні для роботи нейромережі, є вкрай енергозатратними. Це призводить до швидкого розряду батареї та, що більш критично, до сильного нагріву пристрою. При досягненні певної температури система активує механізм теплового тротлінгу (thermal throttling), примусово знижуючи тактову частоту процесора. Це призводить до катастрофічного падіння продуктивності та заїкання звуку.

Для задачі трансформації голосу критичною є затримка. Користувач очікує почути змінений голос одразу після того, як закінчив говорити. Це означає, що час, витрачений на обробку 1 секунди аудіо, має бути меншим за 1 секунду. Ця метрика (RTF < 1.0) є головним критерієм успішності для систем реального часу.

Розглянемо методи оптимізації. Для подолання вищезазначених обмежень у галузі машинного навчання на пристроях (On-Device AI) було розроблено низку стандартних методів оптимізації моделей.

1. Квантизація (Quantization) – це найпоширеніший та найефективніший метод [14]. Він полягає у зниженні бітової точності, яка використовується для зберігання ваг моделі. FP32 (32-bit floating point) – Стандартна точність, що

використовується при навчанні. FP16 (16-bit floating point) – зменшує розмір моделі вдвічі та значно прискорює обчислення на сумісних GPU/NPU, практично без втрати якості. INT8 (8-bit integer) – Зменшує розмір моделі в 4 рази. Це дає найбільший приріст швидкості на мобільних CPU та NPU, але може призвести до деградації якості. Вимагає спеціального етапу калібрування.

2. Проріджування (Pruning) – це метод, який полягає у видаленні з моделі нейронних зв'язків (ваг), які є близькими до нуля і найменше впливають на кінцевий результат. Це дозволяє зменшити обчислювальну складність та розмір моделі, створюючи розріджені матриці ваг (як, наприклад, у MobileNets [12]).

3. Дистиляція знань (Knowledge Distillation) – це підхід, який передбачає навчання меншої, студентської моделі, яка намагається імітувати поведінку та результати великої, вчительської моделі. Це дозволяє створити компактну модель, оптимізовану під конкретну задачу, однак вимагає перенавчання.

4. Навіть оптимізована модель не може бути ефективно запущена на мобільному пристрої без спеціального середовища виконання (runtime). Використання повноцінних бібліотек навчання, як-от PyTorch [28] чи TensorFlow, на мобільних платформах є недоцільним через їхній значний розмір та орієнтацію на серверну інфраструктуру. Замість них застосовуються високооптимізовані фреймворки, призначені виключно для інференсу.

Перелічимо найбільш поширені та технологічно досконалі рішення.

1. TensorFlow Lite [1] (TFLite) – це спеціалізований формат для моделей екосистеми Google, оптимізований для мобільних ОС.

2. PyTorch Mobile [7] являє собою інструментарій для запуску моделей PyTorch на мобільних пристроях, що дозволяє виконувати інференс без складної конвертації, однак часто поступається у швидкодії іншим рішенням.

3. ONNX Runtime [26] (ORT) – це універсальний кросплатформний рушій від Microsoft. Забезпечує високу сумісність та продуктивність для моделей, конвертованих у відкритий формат ONNX.

4. NCNN [34] є високопродуктивним фреймворком, розробленим Tencent, що оптимізований спеціально для мобільних обчислень. Його

ключовими перевагами є мінімальний розмір бібліотеки, відсутність сторонніх залежностей та нативна підтримка графічного API Vulkan [35], що дозволяє ефективно залучити мобільні GPU.

Аналіз цих методів показує, що для успішної реалізації системи необхідно не лише обрати ефективну архітектуру моделі, але й застосувати комбінацію методів оптимізації та відповідних фреймворків інференсу для різних типів обчислювальних завдань.

1.3. Обґрунтування вибору архітектури RVC та засобів кросплатформної розробки

Вибір технологічного стека для даної кваліфікаційної роботи ґрунтується на аналізі, проведеному в попередніх підпунктах. Необхідно було знайти компроміс між високою якістю генерації голосу (п. 1.1) та жорсткими апаратними обмеженнями мобільних платформ (п. 1.2).

Обґрунтуємо вибір та модифікацію архітектури RVC. На основі проведеного огляду методів конверсії голосу як базову архітектуру було обрано генеративну основу моделі RVC (Retrieval-based Voice Conversion). Цей вибір обумовлений насамперед високою якістю самого генератора (вокодера), який є передовим (state-of-the-art) рішенням, здатним відтворювати високоякісний звук, а також гнучкою компонентною структурою архітектури, що дозволяє незалежно модифікувати енкодер, детектор висоти тону та генератор.

Водночас класична реалізація RVC передбачає використання пошукового (retrieval) модуля з індексом Faiss для «підмішування» векторів цільового голосу [15]. При аналізі можливості перенесення системи на мобільні платформи цей компонент було ідентифіковано як фундаментальне вузьке місце. Це пояснюється тим, що файли індексу можуть займати сотні мегабайт оперативної пам’яті, що є неприйнятним для обмежених ресурсів смартфона, а алгоритм векторного пошуку створює значне навантаження на центральний процесор, перешкоджаючи обробці в реальному часі.

У зв’язку з цим, у даній роботі прийнято інженерне рішення відмовитись

від використання компонента retrieval та застосовувати генератор RVC у режимі чистого прямого поширення (feed-forward), де модель напряму використовує ознаки з енкодера та висоту тону. Такий підхід дозволяє кардинально знизити вимоги до апаратних ресурсів та уможливлює запуск на Android, перетворивши весь процес на послідовність виключно нейромережевих операцій. Таким чином, для реалізації обрано оптимізований пайплайн, що складається з екстрактора семантичних ознак HuBERT для видобування змісту, детектора висоти тону RMVPE для фіксації інтонації та власне генератора RVC без модуля пошуку для синтезу кінцевого аудіосигналу.

Обґрунтуємо вибір інструментарію реалізації (Android, ONNX, NCNN). Вирішення проблеми запуску оптимізованого пайплайну з трьох моделей на мобільній платформі є ключовим інженерним завданням роботи.

В якості платформи обрано Android, як домінуючу мобільну ОС. Розробка ведеться на мові Kotlin, що є інструментом для створення сучасних застосунків.

Усі моделі (HuBERT, RMVPE, RVC-генератор) розроблені на Python/PyTorch. Окрім проблеми сумісності, існує проблема нерівномірного навантаження: генератор RVC є найбільш обчислювально-важким компонентом, який може перевантажити CPU.

Для досягнення максимальної продуктивності (RTF) та мінімізації затримок у роботі системи було застосовано гібридний підхід, який передбачає розподіл обчислювального навантаження між центральним (CPU) та графічним (GPU) процесорами мобільного пристрою. Зокрема, моделі екстракції ознак (HuBERT) та висоти тону (RMVPE) конвертуються у формат ONNX, а їх запуск здійснюється через середовище ONNX Runtime (ORT) на центральному процесорі. Такий вибір обґрунтований стабільністю роботи трансформерів на CPU та простотою інтеграції даного інструментарію з мовою Kotlin.

Своєю чергою, найбільш ресурсомісткий компонент системи, генератор RVC, було конвертовано у формат NCNN. Це високопродуктивний фреймворк, оптимізований спеціально для мобільних платформ, ключовою перевагою якого є нативна підтримка Vulkan API, що дозволяє виконувати обчислення

безпосередньо на мобільному графічному процесорі (GPU, наприклад, серії Adreno або Mali). Використання NCNN дозволяє реалізувати апаратне прискорення, що надає дві критичні переваги: розвантаження центрального процесора, який звільняється для обробки вхідного аудіобуфера та роботи моделей аналізу, а також суттєве покращення показника Real-Time Factor завдяки ефективності графічних процесорів у паралельних обчисленнях згорткових шарів. Таким чином, розроблена система функціонує під управлінням уніфікованого Kotlin-інтерфейсу, який оркеструє роботу двох різних рушіїв (ONNX Runtime та NCNN) для досягнення максимальної ефективності на мобільному апаратному забезпеченні.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ЛОКАЛЬНОЇ ТРАНСФОРМАЦІЇ АУДІОПОТОКУ

2.1. Архітектура та алгоритми функціонування моделі RVC

Проєктована інформаційна система базується на архітектурі, відомій як RVC (Retrieval-based Voice Conversion), яка, по суті, є комбінованим програмним конвеєром (pipeline), що інтегрує декілька state-of-the-art моделей, розроблених у суміжних галузях обробки мовлення.

Ключовим проєктним рішенням, прийнятим на основі аналізу обмежень (п. 1.2), є відмова від retrieval-компонента. Класичний RVC використовує пошук по індексу Faiss для інтерполяції векторів цільового голосу, що є неприйнятним для мобільних пристроїв через надмірні вимоги до RAM (для .index файлу) та CPU (для k-NN пошуку).

Натомість в даній роботі проєктована система використовує чистий feed-forward (прямого поширення) пайплайн, що складається з трьох незалежних нейромережевих модулів. Це перетворює систему на детермінований, хоч і обчислювально-інтенсивний, потік даних, придатний для оптимізації та запуску на мобільному пристрої.

Розглянемо Екстрактор семантичних ознак HuBERT. Модель призначена для виділення лінгвістичного змісту з аудіосигналу, дистилюючи його від акустичних характеристик мовця (тембру, просодії).

HuBERT (Hidden-Unit BERT) [13] – це модель, заснована на архітектурі Transformer [38] (зокрема, його кодувальній частині), що використовує самоконтрольоване навчання (self-supervised learning). Її навчання відбувається на великих нерозмічених аудіокорпусах. Архітектура HuBERT є розвитком ідей Contrastive Predictive Coding (CPC) [27]. Для стабілізації глибокої мережі всередині блоків трансформера використовується нормалізація шарів [3].

На першому етапі навчання, аудіосигнали перетворюються на спектрограми, а потім акустичні кадри кластеризуються (наприклад, за допомогою K-means). Кожен кластер відповідає певній акустичній одиниці

(псевдофонемі). Це формує прихований (hidden-unit) простір цільових токенів.

Під час основного навчання (рис. 2.1) модель функціонує за аналогією до BERT (Masked Language Modeling), але для аудіо (Masked Audio Modeling):

– на вхід подається послідовність аудіокадрів;

– випадкова частина кадрів маскується (замінюється);

– функція втрат моделі полягає у коректному передбаченні ідентифікатора кластера (акустичної одиниці) для маскованих ділянок на основі навколишнього контексту.

Для мінімізації функції втрат модель змушена вивчати складні залежності та фонетичну структуру мови, ігноруючи при цьому характеристики мовця, оскільки вони не є релевантними для передбачення лінгвістичного змісту.

На виході модель генерує тензор семантичних ознак (content vectors) – багате, стиснене представлення лінгвістичного змісту, відокремлене від ідентичності мовця.

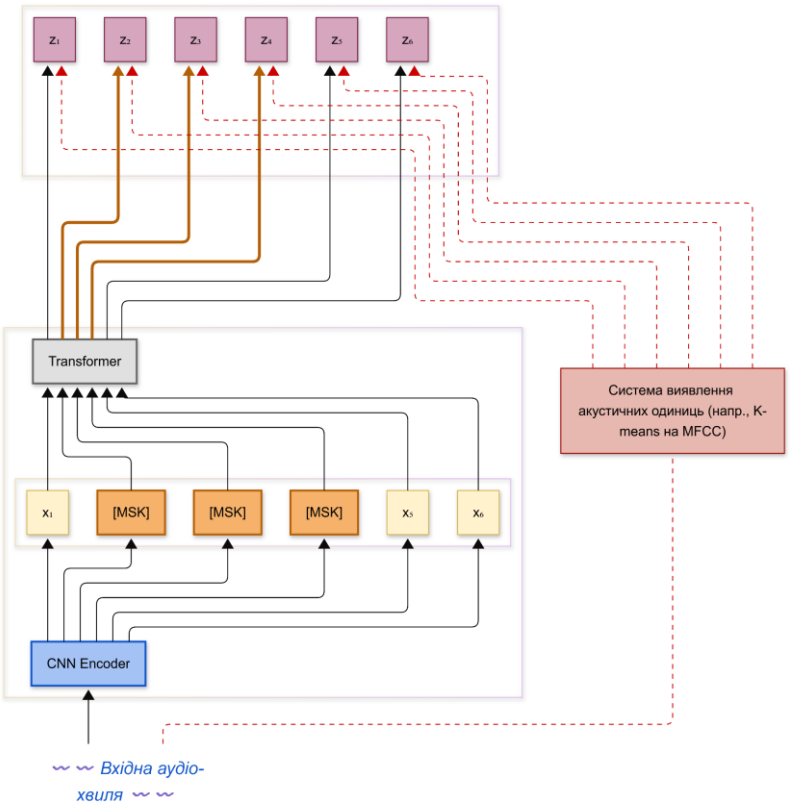


Рис. 2.1 Архітектура моделі HuBERT та процес передбачення маскованих кластерів

Далі розглянемо детектор висоти тону RMVPE. Його призначення у вилученні (оцінці) висоти основного тону (F0) для фіксації просодичних характеристик сигналу (інтонації).

RMVPE (Robust Median-based Pitch Estimation) [39] – це нейромережевий підхід до оцінки F0. Традиційні алгоритми (наприклад, YIN) схильні до помилок на октаву, особливо в присутності шумів або при аналізі вокалу, плутаючи основний тон з його гармоніками.

RMVPE являє собою модель, що базується на згорткових нейронних мережах (CNN), аналізує вхідну спектрограму і генерує для кожного часового кадру розподіл ймовірностей по всьому діапазону можливих частот F0 (рис. 2.2). Ефективність згорткових мереж для моделювання послідовностей у задачах обробки аудіо була підтверджена емпіричними дослідженнями [4].

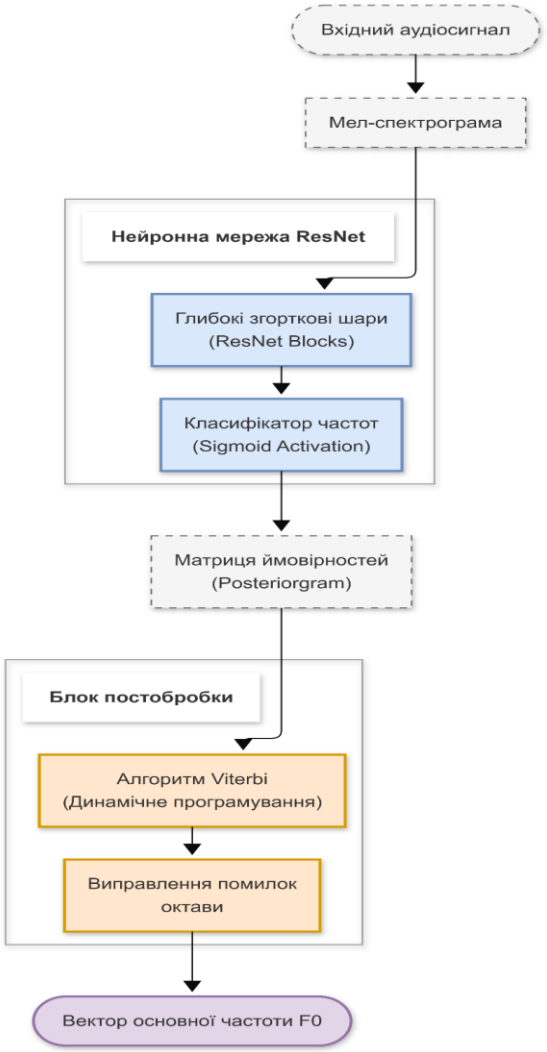


Рис.2.2 Схема нейромережевого детектора висоти тону RMVPE

Застосування *медіанного* усереднення до отриманих розподілів ймовірностей дозволяє RMVPE досягати високої стійкості (Robust) до шумів та уникати октавних помилок, характерних для інших методів.

На виході модель генерує 1D-вектор значень висоти основного тону, що точно описує інтонаційний контур вхідного сигналу.

На останок розглянемо Синтезатор (Генератор RVC).

Його призначення в синтезі кінцевої аудіохвилі шляхом декодування семантичних ознак (від HuBERT) та вектора висоти тону (від RMVPE) з накладанням тембральних характеристик цільового мовця.

Архітектурою є генератор RVC, який базується на модифікованій архітектурі VITS (Variational Inference with adversarial learning for end-to-end Speech Synthesis) [19] з інтеграцією модуля NSF (Neural Source Filter). На відміну від класичного VITS для синтезу мовлення з тексту, RVC не потребує передбачення тривалості фонем, оскільки часова структура (ритм) запозичується безпосередньо з вхідного аудіосигналу.

Перелічимо основні компоненти генератора (рис. 2.3) (відповідно до програмної реалізації SynthesizerTrnMs768NSFsid).

1. Content Encoder (Кодувальник змісту) приймає вектори ознак HuBERT (розмірністю 768) та перетворює їх у внутрішнє представлення. Оскільки RVC працює за принципом кадр-у-кадр, цей модуль зберігає часову вирівняність вхідного сигналу.

2. Normalizing Flows (Нормалізуючі потоки) відповідають за перетворення простого розподілу ознак (отриманого від енкодера) у складний розподіл, необхідний для декодера. Цей модуль дозволяє моделі маніпулювати стилем та варіативністю мовлення, покращуючи природність звучання.

3. Neural Source Filter (NSF, Нейронний фільтр джерела) – ключовий компонент архітектури RVC. Замість того щоб покладатися виключно на здатність нейромережі генерувати періодичні сигнали, NSF генерує синусоїдальне збудження (sine-wave excitation) на основі точного значення F0 (висоти тону). Це направляє генератор, забезпечуючи стабільність голосу та

відсутність металічних артефактів, особливо при співі.

4. HiFi-GAN Decoder (Декодер) [21] виконує роль вокодера. Він приймає дані від нормалізуючих потоків та сигнал збудження від NSF. Використовуючи серію блоків транспонованої згортки (Transposed Convolutions) та залишкових блоків (ResBlocks) [11], декодер виконує апсемплінг (upsampling) цих даних до частоти дискретизації цільового аудіо (наприклад, 40 кГц або 48 кГц), формуючи фінальну хвильову форму. В якості функції активації використовується ELU [5], що забезпечує швидшу збіжність. Для покращення відтворення тонкої акустичної структури використовуються методи нормалізації Instance Normalization [36] та Group Normalization [40], а функція втрат включає багатомасштабне передбачення (Multi-scale prediction) [25].

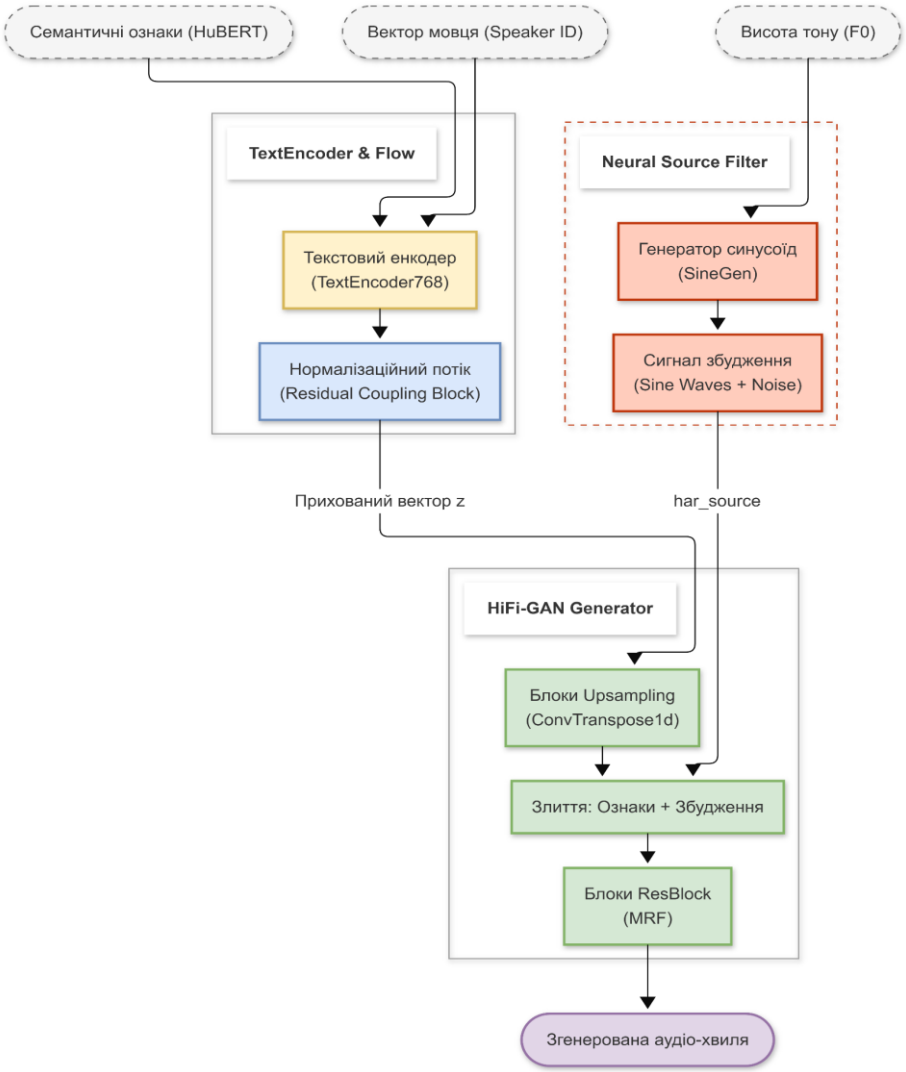


Рис. 2.3 Структурна схема генератора RVC

Для кожного вхідного аудіоблока (Audio_In), процес обчислень виглядає наступним чином:

1. обчислюється вектор висоти тону: Pitch_Vector = RMVPE_Module(Audio_In);

2. обчислюється тензор контенту: Content_Tensor = HuBERT_Module(Audio_In);

3. генерується вихідний сигнал: Audio_Out = RVC_Generator_Module(Content_Tensor, Pitch_Vector).

Таким чином, проєктована система є конвеєром з трьох обчислювально-інтенсивних нейронних мереж. Це створює значне навантаження, але відмова від retrieval-компонента робить задачу перенесення на мобільні пристрої принципово можливою. Наступний крок проєктування – розробка методики конвертації цих моделей у формат, оптимізований для мобільної платформи.

2.2. Розробка методики конвертації та оптимізації моделей для середовищ ONNX Runtime та NCNN

Оскільки оригінальні моделі пайплайну RVC (HuBERT, RMVPE, Генератор) розроблені на фреймворку PyTorch, вони не можуть бути безпосередньо інтегровані в Android-застосунок. Щобільше, прямий запуск важковагових моделей на мобільному пристрої без попередньої оптимізації призвів би до критичних затримок.

Для вирішення цієї проблеми було розроблено диференційовану методику конвертації, яка враховує специфіку кожного компонента системи та цільового апаратного забезпечення: центрального процесора (CPU) для аналізу та графічного процесора (GPU) для синтезу.

Для моделей екстракції ознак (HuBERT) та висоти тону (RMVPE), що виконуються на CPU через ONNX Runtime, було обрано стратегію використання спеціалізованого мобільного формату .ort. Цей формат дозволяє мінімізувати розмір бінарного файлу та прискорити ініціалізацію моделі.

Опишемо алгоритм конвертації та оптимізації.

1. Виконується експорт моделей з PyTorch з визначенням динамічних осей (Dynamic Axes), що дозволяє обробляти вхідні аудіобуфери змінної довжини без перекомпіляції графа.

2. Замість використання сторонніх спрощувачів, застосовано вбудовані засоби оптимізації ONNX Runtime. Процес включає конвертацію базового .onnx файлу у формат .ort з рівнем оптимізації ORT_ENABLE_ALL. Це охоплює: fusion (злиття) операторів; об'єднання послідовностей математичних операцій в одну інструкцію процесора; видалення невикористовуваних вузлів; очищення графа від операцій, що не впливають на вихідний результат (наприклад, Dropout шари); оптимізація пам'яті; перевпорядкування констант для кращої локальності кешу.

3. Отримано файли hubert.with_runtime_opt.ort та rmvpe.with_runtime_opt.ort, які є бінарно оптимізованими для виконання на мобільному CPU.

Далі розглянемо методику конвертації генератора RVC у формат NCNN (через PNNX).Генератор RVC є найбільш обчислювально-містким компонентом системи. Для його виконання на мобільному GPU через Vulkan API було обрано фреймворк NCNN [34]. Для конвертації використано сучасний інструмент PNNX (PyTorch Neural Network Exchange).

Проте, пряма конвертація оригінального коду виявилася неможливою через несумісність низки операцій PyTorch з обмеженим набором інструкцій мобільного інференсу та проблемою чисельної стабільності. Процес підготовки складався з двох етапів.

На першому етапі проведена адаптація та рефакторинг вихідного коду (PyTorch). Перед експортом було проведено глибоку модифікацію архітектури моделі на рівні Python-коду.

А саме, під час тестування було виявлено, що стандартна реалізація torch.cumsum при роботі в режимі FP16 (на GPU) призводить до швидкого накопичення помилки округлення (accumulated error). Це викликало значні спотворення звукового сигналу (шум, тріск) на виході генератора. Тому було

розроблено та імплементовано кастомну функцію розрахунку, яка забезпечує чисельну стабільність операції на мобільному графічному процесорі, замінивши стандартний виклик бібліотеки.

Оригінальна модель VITS використовує генератори випадкових чисел (random noise extraction) всередині графа під час інференсу. Ці операції погано підтримуються трансляторами в статичні графи. У зв'язку з цим стохастичні вузли було вилучено з графа моделі, а необхідний шум замінено на детерміновані операції.

Складні операції індексації (advanced slicing) та логічні порівняння над тензорами (оператори >, <, ==, !=, %), що використовувалися для динамічного керування потоком даних, викликали помилки при трасуванні графа PNNX. Через це проблемні ділянки коду було переписано з використанням матричних операцій, які мають прямі еквіваленти в наборі інструкцій NCNN/Vulkan.

На другому етапі, після модифікації коду, процес конвертації пройшов успішно за наступним алгоритмом.

1. Проведено експорт моделі у .pt файл за допомогою torch.jit.trace.
2. Використано утиліти pnnx для генерації файлів .param та .bin. Перевага PNNX полягає у збереженні високорівневих операторів без їх розбиття на безліч дрібних неефективних вузлів.
3. Активовано режим 16-бітної точності. Сучасні мобільні GPU (Adreno, Mali) мають апаратну підтримку FP16, що дозволяє прискорити обчислення генератора майже вдвічі порівняно з FP32.

Після конвертації проводиться перевірка коректності роботи оптимізованих моделей шляхом порівняння їх виходів з еталонними результатами PyTorch-моделей на тестових даних.

Для HuBERT/RMVPE використовується метрика косинусної схожості (Cosine Similarity) вихідних тензорів. Значення схожості >0.95 свідчить про збереження семантичної точності.

Для Генератора RVC проводиться спектральний аналіз згенерованого аудіо (порівняння мел-спектрограм) та суб'єктивний слуховий контроль на

відсутність артефактів, викликаних переходом на FP16.

Завдяки застосуванню PNNX, ORT-оптимізації та ручному рефакторингу коду вдалося отримати стабільний гібридний пайплайн, готовий до інтеграції в Android-застосунок.

2.3. Розробка алгоритмічного забезпечення та схеми потокової обробки даних

Для забезпечення стабільної трансформації голосу в режимі реального часу необхідно спроєктувати архітектуру, яка ефективно вирішує проблему синхронізації вхідного та вихідного аудіопотоків. Оскільки швидкість надходження даних з мікрофона є постійною (ізохронною), а час обробки нейромережами може варіюватися, пряма послідовна обробка є неможливою.

В основу архітектури покладено патерн проєктування Виробник-Споживач, який дозволяє розв’язати процеси запису та обробки.

Система розділяється на три незалежні логічні блоки:

1) блок захоплення (Producer) що відповідає за безперервне зчитування семплів з апаратного входу, пріоритет якого – ніколи не блокуватися, щоб не втратити вхідні дані;

2) буфер обміну (Queue) – це структура даних типу FIFO (First-In-First-Out), яка слугує демпфером між записом та обробкою, яка компенсує пікові затримки при інференсі моделей;

3) блок обробки (Consumer) який виконує ресурсомісткі обчислення і вилучає дані з черги в міру готовності процесора.

Нейронні мережі архітектури RVC не здатні обробляти безперервний потік даних посемплово. Необхідна сегментація на кадр фіксованої довжини.

Визначено оптимальний розмір кадру обробки $N = 32000$ семплів. При частоті дискретизації $F_s = 16000$ Гц це відповідає часовому вікну $T = 2.0$ с.

Такий розмір обрано як компроміс між затримкою та якістю: моделі HuBERT (трансформер) потребують достатнього контексту для коректного визначення фонем. Коротші вікна призводять до зниження точності

розпізнавання контенту.

Вхідний сигнал надходить у форматі 16-бітних цілих чисел (PCM 16-bit, діапазон значень $[-32768; 32767]$). Нейронні мережі оперують числами з рухомою комою у діапазоні $[-1.0; 1.0]$.

Алгоритм нормалізації [33] для кожного семплу x_i:

$$x_{norm} = \frac{x_i}{32768.0}$$

При проєктуванні враховано особливість згорткових мереж (CNN), які вимагають, щоб розмір вхідного тензора був кратним певному кроку згортки. Розроблено алгоритм безпечної довжини (Safe Length Alignment). Якщо останній отриманий кадр менший за необхідний розмір, він автоматично доповнюється нулями (zero-padding) до найближчого валідного значення, щоб уникнути помилок розмірності тензорів під час інференсу.

Логіка обробки одного кадру даних являє собою послідовний алгоритмічний ланцюг (рис. 2.4). Оскільки результат роботи однієї моделі є вхідними даними для іншої, паралелізація можлива лише на рівні внутрішніх операцій моделей, а не на рівні блоків.

Алгоритм обробки кадру:

1. на вході нормалізований вектор X (розмір N);
2. вектор X подається на вхід моделі HuBERT. Результатом є тензор семантичних ознак F;
3. паралельно вектор X подається на вхід моделі RMVPE для отримання вектора основної частоти P та його інтерпольованої версії P_f;
4. тензори F, P та P_f передаються у генератор;
5. отриманий вектор Y (синтезоване аудіо) конвертується назад у цілочисельний формат PCM.

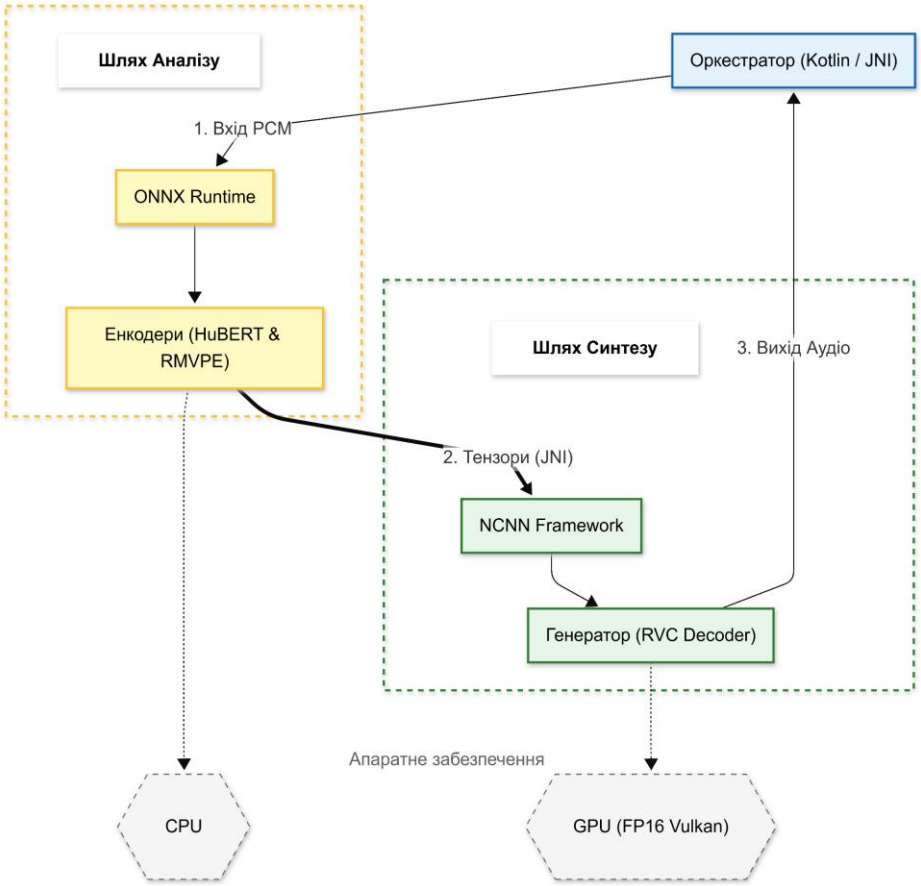


Рис. 2.4 Схема оркестрації нейромереж та гібридного розподілу обчислень

Для забезпечення сумісності з медіаплеєрами та соціальними мережами, система повинна включати блок кодування. Спроєктовано використання стандарту AAC (Advanced Audio Coding) у контейнері MPEG-4.

Алгоритм виведення передбачає автоматичне визначення частоти дискретизації вихідного сигналу, оскільки нейронний вокодер може змінювати частоту (наприклад, з 16 кГц на вході до 32 кГц або 40 кГц на виході) залежно від архітектури моделі.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ ТА ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА

3.1. Особливості реалізації аудіопайплайну та інтеграції нейромереж на платформі Android

Програмна реалізація системи виконана мовою Kotlin для компонентів бізнес-логіки та UI, з використанням мови C++ для інтеграції високопродуктивних модулів. Архітектура додатка побудована на принципах асинхронної обробки даних з використанням засобів Android SDK.

Для забезпечення спільної роботи CPU та GPU було реалізовано механізм ініціалізації двох незалежних рушіїв інференсу.

Робота з моделями HuBERT та RMVPE реалізована через бібліотеку ai.onnxruntime. В ході розробки було виявлено, що об’єкти OrtSession є чутливими до конкурентного доступу. Для забезпечення стабільності було створено виділений диспетчер потоків (Structured Concurrency [6]).

Усі операції завантаження моделей (ortEnvironment.createSession) та інференсу (session.run) виконуються виключно в цьому контексті, що гарантує послідовність виконання та запобігає конфліктам пам’яті.

Інтеграція NCNN реалізована на рівні Native-коду. Для конфігурації генератора використано структуру ncnn::Option, в якій програмно активовано підтримку Vulkan API.

Це дозволяє перенести обчислення згорткових шарів генератора на графічний процесор, що є критичним для досягнення цільових показників RTF.

Однією з ключових інженерних оптимізацій є механізм завантаження моделей NCNN (рис. 3.1). Замість стандартного підходу, що вимагає копіювання файлів моделей з APK у тимчасову папку файлової системи, реалізовано пряме читання з активів (Assets).

У нативному методі loadModelNative використано Android NDK API AssetManager [9]. Отриманий з Java об’єкт AssetManager передається у C++ шар, де методи g_net.load_param та g_net.load_model зчитують структуру графа та

бінарні ваги безпосередньо з пам’яті. Це суттєво зменшує час ініціалізації додатка та економить дисковий простір користувача.

Обмін даними під час інференсу реалізовано через JNI-метод runInferenceNative. Для мінімізації накладних витрат (overhead) використовуються вказівники на примітивні масиви (GetFloatArrayElements), що дозволяє NCNN отримувати доступ до даних, підготовлених у Kotlin, без зайвого глибокого копіювання.

Цикл обробки аудіо реалізовано за патерном Producer-Consumer з використанням блокувальної черги LinkedBlockingQueue.

Захоплення звуку (Producer) реалізовано в корутині Dispatchers.IO. Об’єкт AudioRecord налаштовано на запис у форматі 16-біт моно з частотою 16000 Гц. Сирі дані накопичуються у проміжний буфер до досягнення розміру чанку (32000 семплів), після чого додаються у чергу processingQueue.

Для моделей ONNX реалізовано алгоритм безпечної довжини (getNextSafeLength), який автоматично доповнює останній неповний чанк (padding) до розміру, кратного архітектурним вимогам моделі.

Споживач (Consumer) працює у контексті ortSessionDispatcher. Логіка обробки включає:

- вхідний масив ShortArray конвертується у FloatArray з діленням на $32768.0f$;
- дані послідовно проходять через сесії HuBERT та RMVPE;
- отримані ембедінги та пітч передаються у метод processChunkNCNN, який викликає нативний код;
- будь-який вхідний чанк, менший за еталонний розмір (32000 семплів), попередньо доповнюється нулями (zero-padding), оскільки використана реалізація NCNN-моделі оперує фіксованими розмірностями вхідних тензорів (статичний граф).

Відповідно, генератор завжди повертає масив повної довжини. Для відновлення коректної тривалості аудіо реалізовано логіку обрізки вихідного масиву, яка відсікає частину сигналу, що відповідає доданому паддінгу.

Для експорту обробленого аудіо реалізовано клас кодування на базі низькорівневого API MediaCodec [2].

Вхідний PCM-потік (отриманий після інференсу) передається у вхідний буфер кодека (рис. 3.2), налаштованого на формат AAC-LC (MIME type audio/mp4a-latm).

Використання MediaMuxer дозволяє здійснювати потоковий запис закодованих фреймів у контейнер .m4a без необхідності накопичення всього файлу в оперативній пам’яті (рис. 3.3). Особливістю реалізації є автоматичне визначення частоти дискретизації вихідного файлу на основі фактичного розміру вихідного буфера генератора, що дозволяє коректно зберігати аудіо навіть якщо модель змінює частоту дискретизації (resampling).

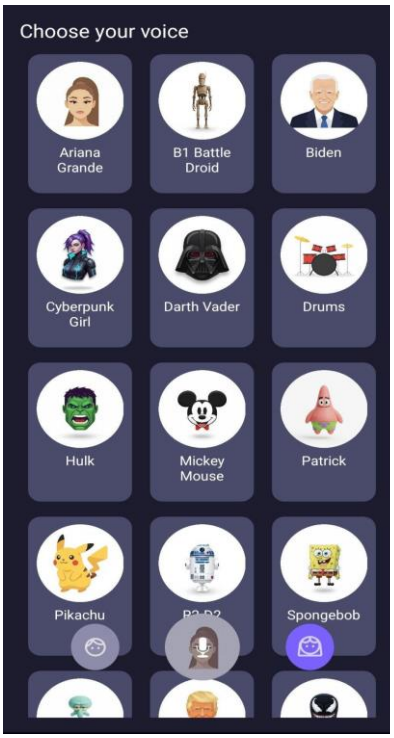


Рисунок 3.1 – Інтерфейс вибору голосової моделі та конфігурації параметрів інференсу

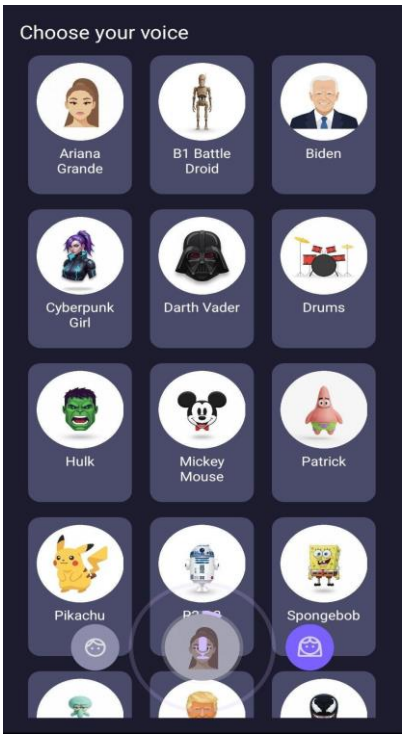


Рисунок 3.2 – Головний екран додатка: процес захоплення та потокової обробки аудіосигналу

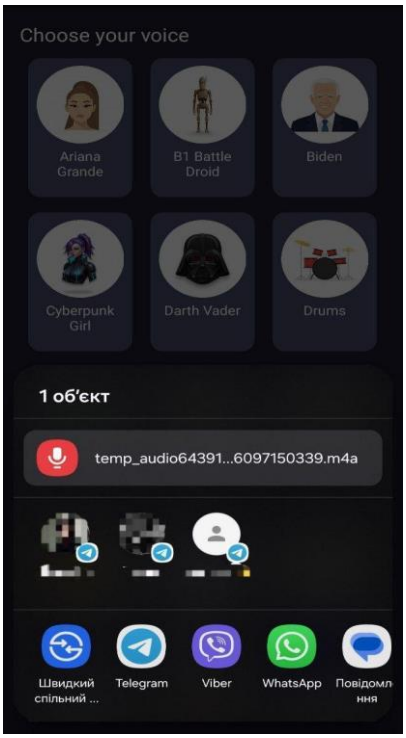


Рисунок 3.3 – Експорт результату: системний діалог (Share Sheet) для передачі згенерованого AAC-файлу в зовнішні додатки.

3.2. Аналіз результатів тестування продуктивності системи та якості вихідного сигналу

Метою експериментального дослідження була верифікація ефективності запропонованої гібридної архітектури (CPU/ONNX + GPU/NCNN) та перевірка відповідності розробленої системи вимогам роботи в режимі реального часу.

Експериментальні дослідження проводилися на фізичному мобільному пристрої, що дозволило оцінити реальні показники продуктивності апаратних прискорювачів (GPU), які неможливо емулювати у віртуальному середовищі.

Апаратна конфігурація тестового стенда:

- пристрій: Samsung Galaxy S24 Ultra;
- система на кристалі (SoC): Qualcomm Snapdragon 8 Gen 3;
- центральний процесор (CPU): 8-ядерна архітектура (Cortex-X4 + Cortex-A720 + Cortex-A520);
- графічний прискорювач (GPU): Adreno 750 (використовується для обчислень NCNN/Vulkan);
- оперативна пам’ять (RAM): 12 ГБ;
- операційна система: Android 14.

Параметри експерименту:

- частота дискретизації: 16000 Гц;
- розмір буфера обробки: 32000 семплів;
- тривалість аудіофрагмента (T_{audio}): 2.0 с.

Основним критерієм ефективності алгоритмів є коефіцієнт реального часу (Real-Time Factor, RTF). Для забезпечення безперервної роботи системи час обробки ($T_{processing}$) має бути меншим за тривалість вхідного аудіофрагмента (T_{audio}).

Формула розрахунку:

$$RTF = \frac{T_{processing}}{T_{audio}}$$

У ході експерименту проведено вимірювання часу виконання кожного етапу пайплайну. Результати усереднено за 100 циклів обробки (табл. 3.1).

Таблиця 3.1

Часові характеристики етапів обробки аудіосигналу

Етап обробки	**Середовище виконання**	**Середній час (мс)**
1. Попередня обробка	CPU (Kotlin)	1,11
2. Екстракція ознак (HuBERT)	CPU (ONNX Runtime)	398,89
3. Екстракція висоти тону (RMVPE)	CPU (ONNX Runtime)	187,28
4. Акустичний синтез (RVC Generator)	GPU (NCNN / Vulkan)	707,61
Загальний час ($T_{processing}$)	—	**1294,89**

Розрахунок ефективності:

- тривалість буфера (T_{audio}): 2000 мс;
- час обробки ($T_{processing}$): 1294,89 мс;
- RTF: 0.647445.

Отриманий показник RTF = 0.647445 свідчить про те, що система обробляє вхідний потік у 1.54 разів швидше за швидкість його відтворення. Розподіл навантаження між CPU та GPU дозволив уникнути вузьких місць у продуктивності.

Моніторинг споживання ресурсів проводився за допомогою засобу Android Profiler для оцінки ефективності використання оперативної пам’яті та навантаження на процесор (табл. 3.2).

Показники споживання оперативної пам’яті знаходяться в межах норми для мультимедійних застосунків. Оптимізація моделей (квантизація та FP16) дозволила уникнути надмірного використання RAM, що забезпечує стабільну

роботу додатка.

Таблиця 3.2

Показники використання ресурсів

Показник	Значення	Коментар
CPU Usage	51%	Середнє навантаження на обчислювальні ядра
GPU Usage	23%	Навантаження під час активної фази синтезу
RAM Usage	1113 МБ	Загальний обсяг пам’яті, виділений процесу

Для контролю якості синтезу після переходу на 16-бітні обчислення (FP16) на GPU було проведено порівняльний аналіз.

Порівняння спектрограм вихідного сигналу мобільної версії та еталонної PyTorch-версії продемонструвало збереження гармонічної структури та відсутність значних спектральних викривлень.

Слуховий контроль підтвердив розбірливість мовлення та коректність передачі інтонації. Артефакти, пов’язані з помилками округлення (наприклад, тріск), відсутні завдяки програмній корекції алгоритмів генерації.

3.3. Перспективи масштабування системи та сфери її практичного застосування

Розроблена інформаційна система є не лише завершеним програмним продуктом, але й базовою платформою для подальших досліджень у галузі мобільного нейроморфного обчислення. Архітектурні рішення, закладені в проєкт (гібридний інференс, модульність), дозволяють визначити чіткі вектори масштабування системи як у технічному, так і в прикладному аспектах.

Подальший розвиток системи передбачає поглиблену оптимізацію для підтримки ширшого спектра пристроїв та розширення функціоналу.

У поточній реалізації основне навантаження виконує GPU через Vulkan

API. Перспективним напрямком є перенесення обчислень на NPU (Neural Processing Unit) – спеціалізовані блоки в сучасних процесорах (наприклад, Qualcomm Hexagon). Це можливо через використання Android NNAPI або Qualcomm QNN SDK. Використання NPU дозволить знизити енергоспоживання у 3-5 разів порівняно з GPU, що критично важливо для мобільних пристроїв.

Генератор RVC наразі працює у точності FP16. Дослідження методів посттренувальної квантизації (Post-Training Quantization) до 8-бітних цілих чисел (INT8) дозволить запускати систему навіть на бюджетних смартфонах середнього цінового сегмента, зменшивши розмір моделей та вимоги до пропускної здатності пам’яті.

Важливим етапом розвитку є впровадження можливості локального донавчання. Використання методів адаптації низького рангу (LoRA) дозволить користувачеві створити клонінг власного голосу або голосу іншої особи безпосередньо на телефоні за кілька хвилин, без необхідності відправляти дані на сервер. Це забезпечить повну автономність екосистеми.

Для перетворення лабораторного прототипу на масовий продукт необхідно вирішити питання системної інтеграції:

- розробка механізму (наприклад, з використанням Android Accessibility Service або віртуального мікрофона), який дозволить перенаправляти оброблений аудіопотік безпосередньо в сторонні додатки (месенджери, Discord, ігри) в реальному часі;
- оскільки ядро системи (NCNN/ONNX) реалізовано на C++, воно може бути легко портоване на інші платформи, зокрема iOS (з використанням CoreML) та Desktop (Windows/Linux/macOS), зі збереженням єдиної кодової бази для логіки інференсу.

Потенціал розробленої технології виходить за межі розважальних застосунків. Можна виділити три ключові сфери впровадження.

1. Система є потужним інструментом для блогерів, стримерів, творців подкастів та незалежних розробників ігор (Indie GameDev). Вона дозволяє здійснювати професійне озвучення персонажів або дубляж відео однією

людиною, значно знижуючи витрати на залучення акторів озвучування.

2. В епоху цифрового стеження технологія може слугувати засобом біометричної анонімізації. На відміну від простих ефектів (зміна тональності), які легко реверсувати, нейронна трансформація створює абсолютно нову, реалістичну голосову особистість. Це може бути використано журналістами, активістами або свідками для захисту своєї ідентичності під час публічних виступів чи інтерв'ю.

3. Соціальна та медична сфера (Assistive Tech) є одним із найбільш соціально значущих векторів розвитку. Технологія може бути використана в галузі голосового банкінгу (Voice Banking). Пацієнти, які ризикують втратити голос через хворобу, можуть заздалегідь зберегти модель свого голосу. У поєднанні з системами синтезу мовлення (TTS), розроблений застосунок дозволить їм у майбутньому спілкуватися, використовуючи синтезований звук, який максимально наближений до їхнього рідного голосу, що має величезне психологічне значення для реабілітації та соціалізації.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну науково-прикладну задачу розробки системи локальної трансформації голосу в реальному часі на мобільних пристроях. На основі проведених досліджень та практичної реалізації отримано наступні результати.

Проведено аналіз сучасних методів нейронного синтезу мовлення та обмежень мобільних апаратних платформ. Встановлено, що архітектура RVC (Retrieval-based Voice Conversion) забезпечує високу якість клонування голосу, проте її стандартна реалізація є непридатною для мобільних пристроїв через високі вимоги до пам’яті та обчислювальної потужності.

Запропоновано та обґрунтовано модифіковану архітектуру системи, що працює в режимі чистого прямого поширення (feed-forward). Прийнято інженерне рішення відмовитися від модуля пошуку (retrieval) на користь оптимізації швидкодії, що дозволило усунути залежність від великих індексних файлів та критично знизити споживання оперативної пам’яті.

Розроблено гібридну схему інференсу, яка ефективно розподіляє навантаження між різними обчислювальними блоками системи на кристалі (SoC). Реалізовано виконання легковагових моделей аналізу (HuBERT, RMVPE) на центральному процесорі (CPU) через ONNX Runtime, а важковагового генератора – на графічному процесорі (GPU) через NCNN з підтримкою Vulkan.

Сформовано методику конвертації та адаптації моделей, що включає використання інструментарію PNNX, застосування 16-бітної точності (FP16) та глибокий рефакторинг вихідного коду PyTorch. Зокрема, розроблено кастомні реалізації математичних операцій (таких як кумулятивна сума) для забезпечення чисельної стабільності на мобільних GPU.

Програмно реалізовано автономний мобільний застосунок для ОС Android мовою Kotlin з інтеграцією нативних модулів на C++. Архітектура додатка базується на патерні Виробник-Споживач (Producer-Consumer) та використовує асинхронну чергу обробки, що забезпечує стабільність аудіопотоку. Реалізовано

механізм прямого читання моделей з активів (Assets) без копіювання, що оптимізує час запуску.

Експериментально підтверджено ефективність розробленої системи. Тестування на пристрої Samsung Galaxy S24 Ultra продемонструвало досягнення показника коефіцієнта реального часу (RTF) на рівні 0.647445, що свідчить про значний запас продуктивності. Суб’єктивна та об’єктивна оцінка якості вихідного сигналу підтвердила відсутність критичних артефактів та збереження природності звучання голосу.

Таким чином, розроблена система є програмним продуктом, що дозволяє виконувати високоякісну трансформацію голосу без підключення до інтернету, забезпечуючи конфіденційність даних користувача та мінімальні затримки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Abadi M., Agarwal A., Barham P. et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv preprint arXiv:1603.04467*. 2016.

2. Android Open Source Project. MediaCodec Class Overview. URL: https://developer.android.com/reference/android/media/MediaCodec (дата звернення: 18.02.2025).

3. Ba J. L., Kiros J. R., Hinton G. E. Layer Normalization. *arXiv preprint arXiv:1607.06450*. 2016.

4. Bai S., Kolter J. Z., Koltun V. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *arXiv preprint arXiv:1803.01271*. 2018.

5. Clevert D.-A., Unterthiner T., Hochreiter S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *International Conference on Learning Representations (ICLR)*. 2016.

6. Elizarov R., Belyaev M., Akhin M., Usmanov I. Kotlin Coroutines: Design and Implementation. *Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*. 2021. P. 68–84.

7. Facebook AI Research. PyTorch Mobile: Deploying PyTorch models on iOS and Android. URL: https://pytorch.org/mobile/home/ (дата звернення: 25.01.2025).

8. Goodfellow I., Pouget-Abadie J., Mirza M. et al. Generative Adversarial Nets. *Advances in Neural Information Processing Systems (NeurIPS)*. 2014. Vol. 27. P. 2672–2680.

9. Google Developers. Android NDK Guides: Audio Latency. URL: https://developer.android.com/ndk/guides/audio/audio-latency (дата звернення: 05.02.2025).

10. Google Developers. Optimize for Android (NPU/GPU). URL: https://developer.android.com/ (дата звернення: 10.02.2025).

11. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. P. 770–778.

12. Howard A. G., Zhu M., Chen B. et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*. 2017.

13. Hsu W.-N., Bolte B., Tsai Y.-H. H. et al. HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. 2021. Vol. 29. P. 3451–3460.

14. Jacob B., Kligys S., Chen B. et al. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018. P. 2704–2713.

15. Johnson J., Douze M., Jégou H. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*. 2021. Vol. 7, no. 3. P. 535–547.

16. Kameoka H., Kaneko T., Tanaka K., Hojo N. StarGAN-VC: Non-parallel many-to-many voice conversion using star generative adversarial networks. *IEEE Spoken Language Technology Workshop (SLT)*. 2018. P. 266–273.

17. Kaneko T., Kameoka H. CycleGAN-VC: Non-parallel Voice Conversion Using Cycle-Consistent Adversarial Networks. *2018 26th European Signal Processing Conference (EUSIPCO)*. 2018. P. 2100–2104.

18. Kim J. W., Salamon J., Li P., Bello J. P. CREPE: A Convolutional Representation for Pitch Estimation. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018. P. 161–165.

19. Kim J., Kong J., Son J. VITS: Conditional Variational Autoencoder with Adversarial Learning for End-to-End Text-to-Speech. *Proceedings of the 38th International Conference on Machine Learning (ICML)*. PMLR, 2021. Vol. 139. P. 5530–5540.

20. Kingma D. P., Ba J. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)*. 2015.

21. Kong J., Kim J., Bae J. HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis. *Advances in Neural Information Processing Systems (NeurIPS)*. 2020. Vol. 33. P. 17022–17033.

22. Kong Z., Ping W., Huang J. et al. DiffWave: A Versatile Diffusion Model for Audio Synthesis. *International Conference on Learning Representations (ICLR)*. 2021.

23. Lorenzo-Trueba J., Yamagishi J., Toda T. et al. The Voice Conversion Challenge 2018: Promoting Development of Parallel and Nonparallel Methods. *arXiv preprint arXiv:1804.04262*. 2018.

24. Loshchilov I., Hutter F. Decoupled Weight Decay Regularization. *International Conference on Learning Representations (ICLR)*. 2019.

25. Mathieu M., Couprie C., LeCun Y. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*. 2015.

26. Microsoft. ONNX Runtime: Cross-platform, high performance ML inference and training accelerator. URL: https://onnxruntime.ai (дата звернення: 12.01.2025).

27. Oord A. v. d., Li Y., Vinyals O. Representation Learning with Contrastive Predictive Coding. *arXiv preprint arXiv:1807.03748*. 2018.

28. Paszke A., Gross S., Massa F. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems (NeurIPS)*. 2019. Vol. 32. P. 8024–8035.

29. Prenger R., Valle R., Catanzaro B. WaveGlow: A Flow-based Generative Network for Speech Synthesis. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019. P. 3617–3621.

30. Qian K., Zhang Y., Chang S. et al. AutoVC: Zero-Shot Voice Style Transfer with Only Autoencoder Loss. *Proceedings of the 36th International Conference on Machine Learning (ICML)*. PMLR, 2019. P. 5210–5219.

31. RVC-Project: Retrieval-based Voice Conversion WebUI. GitHub Repository. URL: https://github.com/RVC-Project/Retrieval-based-Voice-Conversion-WebUI (дата звернення: 10.12.2024).

32. Sisman B., Yamagishi J., King S., Li H. An Overview of Voice Conversion and its Challenges: From Statistical Modeling to Deep Learning. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. 2021. Vol. 29. P. 132–157.

33. Smith S. W. *The Scientist and Engineer's Guide to Digital Signal Processing*. 2nd ed. San Diego: California Technical Publishing, 1999. 626 p.

34. Tencent. NCNN: High-performance neural network inference framework optimized for the mobile platform. GitHub Repository. URL: https://github.com/Tencent/ncnn (дата звернення: 15.01.2025).

35. The Khronos Group. Vulkan 1.3 Specification. URL: https://www.khronos.org/registry/vulkan/ (дата звернення: 20.01.2025).

36. Ulyanov D., Vedaldi A., Lempitsky V. Instance Normalization: The Missing Ingredient for Fast Stylization. *arXiv preprint arXiv:1607.08022*. 2016.

37. Van Den Oord A., Dieleman S., Zen H. et al. WaveNet: A Generative Model for Raw Audio. *arXiv preprint arXiv:1609.03499*. 2016.

38. Vaswani A., Shazeer N., Parmar N. et al. Attention Is All You Need. *Advances in Neural Information Processing Systems (NeurIPS)*. 2017. Vol. 30. P. 5998–6008.

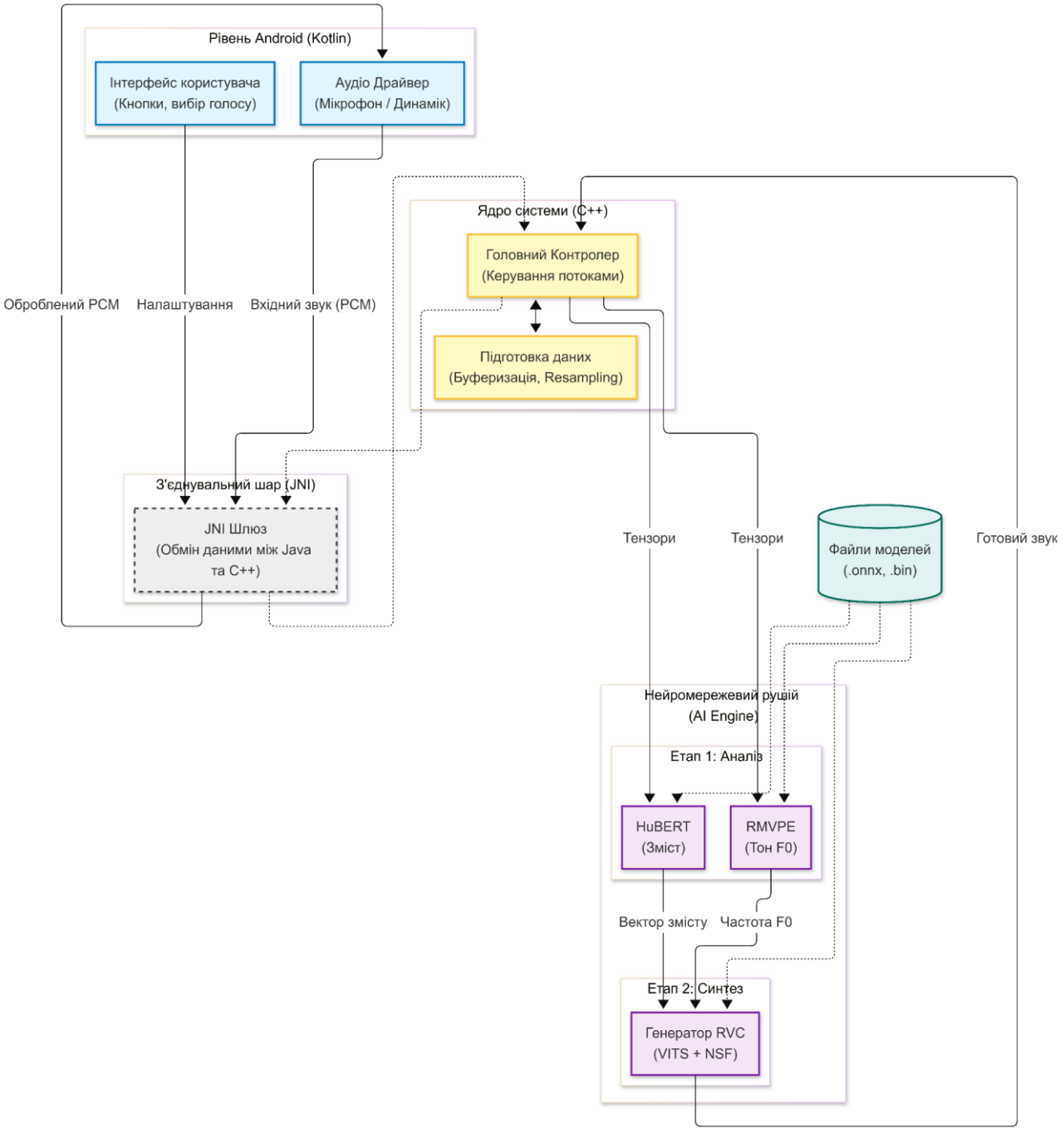
39. Wei H., Cao X., Dan T., Chen Y. RMVPE: A Robust Model for Vocal Pitch Estimation in Polyphonic Music. *arXiv preprint arXiv:2306.15412*. 2023.

40. Wu Y., He K. Group Normalization. *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018. P. 3–19.

ДОДАТКИ

Додаток А

СХЕМА ВЗАЄМОДІЇ КОМПОНЕНТІВ СИСТЕМИ



Додаток Б

ЛІСТИНГ ПРОГРАМИ

```
package com.example.voicetransformer

import android.Manifest
import android.animation.AnimatorSet
import android.animation.ObjectAnimator
import android.animation.PropertyValuesHolder
import android.content.Context
import android.content.Intent
import android.content.pm.PackageManager
import android.content.res.AssetManager
import android.media.AudioFormat
import android.media.AudioRecord
import android.media.MediaCodec
import android.media.MediaCodecInfo
import android.media.MediaFormat
import android.media.MediaPlayer
import android.media.MediaMuxer
import android.media.MediaRecorder
import android.os.Bundle
import android.util.Log
import android.util.TypedValue
import android.view.LayoutInflater
import android.view.MotionEvent
import android.view.View
import android.view.ViewGroup
import android.view.animation.AccelerateDecelerateInterpolator
import android.view.animation.LinearInterpolator
import android.widget.FrameLayout
import android.widget.ImageButton
import android.widget.ImageView
import android.widget.LinearLayout
import android.widget.ProgressBar
import android.widget.SeekBar
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import androidx.core.content.FileProvider
```

```
import androidx.core.content.res.ResourcesCompat
import androidx.recyclerview.widget.GridLayoutManager
import androidx.recyclerview.widget.RecyclerView
import ai.onnxruntime.OnnxTensor
import ai.onnxruntime.OrtEnvironment
import ai.onnxruntime.OrtLoggingLevel
import ai.onnxruntime.OrtSession
import com.google.android.material.floatingactionbutton.FloatingActionButton
import java.io.ByteArrayOutputStream
import java.io.File
import java.io.FileOutputStream
import java.io.IOException
import java.nio.ByteBuffer
import java.nio.ByteOrder
import java.util.concurrent.LinkedBlockingQueue
import kotlinx.coroutines.*
import android.graphics.Color
import android.content.res.ColorStateList
import android.os.CountDownTimer

class MainActivityRefactor : AppCompatActivity() {

    // UI Components
    private lateinit var loadingIndicator: ProgressBar
    private lateinit var recordButton: FloatingActionButton
    private lateinit var seekBar: SeekBar
    private lateinit var modelsRecyclerView: RecyclerView
    private lateinit var playerControls: LinearLayout
    private lateinit var playPauseButton: ImageButton
    private lateinit var shareButton: ImageButton
    private lateinit var currentTimeTextView: TextView
    private lateinit var totalTimeTextView: TextView
    private lateinit var maleButton: ImageButton
    private lateinit var femaleButton: ImageButton
    private lateinit var characterImageView: ImageView
    private lateinit var characterImageWrapper: FrameLayout
    private lateinit var recordingProgressBar: ProgressBar

    // Animation views
    private lateinit var micWrapper: FrameLayout
    private lateinit var waveCircle1: View
    private lateinit var waveCircle2: View
    private lateinit var waveCircle3: View
```

```
private lateinit var waveCircle4: View

// Animation objects
private var rippleAnimators: MutableList<AnimatorSet> = mutableListOf()
private var iconPulseAnimator: ObjectAnimator? = null

private enum class PitchState { NONE, MALE, FEMALE }
private var currentPitchState = PitchState.NONE

// Media Player
private var mediaPlayer: MediaPlayer? = null
private var audioDataStream: ByteArrayOutputStream? = null
private var currentAudioFile: File? = null

// F0 Up Key
private var f0UpKey: Int = 0
private var currentModelName: String? = null

// Coroutines
private val mainScope = CoroutineScope(Dispatchers.Main + SupervisorJob())
private lateinit var modelAdapter: ModelAdapter
private val ortSessionDispatcher = newSingleThreadContext("OrtSessionThread")
private var recordingTimerJob: Job? = null

// ONNX Models and Environment
private lateinit var ortEnvironment: OrtEnvironment
private lateinit var sessRmvpe: OrtSession
private lateinit var sessEmbed: OrtSession
private var sessInfer: OrtSession? = null

// NCNN Models and Inference
private var ncnnInference: NCNNInference? = null
private var isNcnnModelLoaded: Boolean = false

// Audio Recording
private var recorder: AudioRecord? = null
private var isRecording = false
private val inputSampleRate = 16000
private var outputSampleRate = 32000
private var isSampleRateCalculated = false
private val channelConfig = AudioFormat.CHANNEL_IN_MONO
private val audioFormat = AudioFormat.ENCODING_PCM_16BIT
private   val   bufferSize   =   AudioRecord.getMinBufferSize(inputSampleRate,
```

```
channelConfig, audioFormat)
            private val chunkSize = 32000
            private val processingQueue = LinkedBlockingQueue<ShortArray>()
            private var processingJob: Job? = null
            // Logging variables
            private var totalProcessingTimeMs: Long = 0L
            private var totalRecordedAudioSamples: Long = 0L

            private val MAX_RECORDING_DURATION_MS = 30000L

            // Performance logging variables for each stage
            private var totalPreprocessingTimeMs: Long = 0L
            private var totalHubertTimeMs: Long = 0L
            private var totalRmvpeTimeMs: Long = 0L
            private var totalRvcGeneratorTimeMs: Long = 0L
            private var totalPostprocessingTimeMs: Long = 0L
            private var processedChunkCount: Int = 0
            private var producerJob: Job? = null

            override fun onCreate(savedInstanceState: Bundle?) {
                super.onCreate(savedInstanceState)
                setContentView(R.layout.activity_main)

                setupViews()
                checkPermissions()

                // Initialize NCNN Inference and ONNX Environment
                ncnnInference = NCNNInference(this)
                ortEnvironment                                                    =
OrtEnvironment.getEnvironment(OrtLoggingLevel.ORT_LOGGING_LEVEL_VERBOSE)

                mainScope.launch(ortSessionDispatcher) {
                    loadBaseModels()
                }

                // Initialize NCNN with a dummy model for now, actual model loads on click
                ncnnInference?.initializeDummy()


                val modelsList = getModelsListFromAssets()
                modelAdapter = ModelAdapter(modelsList) { modelName, imageResId ->
                    loadSelectedModel(modelName)
                    updateCharacterImage(imageResId)
```

```
            }
            modelsRecyclerView.apply {
                val spanCount = calculateSpanCount()
                layoutManager     =     GridLayoutManager(this@MainActivityRefactor,
spanCount)
                adapter = modelAdapter
            }

            recordButton.setOnTouchListener { _, event ->
                when (event.action) {
                    MotionEvent.ACTION_DOWN -> {
                        startRecording()
                        startRecordingAnimation()
                    }
                    MotionEvent.ACTION_UP, MotionEvent.ACTION_CANCEL -> {
                        stopRecording()
                        stopRecordingAnimation()
                    }
                }
                true
            }
        }

        override fun onBackPressed() {
            if (mediaPlayer?.isPlaying == true) {
                mediaPlayer?.stop()
                mediaPlayer?.release()
                mediaPlayer = null
                playerControls.visibility = View.GONE
            }
            super.onBackPressed()
        }

        private fun setupViews() {
            loadingIndicator = findViewById(R.id.loadingIndicator)
            recordButton = findViewById(R.id.recordButton)
            seekBar = findViewById(R.id.seekBar)
            modelsRecyclerView = findViewById(R.id.modelsRecyclerView)
            playerControls = findViewById(R.id.playerControls)
            playPauseButton = findViewById(R.id.playPauseButton)
            currentTimeTextView = findViewById(R.id.currentTimeTextView)
            totalTimeTextView = findViewById(R.id.totalTimeTextView)
            characterImageView = findViewById(R.id.characterImageView)
```

```
            characterImageWrapper = findViewById(R.id.characterImageWrapper)
            recordingProgressBar = findViewById(R.id.recordingProgressBar)

            // Initialize animation views
            micWrapper = findViewById(R.id.micWrapper)
            waveCircle1 = findViewById(R.id.waveCircle1)
            waveCircle2 = findViewById(R.id.waveCircle2)
            waveCircle3 = findViewById(R.id.waveCircle3)
            waveCircle4 = findViewById(R.id.waveCircle4)

            recordButton.isEnabled = false
            playPauseButton.setOnClickListener { togglePlayback() }
            shareButton = findViewById(R.id.shareButton)
            shareButton.setOnClickListener { shareAudio() }

            maleButton = findViewById(R.id.maleButton)
            femaleButton = findViewById(R.id.femaleButton)

            maleButton.setOnClickListener {
                currentPitchState = if (currentPitchState == PitchState.MALE)
PitchState.NONE else PitchState.MALE
                updatePitch()
            }

            femaleButton.setOnClickListener {
                currentPitchState = if (currentPitchState == PitchState.FEMALE)
PitchState.NONE else PitchState.FEMALE
                updatePitch()
            }

            updatePitchButtonsUI()
            // Initial state: hide character image and set record button to default
color
            updateCharacterImage(0)
        }

        private fun startRecordingAnimation() {
            recordingProgressBar.visibility = View.VISIBLE
            recordingProgressBar.progress = 0

            // Icon pulse animation
            iconPulseAnimator = ObjectAnimator.ofPropertyValuesHolder(recordButton,
                PropertyValuesHolder.ofFloat(View.SCALE_X, 1.0f, 1.1f),
```

```
                PropertyValuesHolder.ofFloat(View.SCALE_Y, 1.0f, 1.1f)
            ).apply {
                duration = 1000L
                repeatCount = ObjectAnimator.INFINITE
                repeatMode = ObjectAnimator.REVERSE
                interpolator = AccelerateDecelerateInterpolator()
                start()
            }

            // Change icon tint to light purple when recording
            recordButton.setColorFilter(ResourcesCompat.getColor(resources,
R.color.md3_primary_container_light_purple, null))

            // Ripple animations
            val waveCircles = listOf(waveCircle1, waveCircle2, waveCircle3,
waveCircle4)
            val animationDelays = listOf(0L, 400L, 800L, 1200L) // as per CSS delays

            rippleAnimators.clear()
            waveCircles.forEachIndexed { index, circle ->
                val scaleX = ObjectAnimator.ofFloat(circle, View.SCALE_X, 0.3f, 2.5f)
                val scaleY = ObjectAnimator.ofFloat(circle, View.SCALE_Y, 0.3f, 2.5f)
                val alpha = ObjectAnimator.ofFloat(circle, View.ALPHA, 0.8f, 0.0f) //
from 10% opacity to 0

                val animatorSet = AnimatorSet().apply {
                    playTogether(scaleX, scaleY, alpha)
                    duration = 2000L // 2 seconds as per CSS
                    startDelay = animationDelays[index]
                    interpolator = LinearInterpolator() // cubic-bezier(0, 0.55,
0.45, 1) is similar to linear for alpha, and scale is smooth
                    addListener(object : android.animation.Animator.AnimatorListener
{
                        override fun onAnimationStart(animation:
android.animation.Animator) {
                            circle.visibility = View.VISIBLE
                        }
                        override fun onAnimationEnd(animation:
android.animation.Animator) {
                            // Reset properties for repeat if not stopped
                            if (isRecording) { // Only reset if still recording
                                circle.scaleX = 0.3f
                                circle.scaleY = 0.3f
```

```
                                circle.alpha = 0.8f
                                animation.start() // Restart the animation
                            } else {
                                circle.visibility = View.GONE
                                circle.scaleX = 0f
                                circle.scaleY = 0f
                                circle.alpha = 0f
                            }
                        }
                        override fun onAnimationCancel(animation:
android.animation.Animator) {
                            circle.visibility = View.GONE
                            circle.scaleX = 0f
                            circle.scaleY = 0f
                            circle.alpha = 0f
                        }
                        override fun onAnimationRepeat(animation:
android.animation.Animator) { /* Not used */ }
                    })
                    start()
                }
                rippleAnimators.add(animatorSet)
            }
        }

        private fun stopRecordingAnimation() {
            iconPulseAnimator?.cancel()
            iconPulseAnimator = null
            recordButton.scaleX = 1.0f
            recordButton.scaleY = 1.0f
            recordButton.setColorFilter(ResourcesCompat.getColor(resources,
R.color.md3_on_primary_white, null))

            rippleAnimators.forEach { it.cancel() }
            rippleAnimators.clear()

            // Ensure all circles are reset and hidden
            listOf(waveCircle1, waveCircle2, waveCircle3, waveCircle4).forEach {
circle ->
                circle.scaleX = 0f
                circle.scaleY = 0f
                circle.alpha = 0f
                circle.visibility = View.GONE
```

```
            }

            recordingProgressBar.visibility = View.GONE
            recordingProgressBar.progress = 0
        }

        private fun updatePitch() {
            f0UpKey = when (currentPitchState) {
                PitchState.MALE -> -16
                PitchState.FEMALE -> 16
                PitchState.NONE -> 0
            }
            updatePitchButtonsUI()
        }

        private fun updatePitchButtonsUI() {
            maleButton.isSelected = currentPitchState == PitchState.MALE
            femaleButton.isSelected = currentPitchState == PitchState.FEMALE
        }

        private fun getModelsListFromAssets(): List<String> {
            // NCNN uses .param and .bin files, but here we list only .ort and NCNN
param files (as a proxy)
            return    assets.list("rvc")?.filter    {    it.endsWith(".ort")    ||
it.endsWith(".param") } ?: emptyList()
        }

        private suspend fun loadBaseModels() {
            withContext(ortSessionDispatcher) {
                loadingIndicator.visibility = View.VISIBLE
            }
            try {
                sessRmvpe = loadModel("rmvpe.with_runtime_opt.ort")
                sessEmbed = loadModel("hubert.with_runtime_opt.ort")
            } catch (e: Exception) {
                withContext(Dispatchers.Main) {
                    Toast.makeText(this@MainActivityRefactor,  "Error  loading  base
models: ${e.message}", Toast.LENGTH_LONG).show()
                }
            } finally {
                withContext(Dispatchers.Main) {
                    loadingIndicator.visibility = View.GONE
                }
```

```
            }
        }

        private fun loadSelectedModel(modelName: String) {
            loadingIndicator.visibility = View.VISIBLE
            mainScope.launch(ortSessionDispatcher) {
                try {
                    // Close previous sessions
                    sessInfer?.close()
                    sessInfer = null
                    ncnnInference?.unloadModel()

                    currentModelName = modelName

                    if (modelName.endsWith(".ort")) {
                        // ONNX model loading logic (kept largely intact)
                        isNcnnModelLoaded = false
                        val modelFile = File(filesDir, modelName)
                        if (!modelFile.exists()) {
                            assets.open("rvc/$modelName").use { input ->
                                FileOutputStream(modelFile).use { output ->
                                    input.copyTo(output)
                                }
                            }
                        }
                        val options = OrtSession.SessionOptions()
                        sessInfer                                                    =
ortEnvironment.createSession(modelFile.absolutePath, options)

                    } else if (modelName.endsWith(".param")) {
                        // NCNN model loading logic
                        isNcnnModelLoaded = true
                        //  Assuming  NCNN  files  follow  a  .param/.bin  pair  naming
convention
                        val modelBinName = modelName.replace(".param", ".bin")
                        val paramPath = "rvc/$modelName"
                        val modelPath = "rvc/$modelBinName"
                        if (ncnnInference?.loadModel(assets, paramPath, modelPath) ==
true) {
                            Log.d("VoiceTransformer",    "NCNN    model    loaded:
${modelName}")
                        } else {
                            Log.e("VoiceTransformer",  "Error  loading  NCNN  model:
```

```
${modelName}")
                                throw IOException("NCNN model failed to load.")
                            }
                        }

                        withContext(Dispatchers.Main) {
                            updatePitch()
                        }

                        withContext(Dispatchers.Main) {
                            recordButton.isEnabled = true
                        }
                    } catch (e: Exception) {
                        Log.e("VoiceTransformer", "Error loading model: ${e.message}", e)
                        withContext(Dispatchers.Main) {
                             Toast.makeText(this@MainActivityRefactor,  "Error  loading
model: ${e.message}", Toast.LENGTH_LONG).show()
                        }
                    } finally {
                        withContext(Dispatchers.Main) {
                            loadingIndicator.visibility = View.GONE
                        }
                    }
                }
            }

            private fun loadModel(assetName: String): OrtSession {
                val modelPath = assetFilePath(this, assetName)
                return                            ortEnvironment.createSession(modelPath,
OrtSession.SessionOptions())
            }

            private fun assetFilePath(context: Context, assetName: String): String {
                val file = File(context.filesDir, assetName)
                if (!file.exists()) {
                    context.assets.open(assetName).use { inputStream ->
                        FileOutputStream(file).use { outputStream ->
                            inputStream.copyTo(outputStream)
                        }
                    }
                }
                return file.absolutePath
            }
```

```
        private fun checkPermissions() {
            if (ContextCompat.checkSelfPermission(this,
Manifest.permission.RECORD_AUDIO) != PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions(this,
arrayOf(Manifest.permission.RECORD_AUDIO), 123)
            }
        }

        private fun getNextSafeLength(currentLength: Int): Int {
            val MIN_SAFE_LENGTH = 1120
            val CYCLE_STEP = 160
            val SUPER_CYCLE = 640

            if (currentLength <= MIN_SAFE_LENGTH) {
                return MIN_SAFE_LENGTH
            }

            val remainder = currentLength % SUPER_CYCLE
            if (currentLength % CYCLE_STEP == 0 && (remainder == 0 || remainder ==
480)) {
                return currentLength
            }

            var targetLength = ((currentLength + CYCLE_STEP - 1) / CYCLE_STEP) *
CYCLE_STEP

            while (true) {
                val targetRemainder = targetLength % SUPER_CYCLE
                if (targetRemainder == 0 || targetRemainder == 480) {
                    return targetLength
                }
                targetLength += CYCLE_STEP
            }
        }

        private fun startRecording() {
            if (!isNcnnModelLoaded && sessInfer == null) {
                Toast.makeText(this, "Please select a model first",
Toast.LENGTH_SHORT).show()
                return
            }
            if (ActivityCompat.checkSelfPermission(this,
```

```
Manifest.permission.RECORD_AUDIO) != PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(this, "Audio recording permission required",
Toast.LENGTH_SHORT).show()
                return
            }

            playerControls.visibility = View.GONE
            mediaPlayer?.release()
            mediaPlayer = null
            audioDataStream = ByteArrayOutputStream()
            processingQueue.clear()

            // Reset logging variables
            totalProcessingTimeMs = 0L
            totalRecordedAudioSamples = 0L
            isSampleRateCalculated = false

            // Reset performance logging variables
            totalPreprocessingTimeMs = 0L
            totalHubertTimeMs = 0L
            totalRmvpeTimeMs = 0L
            totalRvcGeneratorTimeMs = 0L
            totalPostprocessingTimeMs = 0L
            processedChunkCount = 0

            recorder = AudioRecord(MediaRecorder.AudioSource.MIC, inputSampleRate,
channelConfig, audioFormat, bufferSize)
            recorder?.startRecording()
            isRecording = true

            recordingProgressBar.progress = 0

            // Start recording timer for progress bar and duration limit
            recordingTimerJob = mainScope.launch(Dispatchers.Main) {
                var elapsedTime = 0L
                val updateInterval = 100L
                while (isRecording && elapsedTime < MAX_RECORDING_DURATION_MS) {
                    delay(updateInterval)
                    elapsedTime += updateInterval
                    recordingProgressBar.progress = elapsedTime.toInt()
                    if (elapsedTime >= MAX_RECORDING_DURATION_MS) {
                        stopRecording()
                    }
```

```
                }
            }

            // 1. PRODUCER (Recording)
            producerJob = mainScope.launch(Dispatchers.IO) {
                val audioBuffer = ShortArray(bufferSize)
                val chunkBuffer = mutableListOf<Short>()
                while (isRecording) {
                    val readSamples = recorder?.read(audioBuffer, 0,
audioBuffer.size) ?: 0
                    if (readSamples > 0) {
                        chunkBuffer.addAll(audioBuffer.take(readSamples))
                        while (chunkBuffer.size >= chunkSize) {
                            val chunkToProcess = chunkBuffer.subList(0,
chunkSize).toShortArray()
                            chunkBuffer.subList(0, chunkSize).clear()
                            processingQueue.offer(chunkToProcess)
                        }
                    }
                }
                // Handle remaining chunk after stopping
                if (chunkBuffer.isNotEmpty()) {
                    val lastChunk = chunkBuffer.toShortArray()

                    if (isNcnnModelLoaded) {
                        processingQueue.offer(lastChunk)
                    } else {
                        // For ONNX, use the existing safe length logic
                        val safeLength = getNextSafeLength(lastChunk.size)

                        if (safeLength == lastChunk.size) {
                            processingQueue.offer(lastChunk)
                        } else {
                            val paddedChunk = ShortArray(safeLength) { 0 }
                            lastChunk.copyInto(paddedChunk, 0, 0, lastChunk.size)
                            processingQueue.offer(paddedChunk)
                            Log.d("VoiceTransformer", "Last chunk ${lastChunk.size}
padded to $safeLength for ONNX")
                        }
                    }
                }
            }
```

```
                // 2. CONSUMER (Processing)
                processingJob = mainScope.launch(ortSessionDispatcher) {
                    // Wait a bit for producerJob to initialize
                    delay(50)
                    // Continue while: (queue is not empty) OR (Producer is still alive
and writing data)
                    while     (isActive     &&     (processingQueue.isNotEmpty()     ||
producerJob?.isActive == true)) {
                        processingQueue.poll()?.let { chunk ->
                            val processedChunk = processChunk(chunk)
                            // Convert IntArray back to ByteArray (little endian)
                            val  byteBuffer  =  ByteBuffer.allocate(processedChunk.size  *
2).order(ByteOrder.LITTLE_ENDIAN)
                            processedChunk.forEach { byteBuffer.putShort(it.toShort()) }
                            audioDataStream?.write(byteBuffer.array())
                        } ?: delay(10) // If queue is empty but producer is alive - wait
                    }
                }
            }

            private fun stopRecording() {
                if (!isRecording) return
                isRecording = false // Signal producerJob to finish reading loop

                recorder?.stop()
                recorder?.release()
                recorder = null

                recordingTimerJob?.cancel()
                recordingTimerJob = null

                loadingIndicator.visibility = View.VISIBLE

                mainScope.launch {
                    // First, wait for Producer to finish reading the buffer and put the
tail into the queue
                    producerJob?.join()

                    // Then wait for Consumer to process everything remaining in the queue
                    processingJob?.join()

                    loadingIndicator.visibility = View.GONE
```

```
                if (audioDataStream?.size() ?: 0 > 0) {
                    // Log total processing and audio length
                    val                totalAudioDurationSeconds                =
totalRecordedAudioSamples.toDouble() / inputSampleRate
                    Log.i("VoiceTransformer",      "Total      audio      processed:
${String.format("%.2f",   totalAudioDurationSeconds)}   s.   Total   processing   time:
${totalProcessingTimeMs} ms.")

                    // Log average times for each stage
                    if (processedChunkCount > 0) {
                        val avgPreprocessing = totalPreprocessingTimeMs.toDouble() /
processedChunkCount
                        val    avgHubert    =    totalHubertTimeMs.toDouble()    /
processedChunkCount
                        val     avgRmvpe     =     totalRmvpeTimeMs.toDouble()     /
processedChunkCount
                        val  avgRvcGenerator  =  totalRvcGeneratorTimeMs.toDouble()  /
processedChunkCount
                        val avgPostprocessing = totalPostprocessingTimeMs.toDouble()
/ processedChunkCount
                        val  totalAvg  =  avgPreprocessing  +  avgHubert  +  avgRmvpe  +
avgRvcGenerator + avgPostprocessing

                        Log.i("VoiceTransformer", "--- Average Processing Times Per
Chunk (${processedChunkCount} chunks) ---")
                        Log.i("VoiceTransformer",  String.format("1.  Pre-processing:
%.2f ms", avgPreprocessing))
                        Log.i("VoiceTransformer",     String.format("2.     Feature
Extraction (HuBERT): %.2f ms", avgHubert))
                        Log.i("VoiceTransformer", String.format("3. Pitch Extraction
(RMVPE): %.2f ms", avgRmvpe))
                        Log.i("VoiceTransformer",     String.format("4.     Acoustic
Synthesis (RVC Generator): %.2f ms", avgRvcGenerator))
                        Log.i("VoiceTransformer", String.format("5. Post-processing:
%.2f ms", avgPostprocessing))
                        Log.i("VoiceTransformer",  String.format("Total  Average  Per
Chunk: %.2f ms", totalAvg))
                        Log.i("VoiceTransformer", "-----------------------------------
-----------------")
                    }

                    setupMediaPlayer()
                    shareAudio()
```

```
                } else {
                    Toast.makeText(this@MainActivityRefactor, "Recording is empty",
Toast.LENGTH_SHORT).show()
                }
                playPauseButton.setImageResource(android.R.drawable.ic_media_play)
                seekBar.progress = 0
            }
        }

        private fun processChunk(chunk: ShortArray): IntArray {
            val overallChunkStartTime = System.currentTimeMillis()

            // 1. Pre-processing
            val preprocStartTime = System.currentTimeMillis()

            // Remember the size of the data that ACTUALLY went into the model
            val audioPadSource: ShortArray
            if (isNcnnModelLoaded) {
                audioPadSource = if (chunk.size < chunkSize) {
                    chunk + ShortArray(chunkSize - chunk.size) { 0 }
                } else {
                    chunk
                }
            } else {
                audioPadSource = if (chunk.size < 2560) {
                    chunk + ShortArray(2560 - chunk.size) { 0 }
                } else {
                    chunk
                }
            }

            val audioPad = FloatArray(audioPadSource.size) { i -> audioPadSource[i]
/ 32768.0f }
            val pitchfInit = FloatArray(audioPad.size / 160) { 0f }
            totalPreprocessingTimeMs      +=      (System.currentTimeMillis()      -
preprocStartTime)

            // 2. Pitch Extraction (RMVPE)
            val rmvpeStartTime = System.currentTimeMillis()
            val rmvpeInputs = mapOf(
                "audio_pad" to OnnxTensor.createTensor(ortEnvironment, audioPad),
                "pitchf" to OnnxTensor.createTensor(ortEnvironment, pitchfInit),
                "f0_up_key"        to        OnnxTensor.createTensor(ortEnvironment,
```

```
longArrayOf(f0UpKey.toLong()))
                )
                val rmvpeOuts = sessRmvpe.run(rmvpeInputs)
                val pitch = (rmvpeOuts[0].value as Array<LongArray>)
                val pitchfOut = rmvpeOuts[1].value as Array<FloatArray>
                rmvpeOuts.close()
                totalRmvpeTimeMs += (System.currentTimeMillis() - rmvpeStartTime)

                // 3. Feature Extraction (HuBERT)
                val hubertStartTime = System.currentTimeMillis()
                val          embedInputs          =          mapOf("audio_pad"          to
OnnxTensor.createTensor(ortEnvironment, audioPad))
                val embedOuts = sessEmbed.run(embedInputs)
                val feats = embedOuts[0].value as Array<Array<FloatArray>>
                embedOuts.close()
                totalHubertTimeMs += (System.currentTimeMillis() - hubertStartTime)

                // 4. Inference (RVC)
                val audioOut: IntArray
                val rvcGeneratorStartTime = System.currentTimeMillis()

                if (isNcnnModelLoaded) {
                    audioOut = processChunkNCNN(feats, pitch, pitchfOut, chunk.size)
                    totalRvcGeneratorTimeMs  +=  ncnnInference?.getLastInferenceTimeMs()
?: 0
                } else {
                    val inferSession = sessInfer ?: return IntArray(0)
                    val inferInputs = mapOf(
                        "feats" to OnnxTensor.createTensor(ortEnvironment, feats),
                        "pitch" to OnnxTensor.createTensor(ortEnvironment, pitch),
                        "pitchf" to OnnxTensor.createTensor(ortEnvironment, pitchfOut)
                    )
                    val inferOuts = inferSession.run(inferInputs)
                    val       outputTensorValue       =       inferOuts[0].value       as
Array<Array<FloatArray>>
                    val floatResult = outputTensorValue.flattenDeep()
                    audioOut = floatResult.map { it.toInt() }.toIntArray()
                    inferOuts.close()
                    totalRvcGeneratorTimeMs     +=     (System.currentTimeMillis()     -
rvcGeneratorStartTime)
                }

                // 5. Post-processing and FREQUENCY CALCULATION
```

```
            val postprocStartTime = System.currentTimeMillis()

            // Calculate frequency only once, on the first normal chunk
            if (!isSampleRateCalculated && audioOut.isNotEmpty() &&
audioPadSource.isNotEmpty()) {

                // Calculate the stretching coefficient using the data that ACTUALLY
went into the model (audioPadSource),
                // not the raw chunk, to avoid padding errors.
                val ratio = audioOut.size.toDouble() / audioPadSource.size.toDouble()
                val rawRate = (ratio * inputSampleRate).toInt()


                outputSampleRate = when {
                    rawRate < 36000 -> 32000 // For v1 models (usually ~32k)
                    rawRate < 44000 -> 40000 // For most v2 models (usually ~40k)
                    else -> 48000            // High quality v2 (usually ~48k)
                }

                isSampleRateCalculated = true
                Log.i("VoiceTransformer", "Model Ratio: $ratio. Raw Rate: $rawRate
Hz. Set Output Rate: $outputSampleRate Hz")
            }

            totalPostprocessingTimeMs += (System.currentTimeMillis() -
postprocStartTime)

            processedChunkCount++
            val chunkProcessingTime = System.currentTimeMillis() -
overallChunkStartTime
            totalProcessingTimeMs += chunkProcessingTime
            totalRecordedAudioSamples += chunk.size

            return audioOut
        }

        private fun processChunkNCNN(
            feats: Array<Array<FloatArray>>,
            pitch: Array<LongArray>,
            pitchfOut: Array<FloatArray>,
            originalChunkSize: Int
        ): IntArray {
            val ncnnInfer = ncnnInference ?: return IntArray(0)
```

```
            // The inputs (feats, pitch, pitchfOut) were generated from an audio chunk
that was
            // potentially padded to chunkSize. No further padding of these tensors
is needed.

            // 1. Prepare NCNN inputs
            val in0_data = feats.flattenDeep()
            val in1_data_long = pitch.flatten()
            val in1_data_int = in1_data_long.map { it.toInt() }.toIntArray()
            val in2_data = pitchfOut.flatten()

            // 2. Run inference
            var  outputFloatArray  =  ncnnInfer.runInference(in0_data,  in1_data_int,
in2_data)

            if (outputFloatArray == null) {
                 Log.e("VoiceTransformer", "NCNN Inference returned null output.")
                 return IntArray(0)
            }

            // 3. Trim the output if the original input chunk was smaller than
chunkSize.
            if (originalChunkSize < chunkSize) {
                val ratio = originalChunkSize.toDouble() / chunkSize.toDouble()
                val targetOutputSize = (outputFloatArray.size * ratio).toInt()
                if (outputFloatArray.size > targetOutputSize) {
                    outputFloatArray = outputFloatArray.copyOf(targetOutputSize)
                    Log.d("VoiceTransformer",    "NCNN    output    trimmed    from
${outputFloatArray.size} to $targetOutputSize")
                }
            }

            // 4. Convert float output back to IntArray
            return outputFloatArray.map { it.toInt() }.toIntArray()
        }

        // Helper extension functions to flatten arrays
        private fun Array<Array<FloatArray>>.flattenDeep(): FloatArray {
            return this.flatMap { it.flatMap { it.asIterable() } }.toFloatArray()
        }

        private fun Array<LongArray>.flatten(): LongArray {
```

```
            return this.flatMap { it.asIterable() } .toLongArray()
        }

        private fun Array<FloatArray>.flatten(): FloatArray {
            return this.flatMap { it.asIterable() }.toFloatArray()
        }

        private fun togglePlayback() {
            mediaPlayer?.let {
                if (it.isPlaying) {
                    it.pause()

playPauseButton.setImageResource(android.R.drawable.ic_media_play)
                } else {
                    it.start()

playPauseButton.setImageResource(android.R.drawable.ic_media_pause)
                    updateSeekBar()
                }
            }
        }

        private fun setupMediaPlayer() {
            val audioData = audioDataStream?.toByteArray()
            if (audioData == null || audioData.isEmpty()) {
                Toast.makeText(this, "No data to play", Toast.LENGTH_SHORT).show()
                return
            }
            try {
                currentAudioFile    =    File.createTempFile("temp_audio",    ".m4a",
cacheDir)
                encodePcmToAac(audioData,  currentAudioFile!!,  outputSampleRate,  1,
128000)

                mediaPlayer = MediaPlayer().apply {
                    setDataSource(currentAudioFile!!.absolutePath)
                    prepare()
                    setOnCompletionListener {

playPauseButton.setImageResource(android.R.drawable.ic_media_play)
                        seekBar.progress = 0
                        currentTimeTextView.text = formatTime(0)
                    }
```

```
                }
                val duration = mediaPlayer?.duration ?: 0
                seekBar.max = duration
                totalTimeTextView.text = formatTime(duration)
                currentTimeTextView.text = formatTime(0)
                mediaPlayer?.start()
                playPauseButton.setImageResource(android.R.drawable.ic_media_pause)
                updateSeekBar()
            } catch (e: Exception) {
                e.printStackTrace()
                Toast.makeText(this, "Error preparing for playback: ${e.message}",
Toast.LENGTH_LONG).show()
            }
        }

        private fun updateSeekBar() {
            mainScope.launch {
                while (mediaPlayer?.isPlaying == true) {
                    val currentPosition = mediaPlayer?.currentPosition ?: 0
                    seekBar.progress = currentPosition
                    currentTimeTextView.text = formatTime(currentPosition)
                    delay(100)
                }
            }
        }

        private fun formatTime(milliseconds: Int): String {
            val minutes = (milliseconds / 1000) / 60
            val seconds = (milliseconds / 1000) % 60
            return String.format("%02d:%02d", minutes, seconds)
        }

        private fun calculateSpanCount(): Int {
            val displayMetrics = resources.displayMetrics
            val screenWidthPx = displayMetrics.widthPixels

            // Convert dp values to pixels
            val cardWidthPx = TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,
100f, displayMetrics) // Reduced card width
            val                              cardMarginPx                              =
TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP, 8f, displayMetrics)

            // Total width occupied by one card including its margins (left and right)
```

```
            val cardWidthWithMarginsPx = cardWidthPx + (cardMarginPx * 2)

            return (screenWidthPx / cardWidthWithMarginsPx).toInt().coerceAtLeast(1)
        }


        @Throws(IOException::class)
        private fun encodePcmToAac(pcmData: ByteArray, outputFile: File, sampleRate:
Int, channels: Int, bitRate: Int) {
            val                              mediaFormat                              =
MediaFormat.createAudioFormat(MediaFormat.MIMETYPE_AUDIO_AAC, sampleRate, channels)
            mediaFormat.setInteger(MediaFormat.KEY_BIT_RATE, bitRate)
            mediaFormat.setInteger(MediaFormat.KEY_AAC_PROFILE,
MediaCodecInfo.CodecProfileLevel.AACObjectLC)
            mediaFormat.setInteger(MediaFormat.KEY_MAX_INPUT_SIZE,  1024  *  64)  //
Limit buffer size

            val                                 mediaCodec                                =
MediaCodec.createEncoderByType(MediaFormat.MIMETYPE_AUDIO_AAC)
            mediaCodec.configure(mediaFormat,              null,              null,
MediaCodec.CONFIGURE_FLAG_ENCODE)
            mediaCodec.start()

            val       mediaMuxer       =       MediaMuxer(outputFile.absolutePath,
MediaMuxer.OutputFormat.MUXER_OUTPUT_MPEG_4)
            var trackIndex = -1
            var isMuxerStarted = false

            val bufferInfo = MediaCodec.BufferInfo()
            val pcmByteBuffer = ByteBuffer.wrap(pcmData) // Data (Little Endian by
default in WAV, but here Raw bytes)

            var totalInputSamples: Long = 0
            var inputDone = false

            val timeoutUs = 5000L // 5ms

            try {
                while (true) {
                    // 1. Try to feed data to the input (INPUT)
                    if (!inputDone) {
                        val                    inputBufferIndex                    =
mediaCodec.dequeueInputBuffer(timeoutUs)
```

```
                        if (inputBufferIndex >= 0) {
                            val                    inputBuffer                    =
mediaCodec.getInputBuffer(inputBufferIndex)
                            inputBuffer?.clear()

                            val availableSize = minOf(inputBuffer?.remaining() ?: 0,
pcmByteBuffer.remaining())
                            if (availableSize > 0) {
                                inputBuffer?.put(pcmByteBuffer.array(),
pcmByteBuffer.position(), availableSize)
                                pcmByteBuffer.position(pcmByteBuffer.position()    +
availableSize)
                            }

                            inputDone = !pcmByteBuffer.hasRemaining()

                            val        flags        =        if        (inputDone)
MediaCodec.BUFFER_FLAG_END_OF_STREAM else 0

                            // Calculate time: (Samples / Frequency) * 1 million
                            // Divide availableSize by 2, because 16-bit = 2 bytes
                            val presentationTimeUs = (totalInputSamples * 1000000L)
/ sampleRate

                            mediaCodec.queueInputBuffer(inputBufferIndex,         0,
availableSize, presentationTimeUs, flags)

                            totalInputSamples += (availableSize / 2)
                        }
                    }

                    // 2. Try to retrieve data from the output (OUTPUT)
                    // Loop while the codec gives data to avoid buffer overflow
                    var encoderOutputAvailable = true
                    while (encoderOutputAvailable) {
                        val                    outputBufferIndex                    =
mediaCodec.dequeueOutputBuffer(bufferInfo, timeoutUs)
                        if (outputBufferIndex >= 0) {
                            val                    outputBuffer                    =
mediaCodec.getOutputBuffer(outputBufferIndex)

                            if                (bufferInfo.flags                and
MediaCodec.BUFFER_FLAG_CODEC_CONFIG != 0) {
```

```
                                bufferInfo.size = 0
                            }

                            if (bufferInfo.size != 0) {
                                if (!isMuxerStarted) {
                                    trackIndex                                    =
mediaMuxer.addTrack(mediaCodec.outputFormat)
                                    mediaMuxer.start()
                                    isMuxerStarted = true
                                }

                                outputBuffer?.position(bufferInfo.offset)
                                outputBuffer?.limit(bufferInfo.offset              +
bufferInfo.size)
                                mediaMuxer.writeSampleData(trackIndex,
outputBuffer!!, bufferInfo)
                            }

                            mediaCodec.releaseOutputBuffer(outputBufferIndex, false)

                            if                 (bufferInfo.flags                 and
MediaCodec.BUFFER_FLAG_END_OF_STREAM != 0) {
                                return // ALL DONE, EXIT FUNCTION
                            }
                        }        else        if        (outputBufferIndex        ==
MediaCodec.INFO_OUTPUT_FORMAT_CHANGED) {
                            if   (isMuxerStarted)   throw   RuntimeException("Format
changed twice")
                            trackIndex                                            =
mediaMuxer.addTrack(mediaCodec.outputFormat)
                            mediaMuxer.start()
                            isMuxerStarted = true
                        } else {
                            encoderOutputAvailable = false // No data yet, return to
feeding input
                        }
                    }
                }
            } catch (e: Exception) {
                e.printStackTrace()
                Log.e("VoiceTransformer", "Encoding error: ${e.message}")
            } finally {
                if (isMuxerStarted) {
```

```
                    try { mediaMuxer.stop() } catch (e: Exception) {
e.printStackTrace() }
                }
                mediaMuxer.release()
                mediaCodec.stop()
                mediaCodec.release()
            }
        }

        inner class ModelAdapter(
            private val models: List<String>,
            private val onModelClick: (String, Int) -> Unit
        ) : RecyclerView.Adapter<ModelAdapter.ModelViewHolder>() {

            override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ModelViewHolder {
                val view = LayoutInflater.from(parent.context)
                    .inflate(R.layout.item_model_card, parent, false)
                return ModelViewHolder(view)
            }

            override fun onBindViewHolder(holder: ModelViewHolder, position: Int) {
                val modelName = models[position]
                holder.bind(modelName)
                holder.itemView.setOnClickListener {
                    onModelClick(modelName, holder.characterImageResId)
                }
            }

            override fun getItemCount(): Int = models.size

            inner class ModelViewHolder(itemView: View) :
RecyclerView.ViewHolder(itemView) {
                private val modelNameTextView: TextView =
itemView.findViewById(R.id.modelNameTextView)
                private val modelImageView: ImageView =
itemView.findViewById(R.id.modelImageView)
                var characterImageResId: Int = 0

                fun bind(modelName: String) {
                    // Get the base name by removing any extension
                    val baseName = modelName.substringBeforeLast(".")
```

```
                    // Further clean up the name for display
                    val displayName = baseName
                        .substringBeforeLast(".with_runtime_opt", baseName)
                        .replace("_", " ")
                    modelNameTextView.text = displayName

                    // Use the same cleaned base name to find the image resource
                    val imageName = baseName
                        .substringBeforeLast(".with_runtime_opt", baseName)
                        .replace(" ", "_")
                        .replace("-", "_")
                        .lowercase()

                    val imageResId = itemView.context.resources.getIdentifier(
                        imageName, "drawable", itemView.context.packageName
                    )

                    if (imageResId != 0) {
                        modelImageView.setImageResource(imageResId)
                        characterImageResId = imageResId
                    } else {

modelImageView.setImageResource(R.drawable.ic_launcher_background)
                        characterImageResId = R.drawable.ic_launcher_background
                    }
                }
            }
        }

        private fun updateCharacterImage(imageResId: Int) {
            if (imageResId != 0) {
                characterImageView.setImageResource(imageResId)
                characterImageWrapper.visibility = View.VISIBLE
                //    Make    recordButton's    background    transparent    to    show
characterImageView
                recordButton.backgroundTintList                                    =
ColorStateList.valueOf(Color.TRANSPARENT)
                // Ensure the microphone icon remains visible on top of the character
image
                recordButton.setColorFilter(ContextCompat.getColor(this,
R.color.md3_on_primary_white))
            } else {
                // Hide character image and revert recordButton to default color
```

```
                characterImageWrapper.visibility = View.GONE
                recordButton.backgroundTintList                                    =
ContextCompat.getColorStateList(this, R.color.md3_primary_purple)
                recordButton.setColorFilter(ContextCompat.getColor(this,
R.color.md3_on_primary_white))
            }
        }

        private fun shareAudio() {
            currentAudioFile?.let { file ->
                if (file.exists()) {
                    try {
                        val    contentUri    =    FileProvider.getUriForFile(this,
"${applicationContext.packageName}.provider", file)
                        val shareIntent = Intent().apply {
                            action = Intent.ACTION_SEND
                            putExtra(Intent.EXTRA_STREAM, contentUri)
                            type = "audio/mp4"
                            addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
                        }
                        startActivity(Intent.createChooser(shareIntent,      "Share
Audio"))
                    } catch (e: IllegalArgumentException) {
                        Toast.makeText(this, "Error: FileProvider not configured",
Toast.LENGTH_LONG).show()
                        e.printStackTrace()
                    }
                }
            } ?: run {
                Toast.makeText(this,      "Audio      file      not      found",
Toast.LENGTH_SHORT).show()
            }
        }

        override fun onDestroy() {
            super.onDestroy()
            recorder?.release()
            mediaPlayer?.release()
            mainScope.cancel()
            recordingTimerJob?.cancel()
            sessRmvpe.close()
            sessEmbed.close()
            sessInfer?.close()
```

```
            ncnnInference?.unloadModel()
            ortEnvironment.close()
            ortSessionDispatcher.close()
            // Stop and clear any ongoing animations
            stopRecordingAnimation()
        }

    }
```