НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«___»_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Застосування нейронних мереж для клонування та генерації голосу у реальному часі»

Виконав :
студент IV курсу, групи КА-05
Нікітаєв Нікіта Віталійович _____

Керівник:
доцент, к.ф.-м.н.,
Барановська Леся Валеріївна _____

Консультант з економічного розділу:
Доцент, к. е. н.
Рощина Надія Василівна _____

Консультант з нормоконтролю:
К.ф.-м.н.
Статкевич Віталій Михайлович _____

Рецензент:
Доцент СП ІПСА, к.т.н.,
Харченко Костянтин Васильович _____

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

Київ – 2024 року

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

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Оксана ТИМОЩУК

«___»_____2024 р.

ЗАВДАННЯ

на дипломну роботу студенту

Нікітаєву Нікіті Віталійовичу

1. Тема роботи «Застосування нейронних мереж для клонування та генерації голосу у реальному часі», керівник роботи Барановська Леся Валеріївна, доцент, к.ф.-м.н., затверджені наказом по університету від «___»_____ 2024 р. №_____

2. Термін подання студентом роботи 11.06.2024

3. Вихідні дані до роботи: теоритичні відомості про системи синтезу мовлення; власний набір аудіо даних.

4. Зміст роботи: Дослідження предметної області; огляд технологій синтезу мовлення та клонування голосу; реалізація програмного продукту та аналіз результатів експерименту.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) – презентація, демонстраційні відео, діаграма архітектури Tacotron2.

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
економічний	Рощина Н.В., доц., к.е.н.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формулювання тематики дослідження	05.02.2024 – 26.02.2024	виконано
2	Аналіз актуальності задач стосовно тематики дослідження	12.02.2024 – 04.04.2024	виконано
3	Аналіз відомих результатів стосовно тематики дослідження	19.02.2024 – 11.04.2024	виконано
4	Формулювання задач дослідження	11.04.2024 – 01.04.2024	виконано
5	Уточнення теми дипломної роботи	06.04.2024	виконано
6	Збір статичних даних, попередній аналіз даних	01.04.2024 – 15.04.2024	виконано
7	Розробка програмного продукту для виконання експериментів	15.04.2024 – 10.05.2024	виконано
8	Виконання обчислювальних експериментів, аналіз та оформлення результатів	15.04.2024 – 20.05.2024	виконано
9	Оформлення пояснювальної записки у цілому	20.05.2024 – 25.05.2024	виконано
10	Підготовка презентації для захисту	25.05.2024 – 27.05.2024	виконано
11	Попередній захист дипломної роботи	28.05.2024 – 02.06.2024	виконано

Студент Нікіта НІКІТАЄВ

Керівник Леся БАРАНОВСЬКА

* Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломної роботи.

РЕФЕРАТ

Дипломна робота: 136 с., 25 рис., 6 табл., 2 дод., 39 джерел.

НЕЙРОННІ МЕРЕЖІ, КЛОНУВАННЯ ГОЛОСУ, СИНТЕЗ МОВЛЕННЯ, МОДЕЛІ НА ОСНОВІ ПОШУКУ

Темою роботи є використання моделей та методів тренування нейронних мереж для генерації та копіювання голосу.

Об’єктом дослідження є способи реалістичної конверсії аудіо.

Предметом дослідження є моделі «текст в голос» та клонування голосу на основі пошуку.

Мєтою роботи є дослідження застосування нейронних мереж для синтезу мовлення та клонування голосу, зокрема у режимі реального часу.

Актуальність роботи пов'язана із популяризацією досліджуваної моделі у мережі.

У результаті роботи було детально розглянуто сучасні технології клонування голосу та синтезу мовлення, а також натреновано модель, яка копіює голос автора даної роботи. Для демонстрації її роботи було створено відеоматеріали.

ABSTRACT

Bachelor's Thesis: 136 p., 25 fig., 6 tab., 2 app., 39 references.

NEURAL NETWORKS, VOICE CLONING, SPEECH SYNTHESIS, RETRIEVAL-BASED VOICE CONVERSION

The topic of the work is the usage of models and methods of neural network training for voice generation and voice cloning.

The object of study is the set of methods of realistic audio conversion.

The subject of the study is text-to-speech models and search-based voice cloning.

The purpose of the study is to investigate the use of neural networks for speech synthesis and voice cloning, in particular in real time.

The relevance of the work is related to the popularization of the model under study in the Internet.

As a result of the work the modern technologies of voice cloning and speech synthesis have been reviewed in detail, and a model that copies the voice of the author of this study has been trained. Video materials were created to demonstrate its work.

ЗМІСТ

ВСТУП

Людський голос є унікальним і потужним інструментом для спілкування, передачі емоцій і вираження особистості. Зі швидким розвитком штучного інтелекту та машинного навчання можливість генерувати людський голос і маніпулювати ним стала захоплюючою областю дослідження із багатою областю використання: від екранних дикторів у операційних системах до повноцінних голосових інтерфейсів.

Клонування голосу є ключовим напрямом у штучному інтелекті, що фокусується на перетворенні голосу людини на голос іншої особи зі збереженням лінгвістичного змісту. Розташоване у ширшій сфері синтезу мовлення, ця гілка досліджень охоплює спроби змінити такі властивості мовлення, як ідентичність голосу, емоції та акценти. Як зауважив у 1922 році Джон Стюарт, першопроходець у галузі синтезу мовлення, основний виклик у створенні штучного мовлення полягає не в тому, щоб генерувати мовлення як таке, а в тому, щоб маніпулювати апаратом, який лежить в його основі[1]. Таким чином, штучний голос, який спеціально спрямований на маніпулювання голосовою ідентичністю, стає особливо складним напрямком досліджень у галузі обробки мовлення.

Зусилля, спрямовані на ефективне маніпулювання властивостями мовлення, тривають з моменту зародження комп'ютерного синтезу мовлення в 1950-х роках. Поява цифрової обробки сигналів у 1970-х роках значно полегшила контроль параметрів для маніпулювання мовленням. Хоча початковий поштовх до розвитку віртуального мовлення був зумовлений новизною та цікавістю,

1 J. Q. Stewart, *An electrical analogue of the vocal organs.* Nature, Vol.1, pages 311–312

технологічний прогрес, що охоплює статистичне моделювання та глибоке навчання, вплинувши на численні реальні застосування.

Загалом, характеристики мовця можна окреслити трьома факторами: лінгвістичними, надсегментними та сегментними факторами. Лінгвістичні фактори охоплюють такі аспекти, як структура речення, лексичний вибір та ідіолект, тоді як надсегментні фактори стосуються просодичних характеристик мовних сигналів, а сегментні фактори пов'язані з короткочасними характеристиками, такими як спектр та форманти. У сценаріях, де лінгвістичний зміст залишається незмінним, як надсегментні, так і сегментні фактори відіграють ключову роль у визначенні індивідуальності мовця.

Те, що раніше здавалося неможливим, швидко стає реальністю. Відкриття WaveNet для загального використання у 2016 р. дало сильний поштовх у дослідженні аудіо генеративних моделей[2], завдяки чому з’явилося безліч комерційних проектів із синтезу та клонування голосу, такі як ElevenLabs та Speechify. Втім, упевнені користувачі мережі можуть (відносно без проблем) користуватися численними open-source проектами, таким як Coqui TTS, VITS, so-vits-SVC та новітнім RVC-project.

Перший розділ цієї роботи присвячено цифровому аудіо та роботі із ним.

Другий розділ роботи детально розглядає нейронні мережі, в особливості TTS та VC, фокусуючись на найбільш сучасних.

Третій розділ присвячено демонстрації роботи xTTSv2 та тренуванню власної RVC моделі.

2 URL: https://www.linkedin.com/advice/3/how-can-wavenet-enhance-voice-based-interactions

1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Вступ до аудіоданих

Перетворення Фур'є є потужним інструментом для аналізу аудіосигналів, однак більшість сигналів, таких як музика або записи голосу, мають неперіодичні характеристики, тобто їхні частоти змінюються з часом. Традиційний аналіз Фур'є не може врахувати цю часову еволюцію, оскільки він обчислює частоти та їхні амплітуди протягом усієї тривалості сигналу, нехтуючи часовими варіаціями. Отже, коли часова динаміка має вирішальне значення для вирішення конкретних завдань, алгоритм короткочасного перетворення Фур'є (Short-Time Fourier Transform, STFT, далі - КЧПФ) замінює класичний підхід.

КЧПФ розбиває сигнал на короткі часові сегменти, що перекриваються, і обчислює перетворення Фур'є окремо для кожного сегмента. Це дозволяє аналізувати частотний склад сигналу в часі, надаючи уявлення про те, як частоти змінюються з часом. КЧПФ особливо цінний для таких завдань, як частотно-часовий аналіз, спектральний аналіз нестаціонарних сигналів і виявлення перехідних процесів у сигналах.

Математично КЧПФ сигналу x(t) визначається так:

$$STFT\{x(t)\}(\tau, \omega) \equiv X(\tau, \omega) = \int_{-\infty}^{\infty} x(t)w(t - \tau)e - i\omega t dt, \quad (1.1)$$

де X(t,ω) представляє часово-частотне представлення сигналу, ω(t)- віконна функція, яка використовується для локалізації аналізу в часі, а ω - кутова частота.

КЧПФ можна ефективно обчислити за допомогою алгоритму швидкого перетворення Фур’є (Fast Fourier Transform, FFT, далі – ШПФ), застосовуючи

віконну функцію до кожного часового відрізку сигналу перед обчисленням перетворення Фур'є. Найпоширенішими віконними функціями є вікно Хеммінга, вікно Ганнінга і вікно Гауса, кожна з яких має свої властивості, придатні для різних застосувань.

1.2 Дискретизація сигналів

Звукова хвиля, за своєю природою неперервна, втілює нескінченний масив значень сигналу в межах певного часового інтервалу. Ця характеристика створює проблеми для цифрових систем, які працюють зі скінченними масивами. Таким чином, для цифрової обробки, зберігання та передачі безперервна звукова хвиля потребує перетворення в дискретні значення, що становлять цифрове представлення.

У наборах аудіоданих можна зустріти цифрові файли, що містять звукові фрагменти, такі як текстові розповіді або музика, часто у різних форматах, таких як .wav (Waveform Audio File), .flac (Free Lossless Audio Codec) та .mp3 (MPEG-1 Audio Layer 3). Ці формати відрізняються насамперед своїми підходами до стиснення цифрового представлення аудіосигналу.

Перехід від неперервного сигналу до його цифрового представлення передбачає початкове захоплення аналогового сигналу мікрофоном, перетворення звукових хвиль в електричні сигнали. Згодом аналого-цифровий перетворювач оцифровує електричний сигнал, створюючи цифрове представлення за допомогою семпліювання.

Дискретизація, що означає процес кількісного визначення значень неперервного сигналу через фіксовані інтервали часу, породжує дискретну

форму сигналу з кінцевим набором значень сигналу через рівномірні інтервали часу. Частота дискретизації, що вимірюється в герцах (Гц), позначає частоту відліків, взятих за секунду, таким чином визначаючи часову роздільну здатність. Варто зазначити, що стандартна частота дискретизації аудіо CD-якості становить 44 100 Гц, тобто частоту 44 100 відліків на секунду. І навпаки, аудіо високої роздільної здатності може похвалитися частотою дискретизації 192 000 Гц або 192 кГц. Для навчання мовних моделей найчастіше використовують частоту дискретизації 16 000 Гц або 16 кГц.

Вибір частоти дискретизації в першу чергу диктується верхньою межею захоплених частот, відповідно до межі Найквіста, яка дорівнює половині частоти дискретизації. Враховуючи, що чутні частоти людської мови лежать нижче 8 кГц, достатньою є частота дискретизації 16 кГц. Використання більш високої частоти дискретизації не покращує захоплення інформації, але збільшує вимоги до обчислювальної обробки. І навпаки, недостатня частота дискретизації призводить до втрати інформації, що проявляється в приглушеній якості мови, записаної з частотою 8 кГц, яка не здатна захопити більш високі частоти.

Забезпечення однакової частоти дискретизації для всіх аудіоприкладів у наборах даних залишається ключовим для будь-якого завдання, пов'язаного з аудіо. Якщо ви використовуєте власні аудіодані для точного налаштування попередньо навчених моделей, вирівнювання частоти дискретизації даних з частотою дискретизації попередньо навченої моделі стає обов'язковим. Це вирівнювання визначає часовий інтервал між послідовними звуковими відліками, впливаючи на часову роздільну здатність аудіоданих.

Оскільки трансформаційні моделі, які розглядають аудіозадачі як послідовні дані, покладаються на механізми уваги для навчання, різна частота дискретизації між прикладами перешкоджає узагальненню моделі.

Передискретизація, частина попередньої обробки аудіоданих, виправляє цю розбіжність, вирівнюючи частоту дискретизації.

1.3 Амплітуда і розрядність

Хоча частота дискретизації визначає частоту отримання відліків, важливо розуміти природу значень, що містяться в кожному відліку.

Звук, що виникає внаслідок зміни тиску повітря на чутних частотах, знаходить кількісне вираження в амплітуді, що представляє рівень звукового тиску в будь-який момент часу, який зазвичай вимірюється в децибелах (дБ). Амплітуда корелює з нашим сприйняттям гучності. Наприклад, звичайний розмовний голос має амплітуду нижче 60 дБ, в той час як рок-концерт може досягати піку приблизно 125 дБ, кидаючи виклик людським слуховим можливостям.

У цифровому аудіо кожен семпл фіксує амплітуду звукової хвилі в певний момент часу. Глибина розрядності зразка диктує точність, з якою описується ця амплітуда. Вища розрядність відповідає кращому наближенню до вихідної безперервної звукової хвилі.

Найпоширенішими розрядностями в аудіо є 16- та 24-розрядна, що позначає рівні квантування, доступні під час перетворення з неперервного в дискретний звук: 65 536 кроків для 16-розрядного аудіо та вражаючі 16 777 216 кроків для 24-розрядного аудіо. Оскільки квантування передбачає округлення неперервних значень до дискретних, під час дискретизації виникає власний шум. Підвищена розрядність зменшує цей шум квантування. Однак шум квантування

16-бітового аудіо зазвичай непомітний, що робить вищу розрядність непотрібною.

Крім того, можна зустріти 32-бітне аудіо, яке використовує значення з плаваючою комою на противагу цілочисельним відлікам у 16-бітному та 24-бітному форматах. Подібно до 24-бітового аудіо, 32-розрядні семпли з плаваючою комою мають точність 24 біти. Ці вибірки з плаваючою комою дотримуються діапазону [-1.0, 1.0].

Оскільки моделі машинного навчання працюють з даними з плаваючою комою, перед використанням у навчанні моделі аудіо має бути перетворено у формат з плаваючою комою.

Подібно до неперервних аудіосигналів, амплітуда цифрового звуку зазвичай виражається в децибелах (дБ). Враховуючи логарифмічну природу людського слуху, де чутливість до звукових коливань вища при менших амплітудах, децибели полегшують інтерпретацію гучності звуку.

У той час як у реальному аудіо використовується 0 дБ як поріг для найтихішого чутного звуку, в цифровому аудіо 0 дБ є максимально можливою амплітудою, а всі інші амплітуди виражаються як від'ємні величини. Емпіричне правило стверджує, що кожне зменшення на -6 дБ означає зменшення амплітуди вдвічі, а звуки нижче -60 дБ взагалі непомітні без значного посилення гучності.

1.4 Представлення звуку у формі хвилі (waveform)

В аудіо аналізі часто зустрічається зображення звуку за допомогою візуалізацій, відомих як хвильові форми, що представляють звуковий сигнал у

часовій області шляхом побудови графіків значень зразків у часі. Таке графічне представлення прояснює зміни амплітуди звуку в часі.

Корисність візуалізації осцилограм поширюється на розпізнавання ключових характеристик аудіосигналу, включаючи часові патерни звукових подій, загальну інтенсивність сигналу та наявність аномалій (шуму) в аудіо.

У мові Python бібліотека librosa надає зручний набір інструментів для побудови графіків форми сигналу. Наприклад, зразок звуку з назвою «trumpet», доступний у бібліотеці, можна завантажити як кортеж, що містить часовий ряд звуку (array), і пов'язану з ним частоту дискретизації (sampling_rate). Використання функції waveshow() бібліотеки librosa надає візуалізацію форми хвилі цього звуку.

Після виконання, графік (рис.1.1) відображає амплітуду сигналу на осі у проти часу на осі x. Кожна точка на графіку відповідає дискретному значенню відліку, захопленому під час процесу дискретизації.

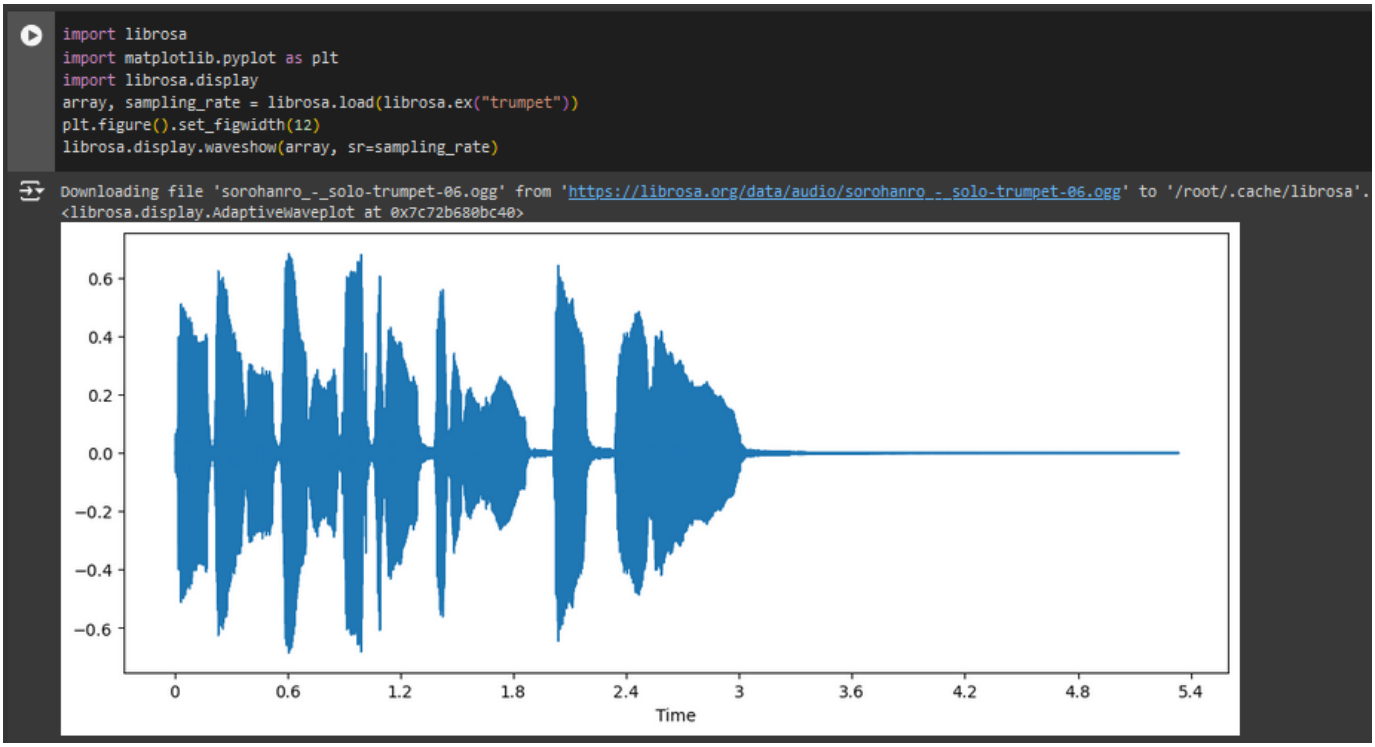


Рисунок 1.1 – виконання коду для відображення графіку форми хвилі звуку «trumpet»

Варто зазначити, що librosa за своєю суттю повертає звук у форматі з плаваючою комою, а значення амплітуди обмежені діапазоном [-1.0, 1.0]. Така нормалізація забезпечує однорідність і сумісність з подальшою аналітичною обробкою (URL: https://librosa.org/doc/latest/index.html).

1.5 Представлення частотного спектру

Альтернативний підхід до візуалізації аудіоданих передбачає побудову частотного спектру аудіосигналу, відомого як представлення в частотній області.

Це зображення, обчислене за допомогою дискретного перетворення Фур'є, виділяє складові частоти сигналу та їхні відповідні амплітуди.

У практичному використанні терміни «ШПФ» (швидке перетворення Фур'є) і «ДПФ» (дискретне перетворення Фур'є) часто використовуються як взаємозамінні. Така взаємозамінність пов'язана з тим, що ШПФ є найефективнішим методом для обчислення ДПФ на обчислювальних платформах.

Для побудови частотного спектру вищезгаданого звуку труби, ДПФ застосовано за допомогою функції rfft() NumPy. Хоча можливо візуалізувати спектр всього звуку, зосередження на певному сегменті дає більш дієві висновки. У цьому прикладі ДПФ обчислюється для перших 4096 відліків, що приблизно відповідає тривалості початкової зіграної ноти.

Побудований частотний спектр (рис. 1.2) ілюструє відносну силу різних частотних компонентів в аналізованому аудіосегменті. Зазвичай значення частоти відкладаються на осі x, часто в логарифмічному масштабі, а амплітуди - на осі y.

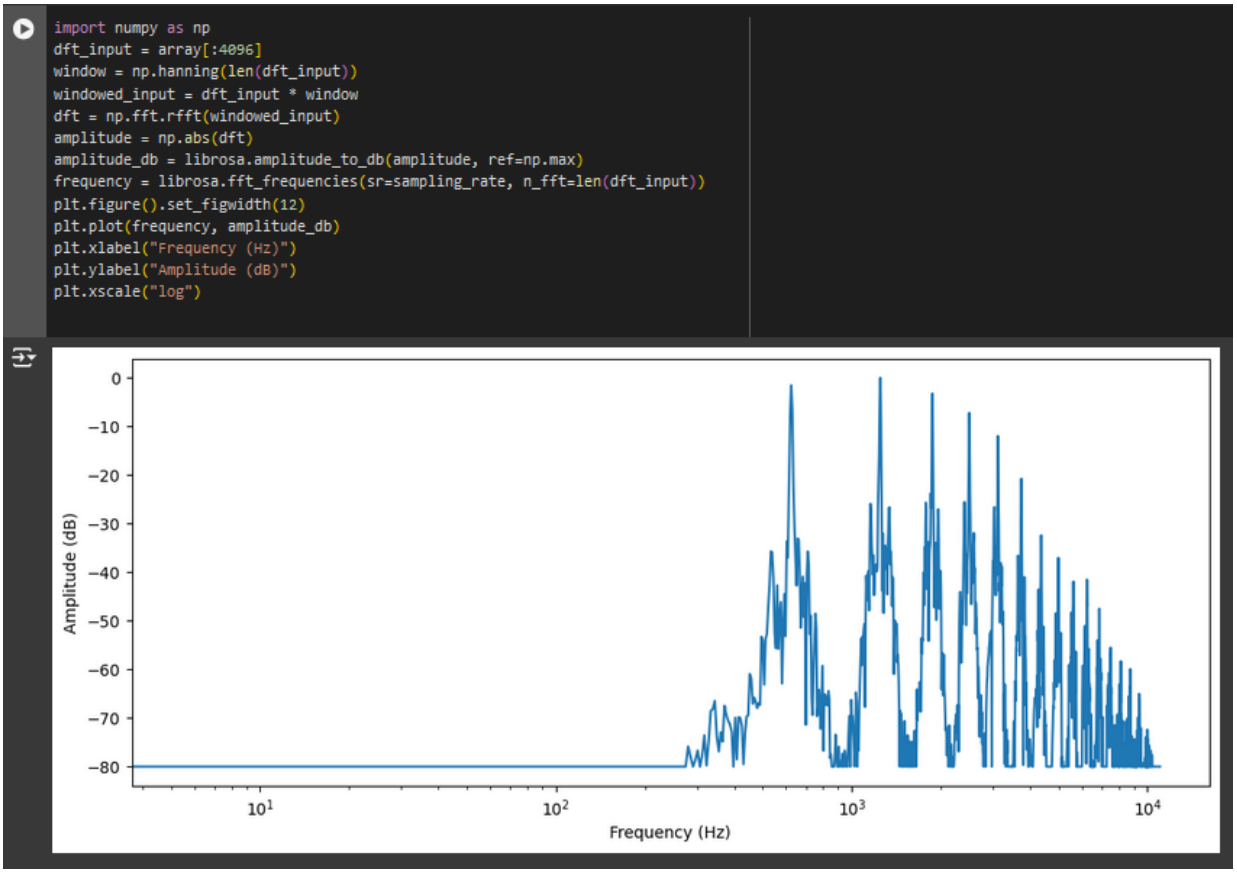


Рисунок 1.2 – виконання коду для відображення частотного спектру звуку «trumpet»

Піки частотного спектра відповідають гармонікам зіграної ноти, причому вищі гармоніки демонструють меншу амплітуду.

Після обчислення ДПФ результуючий масив складається з комплексних чисел, що включають як дійсні, так і уявні компоненти. Вилучення амплітудної інформації передбачає обчислення величини масиву ДПФ за допомогою функції np.abs(). Хоча фазовий спектр, який визначається кутом між дійсною та уявною складовими, дає додаткову інформацію, його часто ігнорують у контексті машинного навчання.

Для підвищення чіткості візуалізації значення амплітуди перетворюються в децибельні за допомогою librosa.amplitude_to_db(), що полегшує розпізнавання

більш тонких спектральних деталей. Крім того, деякі аналізи використовують спектр потужності, який кількісно оцінює енергію, а не амплітуду, що досягається зведенням значень амплітуди в квадрат для отримання спектра потужності.

Частотний спектр аудіосигналу містить ту саму інформацію, що і його форма - це просто два різні способи перегляду одних і тих самих даних (тут - перших 4096 відліків звуку труби). Якщо форма сигналу відображає амплітуду аудіосигналу в часі, то спектр візуалізує амплітуди окремих частот у фіксований момент часу.

1.6 Спектрограма

Для спостереження за зміною частот в аудіосигналі одного знімка спектра може виявитися недостатньо, особливо якщо сигнал містить кілька нот з різними частотами. Щоб вирішити цю проблему, обчислюється кілька ДПФ, кожен з яких охоплює невеликий проміжок часу, а отримані спектри об'єднуються в спектрограму.

Спектрограма окреслює часову еволюцію частотного вмісту в аудіосигналі, об'єднуючи інформацію про час, частоту та амплітуду в єдине графічне представлення.

Спектрограма є надзвичайно інформативним інструментом в аналізі звуку. Наприклад, у музичних записах вона дозволяє ідентифікувати та аналізувати різні інструменти та вокальні доріжки, оцінюючи їхній внесок у загальний слуховий ландшафт.

Аналогічно, в аналізі мовлення вона допомагає розрізняти окремі голосні звуки, кожен з яких характеризується своїм унікальним частотним профілем.

Для візуалізації спектрограми вищезгаданого звуку труби використовуються функції stft() та specshow() з бібліотеки librosa.

На зображеній спектрограмі (рис. 1.3) час відкладається на осі x, подібно до візуалізації форми сигналу, а частота - на осі y в герцах (Гц). Інтенсивність кольору передає амплітуду або потужність кожної частотної складової в будь-який момент часу, виражену в децибелах (дБ).

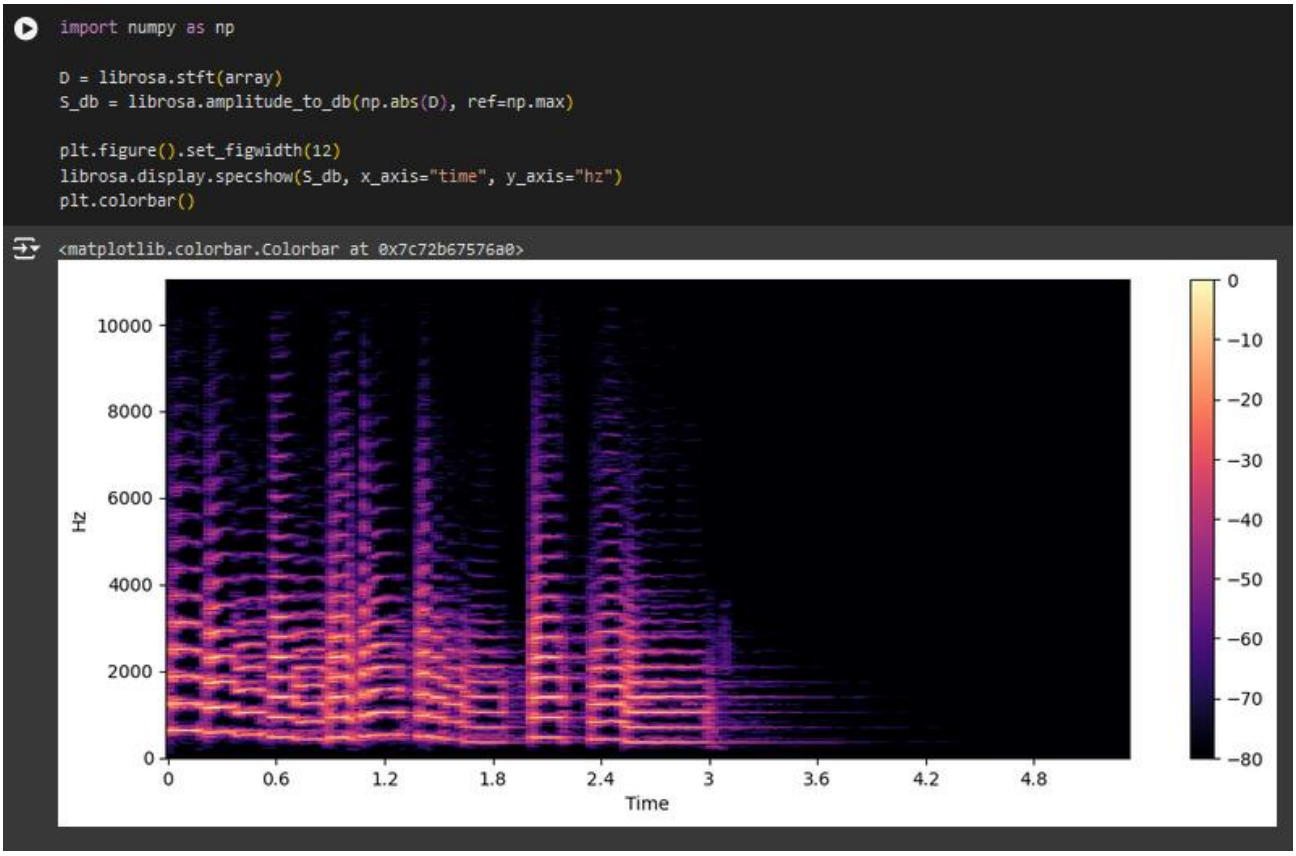


Рисунок 1.3 – виконання коду для відображення спектрограми звуку «trumpet»

Побудова спектрограми передбачає поділ аудіосигналу на короткі сегменти, зазвичай тривалістю кілька мілісекунд, з подальшим обчисленням

дискретного перетворення Фур'є кожного сегмента для отримання його частотного спектра. Потім ці спектри складаються вздовж часової осі для створення спектрограми. Кожен вертикальний зріз на зображенні відповідає окремому частотному спектру, якщо дивитися зверху. За замовчуванням librosa.stft() сегментує аудіосигнал на сегменти по 2048 відліків, досягаючи балансу між частотною і часовою роздільною здатністю (URL: https://huggingface.co/learn/audio-course/chapter0/introduction (дата звернення: 11.05.2024)).

Хоча спектрограма і форма сигналу пропонують різні погляди на ті самі дані, спектрограму можна повернути до початкової форми сигналу за допомогою оберненого короткочасного перетворення Фур'є. Однак це вимагає доступу до фазової інформації поряд з амплітудними даними. У сценаріях, коли спектрограму отримано з вихідних даних моделі машинного навчання, зазвичай доступна лише амплітудна інформація. У таких випадках для відновлення форми сигналу зі спектрограми використовуються методи фазової реконструкції, такі як класичний алгоритм Гріффіна-Ліма або вокодери на основі нейронних мереж.

Окрім простої візуалізації, спектрограми слугують важливими вхідними та вихідними даними для багатьох моделей машинного навчання, особливо тих, що зосереджені на обробці аудіо.

1.7 Мел-шкала

Мел-шкала (шкала мел, mel scale) - це перцептивна шкала висот звуків, які слухачі вважають рівними за відстанню один від одного. Вона названа від слова «melody», що вказує на те, що шкала базується на людському сприйнятті звуку, а

не на фізичних властивостях звукових хвиль. Шкала мел має на меті відповідати нелінійній чутливості людського вуха до різних частот. Люди більш чутливі до різниці у висоті звуку на низьких частотах, ніж на високих: нижче приблизно 1000 Гц людина може розрізняти менші відмінності у висоті звуку, тоді як вище 1000 Гц для такої ж зміни сприйняття потрібні більші відмінності.

Зв'язок між частотою (f) і мел-шкалою (м) часто апроксимується наступною формулою:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$$

Мел-частотні кепстральні коефіцієнти (MFCC) широко використовуються в системах розпізнавання мови, оскільки вони забезпечують компактне представлення спектра потужності аудіосигналів у спосіб, який добре узгоджується зі слуховим сприйняттям людини (URL: https://arxiv.org/abs/2008.03648).

Ці коефіцієнти виводяться шляхом попереднього перетворення аудіосигналу в шкалу Мела за допомогою банку трикутних фільтрів, розташованих відповідно до шкали Мела. Потім береться логарифм потужності на виході кожного фільтра, після чого застосовується дискретне косинусне перетворення (ДКП) для кореляції отриманих коефіцієнтів.

1.8 Мел-спектрограма

Мел-спектрограма є спеціалізованою формою спектрограми, яка широко використовується в обробці мовлення та різних програмах машинного навчання.

Хоча вона схожа на стандартну спектрограму в зображенні частотного вмісту аудіосигналу в часі, вона відрізняється від неї представленням частотної осі.

У звичайній спектрограмі частотна вісь відповідає лінійній шкалі, що вимірюється в герцах (Гц). Однак слухове сприйняття людини демонструє логарифмічну чутливість, коли нижчі частоти розрізняються гостріше, ніж вищі. Для апроксимації цієї нелінійної чутливості використовується шкала мела - шкала частот сприйняття.

Створення мел-спектрограми передбачає використання короткочасного перетворення Фур'є для сегментації звуку на короткі інтервали, таким чином отримуючи серію частотних спектрів, як у стандартній спектрограмі. Крім того, кожен спектр проходить перетворення за допомогою набору фільтрів, відомих як мел-фільтр, який відображає частоти за мел-шкалою.

Для зручного генерування мел-спектрограми використаємо librosa melspectrogram().

У наведеному прикладі n_mels - це кількість мел-смуг (mel bands), які потрібно згенерувати. Ці смуги окреслюють діапазони частот, які розділяють спектр на значущі для сприйняття компоненти, використовуючи фільтри, конфігурація і відстань між якими підібрані таким чином, щоб імітувати реакцію людського вуха на різні частоти (рис. 1.4).

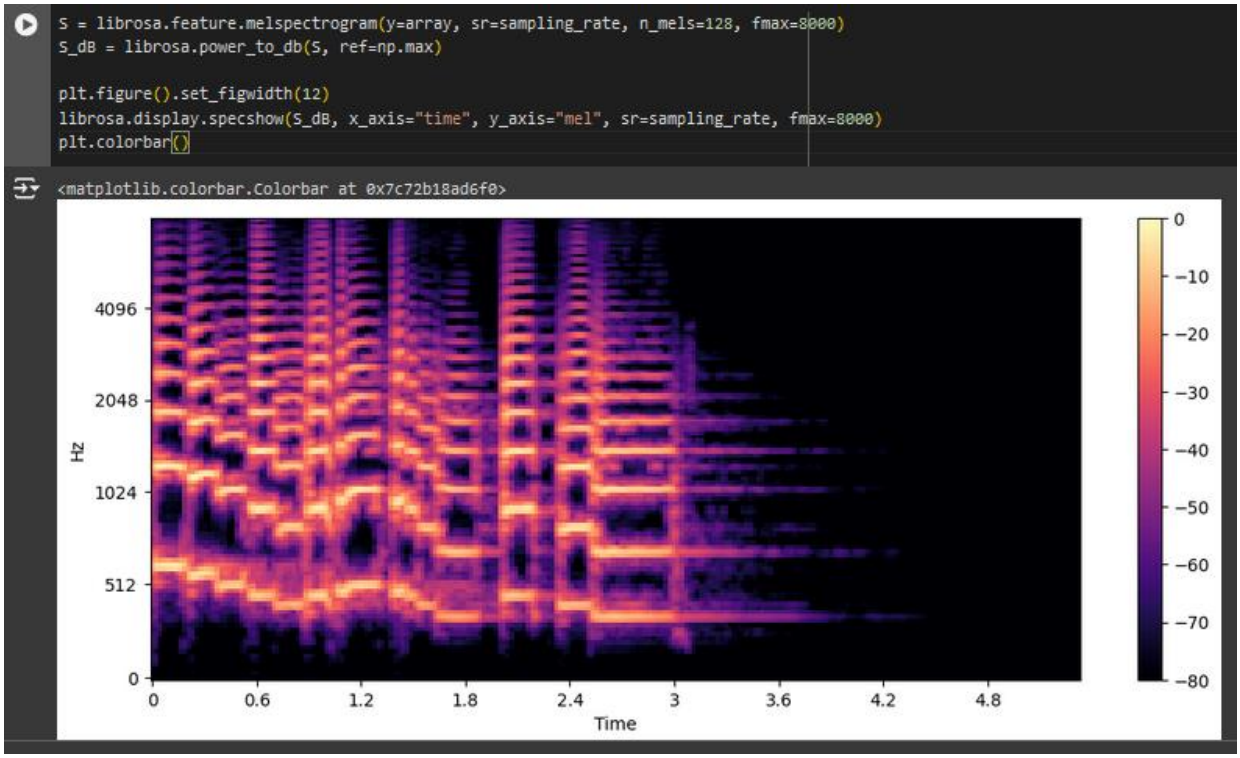


Рисунок 1.4 – виконання коду для відображення мел-спектрограми звуку «trumpet»

Загальноприйнятими значеннями для n_mels є 40 або 80. Крім того, fmax позначає найвищу частоту (в Гц), яка вважається релевантною для аналізу.

Подібно до стандартної спектрограми, прийнято представляти потужність частотних компонентів мела в децибелах. Ця домовленість, яку часто називають лог-мел спектрограмою, виникає через логарифмічну операцію, пов'язану з перетворенням інтенсивності мел-компонентів в децибели. У наведеному вище прикладі для такого перетворення використовується librosa.power_to_db(), оскільки librosa.feature.melspectrogram() за замовчуванням генерує спектрограму потужності (URL: https://huggingface.co/learn/audio-course/chapter0/introduction (дата звернення: 11.05.2024)).

Дійсно, існує варіативність у побудові мел-спектрограм, що зумовлена кількома факторами, включаючи вибір мел-шкали і тип використовуваної спектрограми.

Мел-шкала, критичний компонент розрахунку, може бути реалізована за допомогою різних методологій, зокрема, варіантів «htk» і «slaney», кожен з яких пропонує різні наближення до слухового сприйняття частоти людиною.

Крім того, хоча спектрограми потужності переважають через обчислювальну доцільність, амплітудні спектрограми є прийнятною альтернативою.

Варто зазначити, що перетворення в логарифмічну мел-спектрограму не завжди тягне за собою обчислення справжніх значень децибел, а може просто включати застосування логарифмічної функції до мел спектрограми. Ця тонкість підкреслює важливість забезпечення методологічної узгодженості при створенні мел-спектрограм, особливо при підготовці даних для моделей машинного навчання. Отже, ретельна перевірка параметрів і кроків обробки є обов'язковою для підтримки узгодженості вхідних даних моделі.

Створення мел-спектрограми - це операція з втратами, оскільки вона передбачає фільтрацію сигналу. Перетворення мел-спектрограми назад у форму хвилі сигналу складніше, ніж для звичайної спектрограми, оскільки це вимагає оцінки частот, які були відкинуті. Ось чому для отримання форми хвилі сигналу з мел-спектрограми потрібні моделі машинного навчання, такі як, наприклад, вокодер HiFiGAN.

Порівняно зі звичайною спектрограмою, мел-спектрограма може зафіксувати більш значущі для людського сприйняття особливості аудіосигналу, що робить її популярним вибором у таких завданнях, як розпізнавання мови, ідентифікація диктора, класифікація музичних жанрів та синтез голосу.

1.9 Мел-частотні кепстральні коефіцієнти

Мел-частотний кепстр (далі – МЧК)(cepstrum, від «spectrum»; у аналізі Фур’є – результат обчислення зворотного перетворення Фур'є логарифма спектра сигналу) – це представлення короткочасного спектра потужності звуку, засноване на лінійному косинус-перетворенні логарифмічного спектра потужності на нелінійну мел-шкалу частот.

Мел-частотні кепстральні коефіцієнти (Mel-Frequency Cepstral Coefficients, далі – МЧКК) це коефіцієнти, які в сукупності складають МЧК. Вони походять від типу кепстрального представлення аудіокліпу (нелінійного «спектру спектра»). Різниця між кепстром і мелчастотним кепстром полягає в тому, що в МЧК частотні смуги розташовані на однаковій відстані на мел-шкалі, що наближає реакцію слухової системи людини більш точно, ніж лінійно розташовані частотні смуги. Таке викривлення частоти може забезпечити краще представлення звуку, наприклад, при стисненні звуку, що потенційно може зменшити смугу пропускання та вимоги до зберігання аудіосигналів (URL: https://arxiv.org/abs/2008.03648).

1.10 Постановка задачі синтезу мовлення та клонування голосу

Основна задача синтезу мовлення полягає у конвертації друкованого тексту у аудіо. Технологія text to speech спрямована на створення мовлення, яке є зрозумілим і природно звучить. Синтез мовлення застосовується у багатьох

сферах, включаючи допоміжні технології для людей з вадами зору та мовлення, освітні інструменти, автоматизацію обслуговування клієнтів та вивчення мов.

Задача клонування голосу полягає у репродукції індивідуального голосу, включаючи такі характеристики як висота голосу, тон, акцент і стиль мовлення. Клонування голосу має великий потенціал, і наразі успішно використовується у сфері розваг.

Цілі синтезу мовлення та клонування голосу зосереджені на покращенні комунікації та доступності шляхом створення природного, зрозумілого та персоналізованого мовлення. Синтез мовлення спрямований на широке застосування і масштабування, в той час як клонування голосу зосереджене на точному відтворенні індивідуальних голосів.

Основні кроки для розв'язання задачі синтезу мовлення та клонування голосу виглядають наступним чином.

1. Збір та обробка даних: необхідно зібрати якісні і бажано великі набори аудіо та текстових даних. Аудіо має містити різноманітні фонетичні контексти, інтонації та стилі мовлення, а також не мати шумів. Для text to speech моделей важливо щоб кожен аудіофайл мав відповідний, точно транскрибований текст. Текстові дані необхідно нормалізувати (прибрати використання цифер у реченнях, наприклад), та записати у вигляді фонетичної транскрипції, співставивши її з аудіоданими.

2. Вибір архітектури моделі: наразі популярними є Tacotron 2, трансформери (HuBERT) та FastSpeech. Для клонування голосу, модель має підтримувати процес виділення ознак голосу (feature extraction) у вигляді мел-спектрограми.

3. Тренування моделі: слід визначити потрібні функції втрат, наприклад для передбачення мел-спектрограм у Tacotron 2 використовується середня квадратична похибка.

4. Тренування голосового кодувальника (вокодера): вокодер конвертує згенеровані моделю мел-спектрограми у форму хвилі, яку можна програти. Сучасні вокодери базуються на генеративних мережах і генерують звук високої якості.

5. Оптимізація для використання у реальному часі: такі процеси як обрізання моделі, квантування або використання спеціалізованого обладнання (наприклад відеокарти Nvidia з технологією CUDA)

Описані процеси детальніше розглянуті у наступних розділах даної роботи.

1.11 Висновки до розділу 1

Людський голос є винятковим і потужним інструментом комунікації, здатним передавати емоції та виражати індивідуальність. Можливість генерувати та клонувати голос завдяки нейронним мережам перетворилася на захоплюючу сферу досліджень. Детальний огляд процесів оцифрування аудіосигналів та аналізу отриманих даних, зокрема мел-спектрограм, дозволяє детальніше зрозуміти процес навчання та використання TTS та VC моделей. Розглянуто постанову задачі синтезу мовлення та конвертації голосу

2. ОГЛЯД ТЕХНОЛОГІЙ СИНТЕЗУ МОВЛЕННЯ ТА КЛОНУВАННЯ ГОЛОСУ

2.1 Обробка природної мови

Обробка природної мови (Natural Language Processing, далі – NLP) – це підгалузь штучного інтелекту та лінгвістики, що фокусується на взаємодії між комп'ютерами та людськими мовами. Основна мета NLP – дати можливість комп'ютерним системам розуміти, інтерпретувати та генерувати людську мову таким чином, щоб вона була змістовною та корисною. Це включає в себе різні завдання, які варіюються від простої обробки тексту до складного розуміння і створення людської мови (URL: https://arxiv.org/abs/2008.03648).

NLP включає у себе багато завдань, таких як класифікація текстів, машинний переклад, пошук інформації, генерація текстів, тощо. У контексті даної роботи, завдання NLP полягає у розпізнаванні та відтворенні мовлення.

2.2 Рекурентні нейронні мережі

Рекурентні нейронні мережі (далі – RNN) - це клас штучних нейронних мереж, призначених для обробки послідовностей даних, що робить їх дуже ефективними для задач, пов'язаних з часовими рядами даних, обробкою природної мови (далі – NLP) та інших послідовних даних. На відміну від традиційних нейронних мереж прямого поширення, RNN мають зв'язки, які

формують спрямовані цикли, що дозволяє інформації зберігатися на різних часових кроках послідовності (URL: https://medium.com/@prudhviraju.srivatsavaya/lstm-vs-gru-c1209b8ecb5a).

RNN пристосовані для обробки послідовних даних, підтримуючи прихований стан, який фіксує інформацію з попередніх часових кроків. Цей прихований стан оновлюється, коли мережа обробляє кожен елемент послідовності, що дозволяє їй «пам'ятати» минулу інформацію.

Прихований стан RNN є критично важливим компонентом, який допомагає підтримувати контекст послідовності. На кожному часовому кроці прихований стан оновлюється на основі поточних вхідних даних і попереднього прихованого стану.

RNN мають однакову вагу на всіх часових кроках послідовності. Такий розподіл ваг дозволяє мережі узагальнювати різні частини послідовності та зменшує кількість необхідних параметрів.

2.2.1 Довга короткочасна пам’ять

Мережі з довгою короткочасною пам'яттю (далі – LSTM) - це тип архітектури RNN, розроблений для подолання обмежень традиційних RNN, зокрема проблеми навчання довготривалих залежностей. Вони були представлені Сеппом Хохрайтером і Юргеном Шмідхубером у 1997 році і з тих пір стали однією з найбільш широко використовуваних моделей для послідовної обробки даних (URL : https://medium.com/@prudhviraju.srivatsavaya/lstm-vs-gru-c1209b8ecb5a)).

Комірки пам'яті в LSTM зберігають інформацію протягом тривалих періодів часу, допомагаючи мережі вивчати залежності, які охоплюють довгі послідовності. Кожна комірка LSTM містить три вентилі: вхідний вентиль, вентиль забування і вихідний вентиль, які регулюють потік інформації.

Комірки діють як конвеєр, що проходить через всю послідовність з незначними лінійними взаємодіями, дозволяючи мережі передавати інформацію вперед без змін, якщо тільки вона не буде явно змінена воротами.

LSTM розроблені для подолання проблеми зникаючого градієнта, що дозволяє їм зберігати і вивчати залежності на довших послідовностях порівняно з традиційними RNN. Механізми вентилювання (gating mechanisms) дозволяють LSTM вибірково запам'ятовувати або забувати інформацію, що підвищує їхню здатність обробляти послідовні дані з різною довжиною залежностей.

LSTM можуть обробляти часові залежності, властиві мовним даним і фіксувати контекст слів, що робить їх придатними для NLP.

2.2.2 Вентильні рекурентні вузли

Як і традиційні RNN, вентильні рекурентні вузли (Gated Recurrent Unit, далі – GRU) призначені для обробки послідовних даних. Вони мають рекурентні зв'язки, які дозволяють інформації зберігатися на часових кроках. На кожному часовому кроці GRU приймає вхідні дані та свій внутрішній прихований стан з попереднього часового кроку і виробляє вихідні дані та новий прихований стан. Ключова інновація GRU полягає в механізмах вентилів, які допомагають їм вибірково оновлювати і забувати інформацію. GRU зазвичай мають два входи:

а. Ворота оновлення (Update Gate): ці ворота визначають, скільки минулої інформації слід зберегти, а скільки нової інформації додати до прихованого стану.

б. Ворота скидання (Reset Gate): Ці ворота контролюють, яку частину минулої інформації слід забути.

Ворота в GRU використовують сигмоїдні функції активації для виведення значень від 0 до 1, що вказують на ступінь, до якого кожна з них повинна бути відкрита або закрита.GRU використовують прихований стан-кандидат, який є сумішшю попереднього прихованого стану і нового вхідного сигналу. Цей стан-кандидат обчислюється на основі воріт скидання і використовується для обчислення воріт оновлення.

Кінцевий прихований стан - це комбінація попереднього прихованого стану і прихованого стану-кандидата, зважена за допомогою воріт оновлення.

Це дозволяє моделі дізнатися, скільки нової інформації слід включити, зберігаючи при цьому релевантну інформацію з попередніх часових кроків. GRU стали популярними в задачах моделювання послідовностей, таких як моделювання мови, машинний переклад, розпізнавання мови тощо. Вони обчислювально ефективніші за, і при цьому здатні вловлювати довгострокові залежності в послідовних даних.

(URL:https://medium.com/@prudhviraju.srivatsavaya/lstm-vs-gru-c1209b8ecb5a, (дата звернення: 12.05.2024))

2.3 Генеративні змагальні мережі

Генеративні змагальні мережі (Generative Adversarial Networks, далі - GAN) – це клас фреймворків машинного навчання, розроблених Яном Гудфеллоу та його колегами у 2014 році. GAN складаються з двох нейронних мереж, генератора та дискримінатора, які навчаються одночасно в процесі змагальної конкуренції (URL: https://arxiv.org/abs/1406.2661).

Роль генератора полягає у створенні даних, які імітують реальні дані. Він приймає випадковий шум як вхідні дані і генерує екземпляри даних (наприклад, зображення, текст тощо).

Роль дискримінатора полягає в тому, щоб відрізнити справжні дані (з навчальної вибірки) від фальшивих. Він виводить ймовірність того, що даний екземпляр даних є справжнім або підробленим.

Генератор і дискримінатор навчаються в грі з нульовою сумою.

Завдання генератора полягає у створенні дані, які неможливо відрізнити від справжніх, щоб обманути дискримінатор, який класифікує екземпляри даних як справжні або фальшиві.

Функція втрат генератора вимірює, наскільки добре генератор обманює дискримінатор, в той час як функція втрат дискримінатора показує, наскільки добре дискримінатор може відрізнити справжні дані від підроблених.

GAN широко використовуються для завдань перекладу зображень, синтезу тексту та синтезу аудіо.

2.3.1 Генеративні змагальні мережі високої точності

Генеративні змагальні мережі високої точності (High-Fidelity Generative Adversarial Networks, далі - HiFi-GAN) - це нейромережева модель, спеціально розроблена для високоякісного синтезу мови в режимі реального часу.

HiFi-GAN виступає у якості кодувальника голосу (voice encoder), створюючи звук, максимально наближений до людської мови, з високою чіткістю і природністю.

Модель розроблена для ефективної генерації звуку, що дозволяє використовувати її в реальному часі, наприклад, для голосових помічників і синтезу живого мовлення. Як і інші GAN, HiFi-GAN використовує архітектуру генератор-дискримінатор. Генератор створює форми звукових хвиль, а дискримінатор оцінює реалістичність цих форм.

Генератор у HiFi-GAN призначений для перетворення вхідних даних (наприклад, мел-спектрограми) у сирі звукові сигнали, використовуючи серію транспонованих згорткових шарів для поступового підвищення частоти дискретизації вхідних сигналів, перетворюючи їх на детальні аудіосигнали.

HiFi-GAN використовує декілька дискримінаторів, кожен з яких працює в різних масштабах і з різною роздільною здатністю. Такий підхід з використанням декількох дискримінаторів допомагає моделі вловлювати як дрібні деталі, так і глобальні патерни в аудіосигналі. Зазвичай застосовуються три типи дискримінаторів: багатоперіодний, багатомасштабний і дискримінатор мела-спектрограми, кожен з яких фокусується на різних аспектах форми звукового сигналу. (URL: https://arxiv.org/abs/2010.05646 (дата звернення: 20 травня 2024)).

HiFi-GAN можна використовувати для різних завдань синтезу мовлення, включаючи перетворення тексту в мовлення, клонування голосу та покращення мовлення. HiFi-GAN часто використовується в системах TTS для перетворення тексту в природне мовлення, корисне для віртуальних асистентів, аудіокниг та інструментів доступності. Можливості моделі в режимі реального часу роблять її

ідеальною для інтерактивних додатків, таких як голосові асистенти та боти для обслуговування клієнтів.

HiFi-GAN виявив значний вплив на сферу синтезу мови, розширивши межі можливого за допомогою моделей на основі GAN.

2.4 Варіаційний автокодувальник

Варіаційний автокодувальник (Variational Autoencoder, далі – VAE) - це тип генеративної моделі, яка навчається кодувати дані в латентний простір, а потім декодувати їх назад, щоб відновити вихідні дані. Вона поєднує в собі принципи нейронних мереж та імовірнісних графічних моделей для генерації нових точок даних, подібних до навчальних даних.

VAE кодують вхідні дані в безперервний латентний простір, де кожна точка цього простору є потенційною точкою даних. Цей латентний простір може бути дискретизований для генерації нових даних. VAE використовують варіаційне виведення для апроксимації апостеріорного розподілу латентних змінних на основі даних, що дає змогу проводити ефективне навчання та вибірку. (URL: https://arxiv.org/abs/1906.02691 (дата звернення: 14 травня 2024)).

VAE складається з:

1) кодувальника (мережа виводу): відображає вхідні дані x у латентний простір z. Замість відображення в одну точку, кодувальник виводить параметри розподілу ймовірностей (як правило, гаусівського), що характеризується середнім значенням μ і стандартним відхиленням σ.

2) декодувальника (генеративна мережа): бере вибірку з латентного розподілу z і реконструює вхідні дані x. Декодувальник вчиться генерувати дані, які максимально наближені до початкових вхідних даних.

Функція втрат VAE поєднує в собі:

1) втрати при реконструкції: вимірює, наскільки добре декодовані дані відповідають початковим вхідним даним. Зазвичай це від'ємна логарифмічна вірогідність даних з урахуванням латентних змінних.

2) розбіжність Кульбака-Лейблера: забезпечує близькість вивченого латентного розподілу до попереднього розподілу (зазвичай це стандартний нормальний розподіл). Це впорядковує латентний простір, щоб запобігти надмірному припасуванню та забезпечує гладкість.

VAE забезпечують спосіб вивчення складних розподілів даних. Вони пропонують гладкий і безперервний латентний простір, що дозволяє проводити інтерполяцію та дослідження, а імовірнісна природа моделі дозволяє кількісно оцінити невизначеність.

В той же час, процес тренування зазвичай набагато складніший ніж у GAN, а згенеровані семпли можуть бути менш детальні.

Варіаційні автокодери є потужним інструментом генеративного моделювання, що забезпечує надійну основу для навчання та вибірки зі складних розподілів даних.

2.5 TTS моделі

Моделі TTS (text to speech - перетворення тексту в мовлення) стали ключовими компонентами систем обробки природної мови, що дозволяють

перетворювати текстове введення в синтезоване мовлення. Останніми роками ці моделі зазнали значного розвитку завдяки появі методів глибокого навчання, зокрема нейромережевих архітектур. Рисунок 2.1 демонструє архітектуру моделі Tacotron2 від Nvidia.

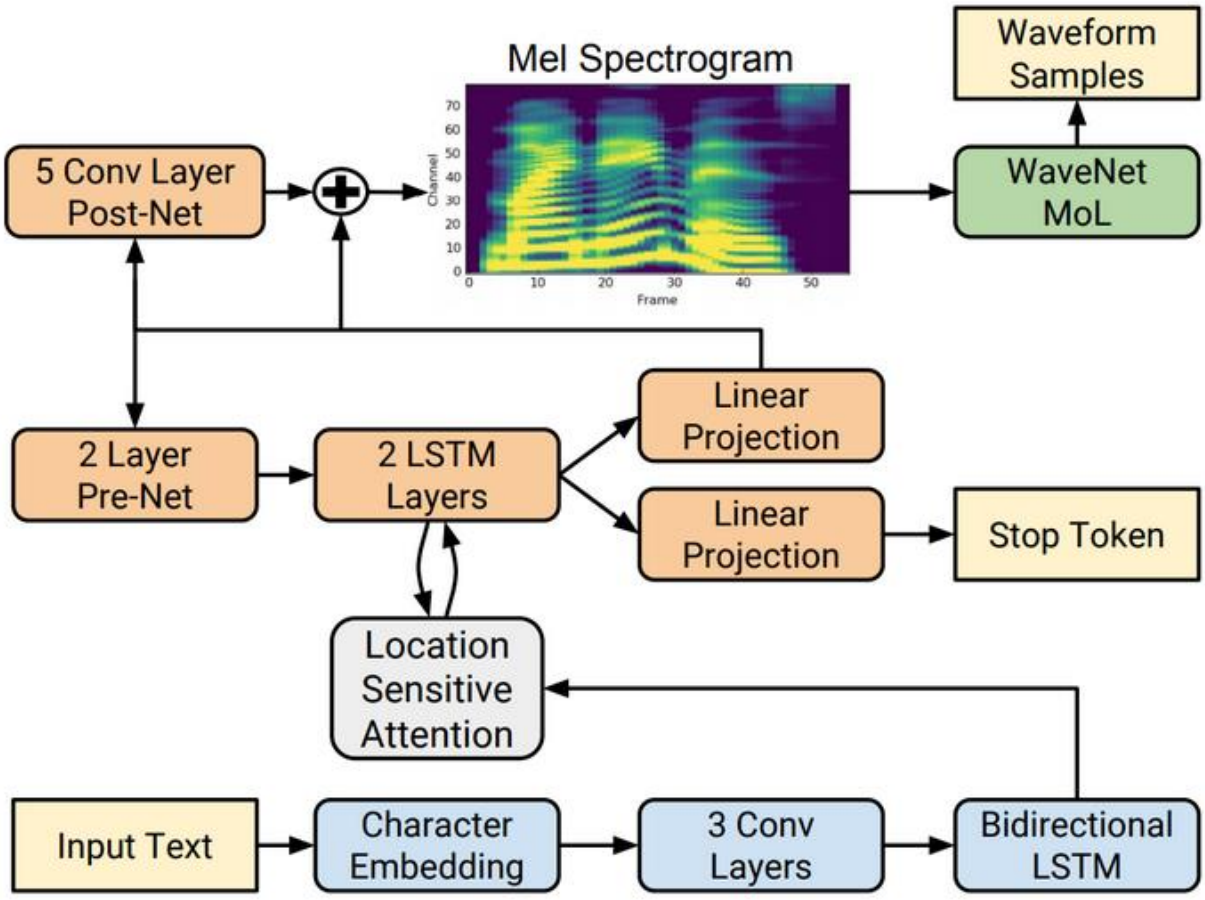


Рисунок 2.1 Діаграма архітектури TTS моделі Tacotron2 (URL:https://pytorch.org/hub/nvidia_deeplearningexamples_tacotron2/)

Одним з основних підходів у моделюванні TTS є використання нейромережевих архітектур, зокрема RNN. Моделі на основі RNN, такі як LSTM і GRU, здатні вловлювати часові залежності в послідовних даних, що робить їх добре придатними для задач, пов'язаних з генерацією тексту.

Аналогічно, CNN досягають успіху у вилученні ієрархічних ознак з вхідних даних, що дозволяє їм вивчати складні патерни в текстовій інформації.

Крім того, еволюція механізмів уваги покращила TTS, дозволивши моделі фокусуватися на релевантних частинах вхідного тексту під час генерації мовлення. Такі механізми, як архітектура Transformer, продемонстрували чудову продуктивність у різних завданнях обробки природної мови, включаючи TTS. Звертаючи увагу на релевантні слова та фрази у вхідному тексті, ці моделі покращують зв'язність та природність синтезованого мовлення.

Включення GAN у моделювання TTS додало елемент змагального навчання, коли мережа-генератор генерує синтезовані зразки мови, а мережа-дискримінатор оцінює їхню автентичність. Ця парадигма змагального навчання сприяє створенню більш реалістичного і точного мовлення, заохочуючи генератор видавати результати, які неможливо відрізнити від справжнього людського мовлення.

Іншим ключовим аспектом TTS-моделювання є використання великих наборів даних для навчання. Датасети які містять різноманітний лінгвістичний контент, дозволяють моделям TTS вловлювати нюанси вимови, інтонації та просодії в різних мовах і діалектах. Крім того, наявність багатомовних наборів даних полегшує розробку систем TTS, здатних синтезувати мовлення кількома мовами, задовольняючи потреби ширшого кола користувачів.

Останніми роками в дослідженнях TTS набула поширення концепція трансферного навчання, коли попередньо навчені моделі допрацьовуються на даних конкретної галузі для адаптації до конкретних завдань або додатків. Трансферне навчання не лише прискорює процес навчання, але й розширює можливості узагальнення моделей TTS, дозволяючи їм ефективно працювати в різноманітних реальних сценаріях.

Інтеграція експресивних та емоційних аспектів у моделі TTS привертає значну увагу, особливо в додатках, де передача почуттів або настрою за допомогою синтезованого мовлення має важливе значення. Такі методи, як вбудовування просодії та передача стилю, дозволяють системам TTS модулювати висоту тону, ритм та інші акустичні атрибути, щоб надати синтезованому мовленню бажаних емоційних характеристик.

2.5.1 XTTSv2

XTTSv2, вдосконалена TTS, розроблена Coqui.ai, являє собою значне оновлення порівняно зі своєю попередницею, XTTSv1. Ця модель призначена для створення виразного мовлення з тексту.

Одним з найпомітніших досягнень XTTSv2 є підтримка 17 мов, включаючи нові мови, такі як угорська та корейська. Така широка багатомовність робить модель надзвичайно універсальною для різних застосувань у різних мовних контекстах. Крім того, XTTSv2 має вдосконалену обробку голосу за допомогою моделі Perceiver, яка вводить мел-спектрограму і виводить латентні вектори, що представляють характеристики голосу диктора. Цей метод забезпечує більш точне і послідовне клонування голосу, захоплюючи нюанси різних дикторів більш ефективно, ніж простіші підходи, що використовувалися в попередніх моделях, таких як Tortoise або Vall-E (URL: https://arxiv.org/abs/1906.02691 (дата звернення: 14.05.2024)).

Архітектура моделі включає магістраль GPT-2, яка передбачає аудіо маркери, обчислені за допомогою попередньо навченої моделі Discrete VAE. Ця структура була вдосконалена, щоб значно зменшити затримку виведення,

досягнувши менш ніж 150 мілісекунд затримки потокового передавання на графічних процесорах споживчого класу. Такі покращення роблять XTTSv2 одним з найшвидших рішень з відкритим вихідним кодом для додатків реального часу (URL: http://erogol.com/2023/11/11/xtts-v2-release-and-notes (дата звернення: 15.05.2024)).

Використання HifiGAN для остаточного обчислення аудіосигналу сприяє чудовій якості звуку і просодії моделі, забезпечуючи природну і чітку генерацію мовлення.

Ще однією важливою особливістю XTTSv2 є його здатність виконувати клонування голосу за допомогою лише шестисекундного аудіокліпу. Така ефективність усуває необхідність у великих навчальних даних і забезпечує швидку та легку адаптацію до нових голосів. Крім того, модель може переносити емоції та стилі з еталонного аудіо, посилюючи виразність синтезованого мовлення.

2.6 VC моделі

Моделі Voice Cloning (клонування голосу) - це нове досягнення в галузі штучного інтелекту та обробки природної мови, що дозволяє відтворювати людський голос з надзвичайною точністю. Розробка цього виду моделей активно розпочалася після успіху WaveNet.

Однією з основних методологій клонування голосу є використання методів глибокого навчання, зокрема архітектур генеративного моделювання, таких як GAN і VAE. Ці моделі навчаються зіставляти вхідні спектрограми або

лінгвістичні ознаки з відповідними акустичними характеристиками, ефективно передаючи нюанси голосу диктора. Навчаючись на великих масивах даних, що містять аудіозаписи цільового диктора, моделі клонування голосу можуть імітувати вокальні характеристики диктора, включаючи висоту, тембр і просодію.

Крім того, розробка архітектур нейронних мереж з механізмами уваги розширила можливості моделей клонування голосу зосереджуватися на важливих особливостях вхідного аудіо, тим самим покращуючи якість і достовірність клонованого голосу на виході. Механізми уваги дозволяють моделі вибірково звертати увагу на відповідні часові та спектральні особливості, що сприяє більш точному відтворенню голосу диктора.

Вбудовуючи інформацію про диктора в архітектуру моделі, системи клонування голосу можуть генерувати персоналізовані голосові репліки, які дуже схожі на голос цільового диктора, навіть якщо вони навчаються на обмеженій кількості даних (наприклад 6 секунд у випадку XTTS-V2)

Крім того, для клонування голосу застосовуються такі методи, як transfer learning (навчання з перенесенням) і few-shot learning (навчання з мінімальної бази), що дозволяє адаптувати попередньо навчені моделі до нових дикторів або доменів з мінімальними додатковими навчальними даними. Це робить клонування голосу більш доступним і адаптованим до різних сценаріїв.

2.6.1 RVC моделі

Retrieval-based Voice Conversion (клонування голосу на основі пошуку) - це метод синтезу голосу, який передбачає пошук і модифікацію наявних зразків

мовлення. Цей метод використовує попередньо записані аудіодані для створення нових мовних результатів, які точно імітують голос цільового диктора.

Першим кроком у створенні RVC є збір набору аудіозаписів цільового диктора. На відміну від традиційних методів клонування голосу, які значною мірою покладаються на великі та якісні набори даних цільового голосу, моделі на основі пошуку дозволяють досягти високої точності клонування голосу за допомогою меншої кількості даних і більш ефективної обробки. Втім, найкращі результати досягаються якщо набір даних охоплює широкий спектр фонетичного змісту, стилів мовлення та емоційних виразів, щоб відобразити повну варіативність голосу диктора.

Кількість необхідного аудіо у датасеті залежить від декількох чинників, зокрема якості мовлення диктора, якості попередньо тренованої моделі, тощо. Мова попередньо тренованої моделі мало впливає на якість кінцевої моделі, максимум спричиняючи упередженість до якоїсь мови.

Наприклад, модель голосу українця тренована на основі англійської моделі може мати англійський акцент у згенерованому аудіо, або погано вимовляти фонетику, специфічну для нашої мови. Від цього ефекту можна позбутися із більшими та якіснішими датасетами, а також довшими тренуваннями. Якщо набір даних містить сильні шуми, його можна видалити за допомогою вокалу/супроводу, відокремивши та реверберації, використовуючи моделі з проекту Ultimate Vocal Remover, наприклад HP2-all-vocals.

Під час процесу клонування система знаходить сегменти мовлення, які максимально відповідають бажаним фонетичним і просодичним характеристикам цільового висловлювання. Це передбачає пошук у базі даних попередньо записаних зразків для знаходження найближчих збігів за такими критеріями, як послідовність фонем, висота та тривалість мовлення.

Після того, як відповідні сегменти знайдено, вони об'єднуються, щоб сформувати бажаний мовний результат. Особлива увага приділяється межам між сегментами, щоб забезпечити плавні переходи та уникнути артефактів, таких як клацання або неприродні розриви. Для досягнення безшовної конкатенації часто застосовуються такі методи, як накладання та згладжування сигналу.

Для подальшого покращення природності виконується просодичне коригування. Просодичні характеристики, такі як інтонація, наголос і ритм, можуть бути відрегульовані. Цей крок гарантує, що синтезована мова звучить зв'язно і природно, відображаючи задуману експресію та емоції (URL: https://qiita.com/nadare/items/306521c6010bf3efb115 (дата звернення: 20 травня 2024)).

Додаткові методи обробки сигналу, такі як зменшення шуму, еквалізація та стиснення динамічного діапазону, використовуються щоб гарантувати, що кінцевий результат буде чистим і приємним для прослуховування.

Розглянемо переваги RVC.

1. Висока точність: оскільки метод ґрунтується на реальних записаних зразках цільового диктора, отриманий голосовий клон може досягти дуже високого рівня точності, наближаючись до природного тембру і нюансів оригінального диктора.
2. Природність: завдяки використанню реальних мовних сегментів, клонування голосу на основі пошуку часто дає більш природне звучання порівняно з чисто генеративними моделями, які можуть мати труднощі з передачею тонких просодичних варіацій.
3. Ефективність: методи на основі пошуку можуть бути ефективними з точки зору обчислень, особливо коли процес пошуку оптимізовано, а набір даних добре організовано. Це робить їх придатними для застосувань, що вимагають клонування голосу в реальному часі.

4. Використання ресурсів: для застосування моделі у простому перетворенні аудіо потребує бюджетного процесору випуску 2017го року (URL: https://docs.aihub.wtf (дата звернення: 15.05.2024)) (втім, при тестуванні, було визначено, що процесор Intel i5-6600k 2015го року теж підходить).
5. Непаралельне перетворення голосу: RVC не потребує не вимагають парних або вирівняних висловлювань від диктора-джерела та диктора-мішені. Навчальні дані для диктора-джерела та диктора-мішені можуть бути повністю окремими та не вирівняними.

Також варто вказати недоліки RVC.

1. Залежність від pretrain (попередньо тренованих) моделей: якість і універсальність клонованого голосу значною мірою залежать від обсягу та якості записаного набору даних.
2. Граничні артефакти: забезпечення плавних переходів між об'єднаними сегментами може бути складним завданням, і можуть виникати артефакти, якщо отримані сегменти не ідеально відповідають бажаному фонетичному та просодичному контексту.
3. Масштабованість: зі збільшенням розміру набору даних ефективний пошук і вилучення сегментів, що найкраще збігаються, може стати складним завданням, що вимагає складних алгоритмів індексування та вилучення.
4. Специфіка використання системних ресурсів: ефективне тренування можливе лише на графічних процесорах (GPU) 20XX від Nvidia через використання технології CUDA. Рекомендовано виконувати тренування у хмарному середовищі (Colab\Paperspace), що або займає досить багато часу, або потребує придбання просунутих планів використання. Клонування голосу у реальному часі також сильно навантажує GPU/CPU.

RVC на даний момент можна використовувати для наступних цілей.

1. Персоналізовані голосові помічники: Клонування голосу на основі RVC можна використовувати для створення персоналізованих голосових помічників, які розмовлятимуть голосом користувача або голосом обраного персонажа.
2. Створення контенту: У медіа-виробництві така модель може бути використана для створення нових діалогів голосом акторів, які недоступні для запису.

Таким чином, клонування голосу на основі пошуку використовує наявні аудіозаписи для створення високоякісної, природної мови. Хоча це дає значні переваги з точки зору якості та природності, воно також створює проблеми, пов'язані із залежністю від даних та масштабованістю. Цей підхід особливо цінний у додатках, де високоякісний синтез голосу має вирішальне значення. Наразі, ця модель використовується здебільшого у розважальних цілях, проте безсумнівно має більш серйозний потенціал. Важливими питаннями також є етичність використання таких моделей, потенційний ризик застосування їх шахраями, та правові питання, пов’язані з використанням чийогось голосу без їх дозволу чи відома.

2.7 Висновки до розділу 2

Найновітнішими методами виконання задачі синтезу мовлення та клонування голосу є використання моделей RVC або xTTS, які є покращеними версіями попередників, таких як VITS. Можливості розробки моделей такого типу збільшуються разом із загальними розробками у сфері нейронних мереж.

Використання технологій клонування голосу є важливим правовим питанням, адже поки що безпечність цих технологій базується лише на власній відповідальності користувача.

3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ ТА АНАЛІЗ РЕЗУЛЬТАТІВ ЕКСПЕРИМЕНТУ

3.1 Вибір програмних засобів

Наступні підрозділи призначено для лістингу і короткого ознайомлення із задіяними технологіями.

3.1.1 Python

Python слугує потужним та адаптивним інструментом для побудови моделей машинного навчання. Використання Python для створення нейронних мереж має безліч переваг, зокрема:

1) простота: Python може похвалитися нескладним і зрозумілим синтаксисом, що полегшує розробку і розуміння коду;
2) різноманітна екосистема: Python містить широкий спектр бібліотек та фреймворків, пристосованих для задач машинного навчання;
3) прискорена розробка: Python - інтерпретована мова, що дозволяє швидко створювати прототипи та експериментувати з різними моделями; крім того,

Python є дуже розповсюдженим, із доступними для розробки великими клауд-сервісами, такими як Colab та Kaggle.

3.1.2 Librosa

Librosa - це бібліотека Python для аналізу аудіо та музики. Вона надає будівельні блоки, необхідні для створення систем пошуку музичної інформації (MIR) (URL: https://librosa.org/doc/latest/index.html). Librosa широко використовується в дослідницькій спільноті для дослідження аудіо-даних та обробки сигналів.

Librosa може завантажувати аудіофайли у різних форматах (наприклад, WAV, MP3 тощо) і перетворювати їх у безструктурний масив. Вона також може зберігати безструктурні масиви назад у аудіофайли.

Бібліотека надає інструменти для виконання таких операцій, як передискретизація, мікшування, розтягування в часі та зсув висоти тону. Вона також пропонує функції для обчислення перетворень Фур'є, перетворень постійної добротності та мел-спектрограм. Також, використання Librosa дає доступ до утиліт для візуалізації форм сигналів, спектрограм та інших характеристик звуку, що є корисним для аналізу та налагодження (URL: https://librosa.org/doc/latest/index.html).

Librosa містить функції для підготовки даних для моделей машинного навчання, такі як розділення даних на навчальні та тестові набори, перемішування та нормалізація ознак.

3.1.3 Tensorflow

TensorFlow - це фреймворк машинного навчання з відкритим вихідним кодом, розроблений компанією Google. Він широко використовується для різних завдань, таких як класифікація, регресія, кластеризація та нейронні мережі. TensorFlow надає комплексну екосистему інструментів, бібліотек та ресурсів спільноти, які полегшують створення та розгортання моделей машинного навчання. (URL: https://www.tensorflow.org/tutorials).

За своєю суттю TensorFlow представляє обчислення у вигляді графу, вузли якого - математичні операції, а ребра - потік даних між операціями. Цей графовий підхід дозволяє ефективно виконувати обчислення на різних апаратних платформах.

TensorFlow пропонує високорівневі API, такі як Keras, для легкої побудови та навчання нейронних мереж, а також низькорівневі API, які забезпечують більшу гнучкість та контроль над архітектурою моделі та процесом навчання.

3.1.3 TensorBoard

TensorBoard - це інструментарій візуалізації для TensorFlow, який дозволяє користувачам контролювати та налагоджувати моделі машинного навчання.

1. Візуалізація графіків: TensorBoard дозволяє візуалізувати обчислювальний графік моделі TensorFlow. Ця візуалізація допомагає зрозуміти архітектуру моделі, включаючи шари, операції та зв'язки між різними компонентами.
2. Метрики навчання: Під час навчання моделі TensorBoard може відображати різні метрики, такі як втрати, точність та інші користувацькі метрики з плином часу. Ці візуалізації допомагають контролювати процес навчання та виявляти будь-які проблеми, такі як надмірне або недостатнє тренування.
3. Гістограми та розподіли: TensorBoard надає інструменти для візуалізації розподілів ваг, зсувів та інших тензорів у моделі. Це може бути корисно для налагодження та розуміння того, як ці значення змінюються під час навчання.

3.1.4 HuBERT

HuBERT - це попередньо навчена модель самоконтролю мовлення. HuBERT має на меті забезпечити універсальне та надійне представлення мовних сигналів, яке можна використовувати для різних подальших завдань в обробці мовлення та розумінні природної мови.

HuBERT попередньо навчається передбачати певні особливості вхідного мовного сигналу, не вимагаючи явних анотацій. Це дозволяє моделі використовувати великі обсяги немаркованих даних для попереднього навчання.

HuBERT базується на трансформаторній архітектурі, яка виявилася дуже успішною в задачах обробки природної мови. Трансформаторна архітектура

дозволяє моделі вловлювати довгострокові залежності в мовному сигналі та вивчати ієрархічні репрезентації.

HuBERT може приймати мультимодальний ввід, включаючи аудіо та текст. Це означає, що його можна навчати на завданнях, які включають як мовні, так і текстові дані, що робить його універсальним для різних застосувань, зокрема, синтез мовлення та розуміння природної мови.

HuBERT має відкритий вихідний код, що означає, що ваги попередньо навченої моделі та навчальний код є загальнодоступними. Це дозволяє легко використовувати, модифікувати і розширювати модель для власних застосувань. (URL: https://arxiv.org/abs/2106.07447 (дата звернення: 16 травня 2024)).

3.1.5 FFmpeg

FFmpeg - це потужний програмний проект з відкритим вихідним кодом, що складається з набору бібліотек та інструментів для обробки мультимедійних даних, зокрема аудіо- та відеофайлів.

FFmpeg може декодувати (тобто перетворювати зі стисненого формату в нестиснений) і кодувати (тобто перетворювати з нестисненого формату в стиснений) аудіо- та відеопотоки в широкому діапазоні форматів. Підтримує такі популярні кодеки, як H.264, H.265, AAC, MP3 та багато інших (URL: https://huggingface.co/learn/audio-course/chapter0/introduction).

FFmpeg зазвичай використовується для перекодування, яке передбачає перетворення аудіо або відео з одного кодека або формату в інший. Це корисно

для сумісності та зменшення розмірів. FFmpeg дозволяє конвертувати різні мультимедійні формати без необхідності перекодування аудіо- або відеопотоків.

FFmpeg надає багатий набір фільтрів та ефектів, які можна застосувати до аудіо- та відеопотоків. Ці фільтри дозволяють виконувати такі завдання, як масштабування, обрізання, зменшення шуму тощо.

3.1.6 PyTorch

PyTorch - це фреймворк машинного навчання з відкритим вихідним кодом, розроблений лабораторією штучного інтелекту Facebook (FAIR) (URL: https://pytorch.org/hub). Він широко використовується для різних завдань, таких як обробка природної мови, комп'ютерний зір, навчання з підкріпленням тощо. PyTorch надає гнучкий та динамічний обчислювальний граф, який дозволяє легко експериментувати та швидко створювати прототипи.

Однією з відмінних рис PyTorch є його динамічний обчислювальний граф, що означає, що граф будується «на льоту» по мірі виконання операцій. Ця динамічна природа забезпечує більшу гнучкість і полегшує налагодження порівняно зі статичними обчислювальними графами, що використовуються в інших фреймворках.

PyTorch надає багату бібліотеку тензорних операцій, подібну до NumPy, що дозволяє легко виконувати математичні операції над багатовимірними тензорами.

PyTorch пропонує автоматичне диференціювання за допомогою пакету autograd. Ця функція дозволяє обчислювати градієнти тензорів відносно деякого

скалярного значення, що має вирішальне значення для навчання нейронних мереж з використанням таких методів, як зворотне поширення.

PyTorch легко інтегрується з CUDA, що дозволяє проводити ефективні обчислення на графічних процесорах. Таке GPU-прискорення значно прискорює процес навчання моделей глибокого навчання, особливо для великих наборів даних і складних архітектур (URL: https://pytorch.org/hub).

Динамічна природа PyTorch робить його добре придатним для побудови динамічних нейромережевих архітектур, таких як рекурентні нейронні мережі (RNN) і трансформатори, де розмір або структура мережі може динамічно змінюватися в залежності від вхідних даних.

3.1.7 CUDA

CUDA, що розшифровується як Compute Unified Device Architecture, - це платформа паралельних обчислень та інтерфейс прикладного програмування (API), розроблена компанією NVIDIA. Вона дозволяє розробникам використовувати обчислювальну потужність графічних процесорів NVIDIA (GPU) для універсальних обчислювальних задач, що виходять за рамки рендерингу графіки.

CUDA надає модель програмування та середовище виконання, яке дозволяє розробникам писати програмне забезпечення, здатне виконувати масові паралельні обчислення на графічних процесорах NVIDIA.

CUDA дозволяє перекладати інтенсивні обчислення з CPU на GPU, надаючи API для керування пам'яттю на GPU, включаючи виділення та звільнення пам'яті, передачу даних між CPU та GPU, а також керування

ієрархіями пам'яті, такими як глобальна пам'ять, спільна пам'ять та текстурна пам'ять.

NVIDIA надає інструменти та бібліотеки, які допомагають розробникам оптимізувати свій CUDA код для підвищення продуктивності та ефективності. Сюди входять інструменти профілювання для виявлення слабких місць у продуктивності, бібліотеки для поширених завдань паралельних обчислень (наприклад, cuBLAS для лінійної алгебри та cuDNN для глибокого навчання (URL: https://developer.nvidia.com/cuda-python (дата звернення: 22.05.2025)).

3.1.8 Faiss

Faiss, (Facebook AI Similarity Search) - це ефективна і масштабована бібліотека, розроблена FAIR для пошуку схожості та кластеризації великих наборів даних. Вона в першу чергу призначена для обробки даних високої розмірності, що робить її добре придатною для таких завдань, як пошук найближчого сусіда, приблизний пошук найближчого сусіда і кластеризація в таких додатках, як системи рекомендацій, пошук зображень, обробка природної мови, тощо.

Faiss спеціалізується на ефективному індексуванні та пошуку векторів високої розмірності. Він використовує найсучасніші алгоритми та структури даних, оптимізовані для роботи з великомасштабними наборами даних з тисячами або навіть мільйонами вимірів.

Faiss розроблено для масштабування до великих наборів даних з мільйонами або навіть мільярдами векторів. Масштабованість досягається за

рахунок таких методів, як розбиття індексів, розподілені обчислення та оптимізовані структури даних, які мінімізують використання пам'яті та максимізують обчислювальну ефективність.

Faiss підтримує ефективні обчислення на графічних процесорах, використовуючи їх можливості паралельної обробки для прискорення операцій пошуку схожості. Таке GPU-прискорення значно прискорює час пошуку, особливо для великих наборів даних і векторів високої розмірності.

Faiss пропонує різноманітні структури індексів, оптимізовані для різних типів наборів даних і вимог до пошуку. До них відносяться інвертовані файлові індекси, індекси квантування продуктів, ієрархічні навігаційні графи малого світу, тощо.

Faiss - це програмне забезпечення з відкритим вихідним кодом, випущене під ліцензією Apache 2.0, що дозволяє користувачам вільно використовувати, модифікувати і поширювати його. Крім того, Faiss легко інтегрується з популярними фреймворками глибокого навчання, такими як PyTorch і TensorFlow (URL: https://faiss.ai/index.html (дата звернення: 24 травня 2024)).

3.2 Демонстрація використання xTTSv2

Як вказано у попередніх розділах, xTTSv2 є передовою моделю для клонування голосу і синтезу мовлення. Модель працює із few-shot training, і при використанні шести секунд аудіо може видати клонований голос. Як заявлено розробниками, Coqui, модель має кращу просодію при обробці довгих текстів ніж стандартні TTS, вміє передавати емоції, стилі мовлення, акценти, тощо.

У цілях короткої демонстрації роботи моделі без використання власних системних ресурсів, було використано відкритий простір на сайті Huggingface (URL:https://huggingface.co/spaces/coqui/xtts (дата звернення: 24.05.2024)), та записано коротке відео, яке показує принцип клонування голосу на англійській мові. (URL: https://youtu.be/QorGOu8RiJE (дата звернення: 02.06.2024))

У відео також продемонстровано фактор реального часу (real-time factor, далі – RTF), який використовується для оцінки продуктивності систем, що обробляють аудіо, мову або будь-яку іншу форму обробки даних у реальному часі. RTF – це відношення часу, необхідного для обробки даних, до фактичної тривалості даних, що обробляються.

У даному випадку, значення RTF = 0.25, що свідчить про виведення даних у 4 рази швидше за реальний час (faster than real-time inference), іншими словами, створення 10ти хвилин аудіо займає 2.5 хвилини. Значення RTF<1 особливо важливе в додатках, що вимагають швидкої генерації звуку, таких як системи зв'язку в реальному часі, голосові асистенти або автоматизовані системи обслуговування клієнтів.

Дана демонстрація також відображає як головний плюс, так і головний мінус XTTSv2, а саме – few-shot inference. Дійсно, для створення приблизного клону голосу, знадобилося усього 6 секунд аудіо, втім отримане аудіо є сильно «роботизованим».

Для покращення якості моделі, слід збільшувати розмір набору даних та проводити точне налаштування (fine-tune), втім значні вимоги до системи і відсутність зручності у користуванні, є значними перешкодами до її використання.

3.3 Тренування RVC моделі

Наступні підрозділи присвячені процесу створення клону власного голосу автора дипломної роботи: теоретичні відомості про процеси тренування та виведення, створення набору даних та демонстрація роботи.

3.3.1 Огляд тренування

RVC використовує дві моделі: HuBERT для вилучення ознак і GAN для генерації звуку. RVC створюється шляхом накладання інформації з людського голосу на більшу, попередньо треновану модель.

Існування цієї базової моделі означає, що не потрібно готувати велику кількість даних від людини, чий голос потрібно клонувати, достатньо лише 10 хвилин якісного аудіо. Архітектура моделі базується на Transformer.

RVC не використовує багато відеопам'яті (далі – VRAM) на графічному процесорі, лише близько 6 ГБ під час навчання і близько 2 ГБ під час виведення. Тому навчати модель можна навіть на машинах з 8 ГБ VRAM, а виконувати виведення – з 4 ГБ. Рисунки 3.1-3.2 демонструють необхідні системні характеристики для навчання та використання RVC моделей.

Training Inference

SPEC	REQUIREMENT
Operating System	Windows 10
GPU	NVIDIA RTX 2060ti
RAM	16GB
Storage	30 GB

Рисунок 3.1 – Необхідні характеристики системи для навчання RVC моделі (URL: https://docs.aihub.wtf (дата звернення: 15.05.2024))

Training Inference

SPEC	REQUIREMENT
Operating System	Windows 10
GPU	NVIDIA GTX 1050ti
RAM	8GB
Storage	6 GB

Рисунок 3.2 - Необхідні характеристики для виведення RVC моделі (URL: https://docs.aihub.wtf (дата звернення: 15.05.2024))

Вивчені ознаки голосу (feature extraction) представляються у вигляді розміщення у латентному просторі. На вхід HuBERT подається імпульсно-кодова модуляція (далі – PCM), а на виході - вектор ознак. Якщо на вхід подано PCM розміром (1, 156736), то результуючий вектор ознак від HuBERT має вигляд (1, 489, 256).

На вхід GAN подається вектор ознак, а на виході - PCM. У RVC є 4 варіанти GAN, залежно від версії (1 або 2) та наявності або відсутності параметру f0, що використовується для наведення на висоту.

Для вилучення f0 вхідного голосу доступні різні методи, включаючи вокодер WORLD та моделі на основі CNN, такі як Crepe, Mangio, Dio, тощо. Вибір методу вилучення f0 не обов'язково повинен збігатися з тим, що використовувався під час навчання, для даної моделі завжди застосовувався метод RMVPE (Robust Model for Vocal Pitch Estimation).

RVC використовує бібліотеку Faiss, для виконання середньозваженого усереднення з функціями вхідного голосу отриманими HuBERT, щоб легше відображати в синтезованому голосі такі нюанси, як звуки дихання, підвищуючи природність і виразність перетворення голосу.

При навчанні RVC можна використовувати версію v1 або v2. У v1 вихідні дані з HuBERT та вхідні дані до GAN є 256-вимірними, в той час як У v 2 вони 756-вимірні. V2 потенційно пропонує більш нюансовані перетворення голосу завдяки більш високій розмірності простору ознак голосу, отож у роботі використано цю версію.

3.3.1 Створення та обробка набору даних

Для подальшого використання у якості набору даних, було прийняте рішення створити запис власного голосу. RVC моделі швидко отримали популярність, оскільки потребують порівняно невеликої кількості даних і мають гарну якість навіть із коротким циклом тренуванням.

Оскільки RVC моделі працюють за принципом голос-голос (на відміну від текст-голос у клонувальних TTC), датасет має містити лише аудіо дані.

За допомогою програми Audacity, і конденсаторного мікрофона, було записано приблизно 35 хвилин читання художніх творів та наукових статей («Мені тринадцятий минало», «Contra spem spero», уривки з «Ловець у житі», біографії Нестора Махно та Денніса Клатта), з максимальним емоційним забарвленням, на яке був спроможній диктор. Результуючу аудіо доріжку було відредаговано таким чином, щоб видалити паузи, заминки та неправильну вимову, а також нормалізовано з налаштуванням -4 Дб.

Відредаговане аудіо має довжину у 33 хвилини та 55 секунд (рис. 3.3), та має гарну якість, знаходячись у «золотій середині» між рекомендованими 10-60 хвилин.

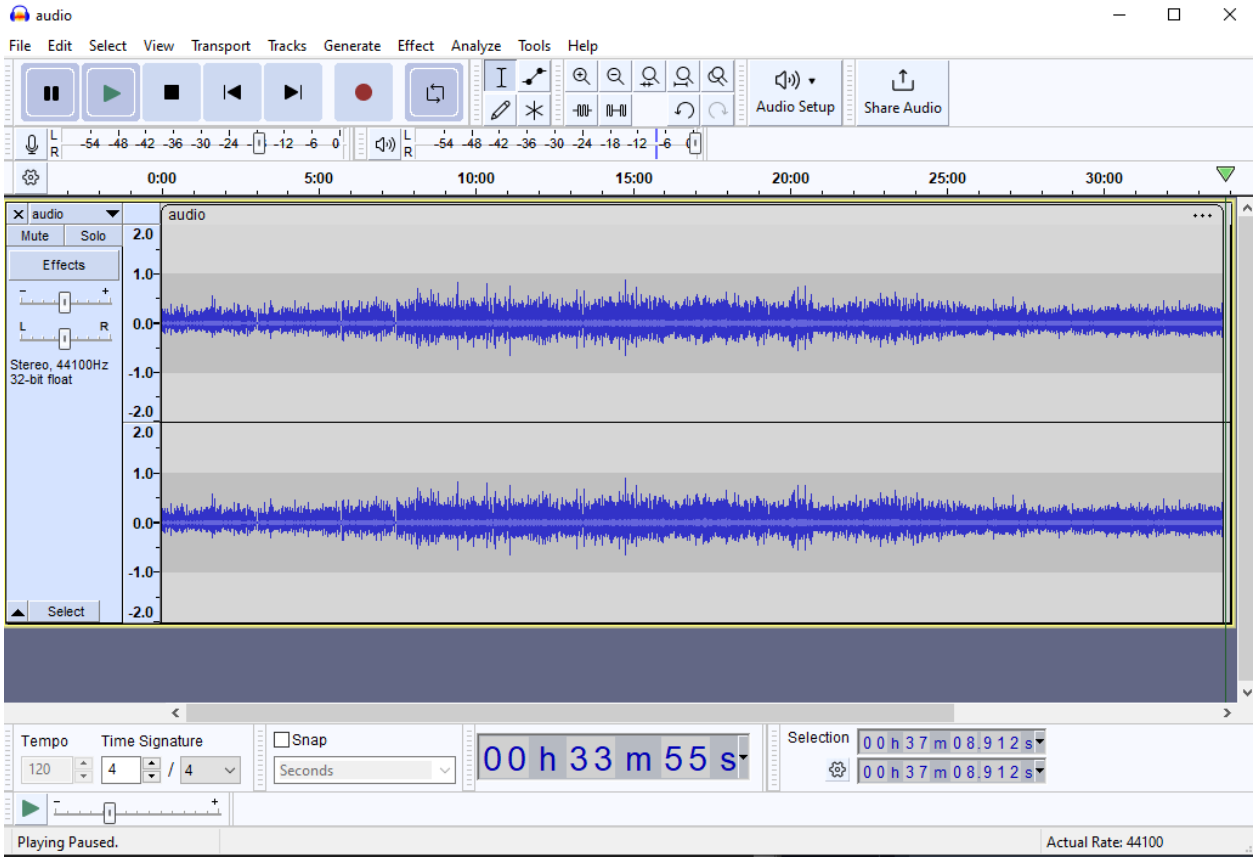


Рисунок 3.3 – Фінальний вигляд аудіо файлу

Далі, аудіо необхідно було розбити на відрізки по 10 секунд, що було зроблено використовуючи бібліотеку soundfile у Python (рис. 3.4).

```
import os
import soundfile as sf

def split_audio_wav(input_file, output_directory, duration=10):
    audio, sr = sf.read(input_file)

    audio_duration = len(audio) / sr
    num_segments = int(audio_duration / duration)

    os.makedirs(output_directory, exist_ok=True)

    for i in range(num_segments):
         start_time = i * duration
        end_time = (i + 1) * duration
        start_index = int(start_time * sr)
        end_index = int(end_time * sr)
        segment = audio[start_index:end_index]
        output_file = os.path.join(output_directory, f"segment_{i+1}.wav")
        sf.write(output_file, segment, sr, format='WAV')
        print(f"Відрізок {i+1} збережено як {output_file}")

def start():
    input_file = "audio.wav"
    output_directory = "segments"
    print("Початок...")
    split_audio_wav(input_file=input_file, output_directory=output_directory)
    print("Готово, відрізки збережено у" + output_directory)

if __name__ == "__main__":
    start()
```

Рисунок 3.4 – виконуваний файл «main.py» для розбиття аудіозапису на відрізки по 10 секунд

3.3.2 Вибір середовища та аналіз задачі

Оскільки доступне обладнання не відповідало мінімальним вимогам для тренування моделі локально, було прийняте рішення скористатися хмарними

сервісами. Зокрема, після порівняння доступних безкоштовних варіантів, між двох найкращих (Paperspace та Colab) було обрано Google Colab, оскільки цей сервіс надає найкраще GPU, хоча і з обмеження у 3.5 години активного використання на добу.

Було створено Colab ноутбук (рис. 3.5) , який включав у себе, зокрема, підключення до Google Drive (за рекомендацією користувачів спільноти AIHub), на якому зберігалися датасет, контрольні точки тренування, та готові файли моделі та індексу, щоб уникнути їх втрати при закінченні доступу до безкоштовних ресурсів. Усі інші необхідні ресурси (загалом приблизно 30 ГБ даних) завантажувалися у хмаровий диск середовища Colab при початку тренування, через ліміт сховища Drive.

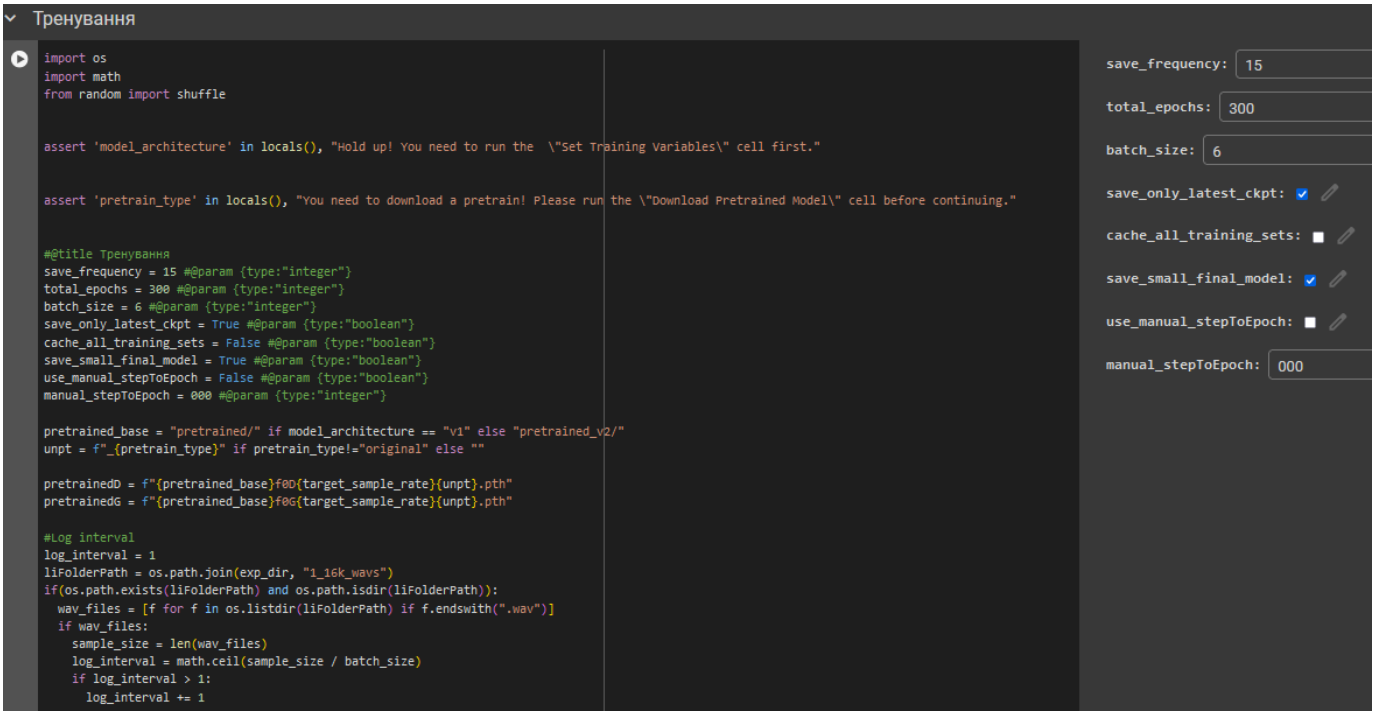


Рисунок 3.5 – демонстрація комірки «Тренування»

У якості попередньо тренованої моделі, було обрано «original», 32к з репозиторію Mangio-RVC-Fork (URL: https://github.com/Mangio621/Mangio-RVC-Fork (дата звернення: 19.05.2024). Попередньо було заплановано тренування довжиною у 300 епох, оскільки за особистими спостереженнями, нижній поріг гарного користувацького досвіду досягався при близько 300х епох тренування.

Як і у інших галузях машинного навчання, перенавчання (overfitting) є проблемою. Для контролю за процесом навчання, а також заради доступу до метрик, у середовищі Colab було встановлено і запущено TensorBoard.

Швидкість тренування залежить від встановленого розміру пакету (batch size). Тестовий запуск тренування продемонстрував, що із розміром 6, модель проходить одну епоху (весь набір даних) приблизно за хвилину. Було зроблено висновок, що тренування мало зайняти приблизно 5 годин. Отже, вирішено розбити тренування на два етапи: від 0 до 150ї епохи, та від 150ї до 300ї.

3.3.3 Перший етап тренування

При виконанні першого етапу, ресурси Google Colab були доступні на протязі 3 годин 30 хвилин. Приблизно 20 хвилин загалом було витрачено на завантаження необхідних файлів, після чого було проведено етап попередньої обробки та виділення показників голосу з датасету (feature extraction) (рис. 3.6).

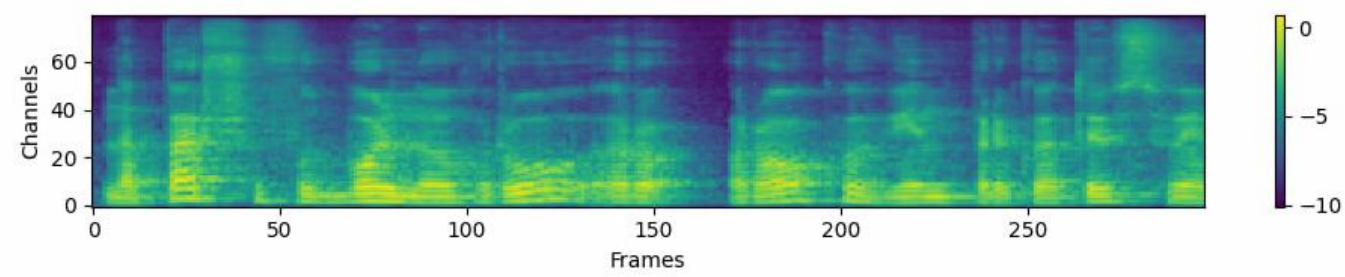


Рисунок 3.6 – мел-спектрограма набору даних «audio»

Після проведення цих процесів, фінальний набір даних було експортовано назад у сховище Drive, щоб уникнути необхідності виконувати обробку заново.

Далі було виконано індексне навчання, яке є однією з основ RVC моделей. Показник індексу використовується для зменшення/вирішення проблеми витоку тембру. Якщо значення індексу дорівнює 1, теоретично немає витоку тембру з джерела виводу, і якість є більш наближеною до навчальної вибірки.

Після індексного навчання, на позначці у 30 хвилин, було запущене тренування вагів, до 150ї епохи, яке зайняло дві з половиною години. Останні 30 хвилин були застосовані для експорту тренованих вагів та показників індексу, створення контрольної точки (checkpoint) для наступного етапу, а також вилучення даних з Tensorboard. «Loss/g/total» (рис. 3.7) вказує на середню похибку генератора моделі, «loss/g/mel» (рис. 3.8) демонструє похибку у репродукції мел-кепстральних коефіцієнтів набору даних, а «loss/d/total»(рис. 3.9) – інформує про середню похибку дискримінатора моделі. Ці три показники вважаються ключовими, і зниження їх значень очевидно призводить до покращення якості вагів. В той же час, існує переломний момент, при якому генеративні властивості переходять до процесу перетренування (overfitting), що призводить до виникнення артефактів при виведенні.

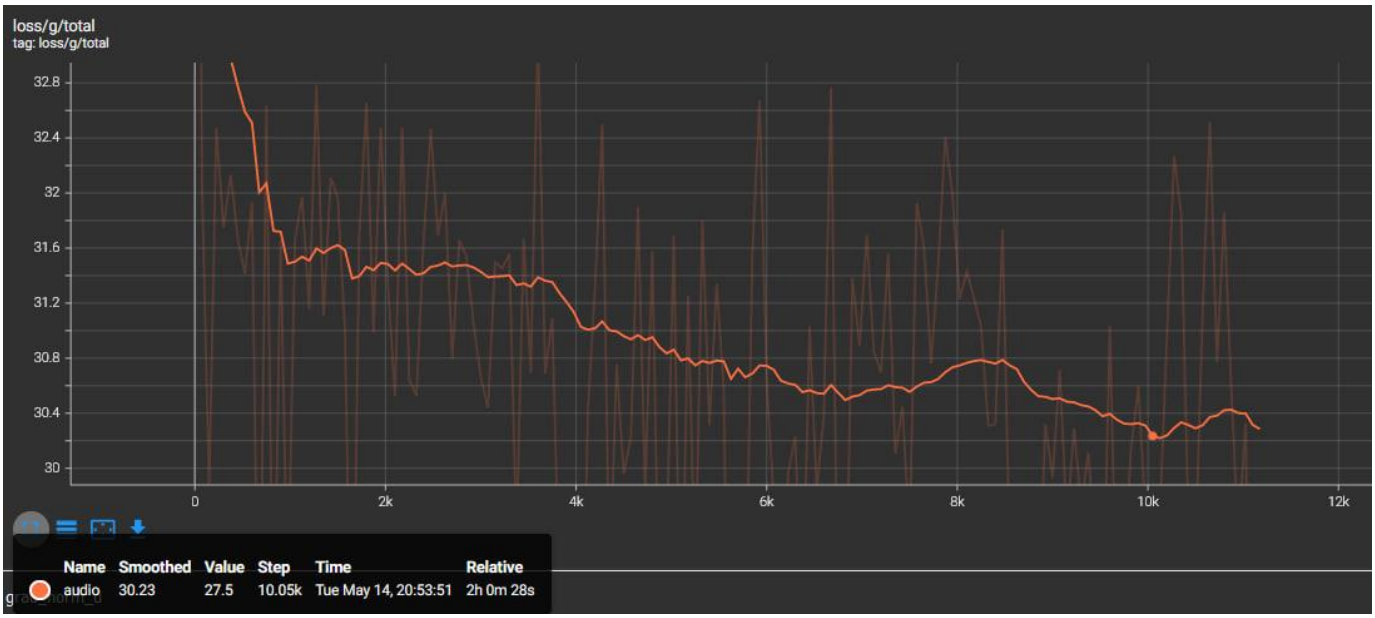


Рисунок 3.7 – графік «loss/g/total», з виділенням на найнижчій точці після 150 епох тренування

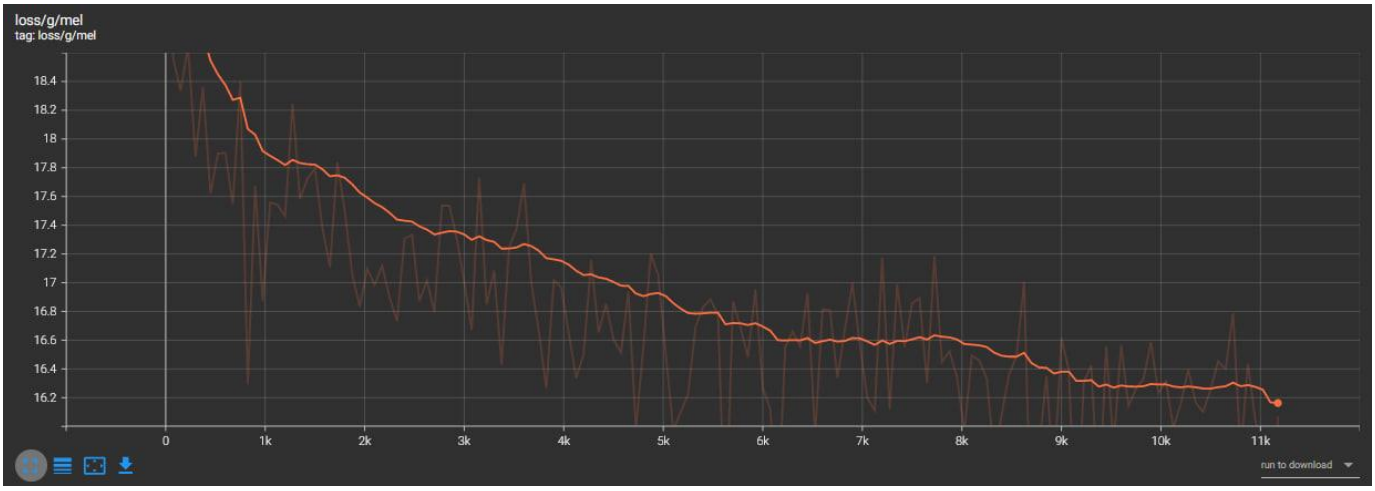


Рисунок 3.8 – графік «loss/g/mel», з виділенням на найнижчій точці після 150 епох тренування

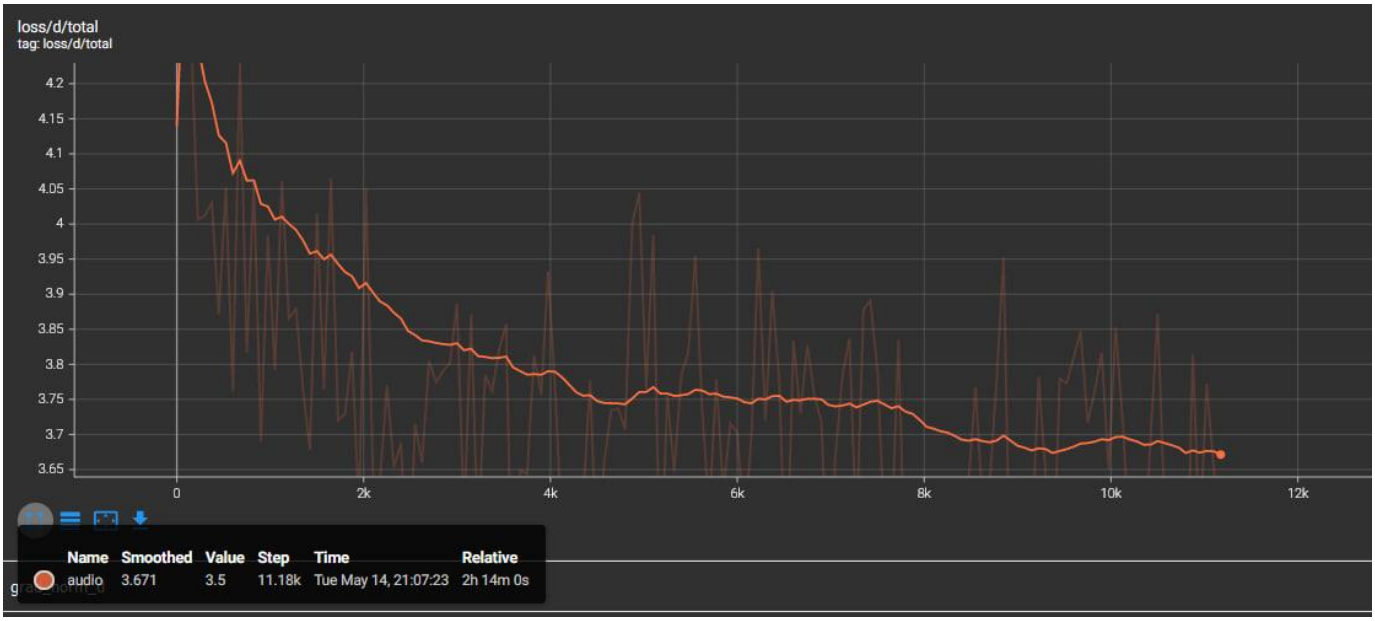


Рисунок 3.9 – графік «loss/d/total», з виділенням на найнижчій точці після 150 епох тренування

3.3.4 Другий етап тренування

При виконанні другого етапу, ресурси Google Colab були доступні на протязі 3 годин 30 хвилин. Приблизно 20 хвилин було витрачено на завантаження необхідних файлів, після чого було виконане підключення до Drive і завантажено контрольну точку. Далі виконувалося тренування вагів до 300ї епохи, яку було досягнуто через 2 години 30 хвилин. Решту доступного часу було використано для експорту індексу, фінальної моделі, а також моделі з 220ї епохи для подальшого порівняння якості, оскільки при аналізі графіків TensorBoard було простежено плато довжиною у 70 хвилин у характеристиці loss/g/total (рис. 3.10), що свідчило про можливе перетренування (рис. 3.11-3.12).

Остаточний вигляд середовища Drive продемонстровано на рисунку 3.13, переглянути його можна за посиланням. (URL:https://drive.google.com/drive/folders/1HuR_sP8muEubT4yir1NNFOMnWqEBYv5V?usp=sharing)

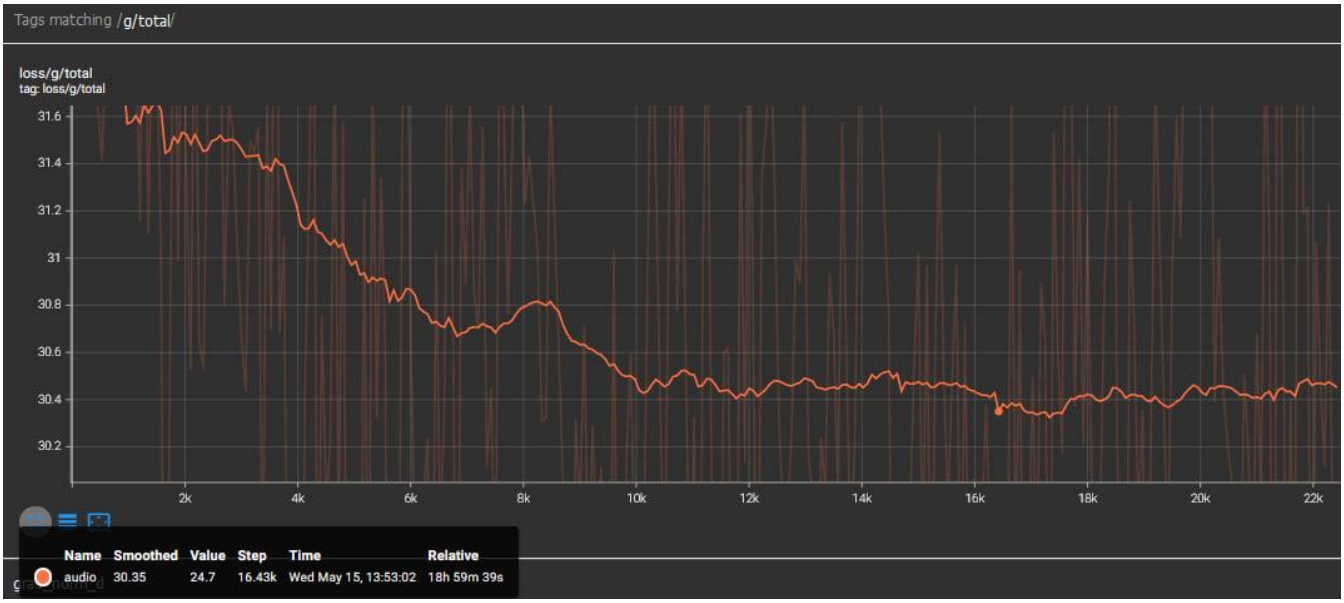


Рисунок 3.10 – графік «loss/g/total» за 300 епох, з виділенням потенційної точки початку перетренування

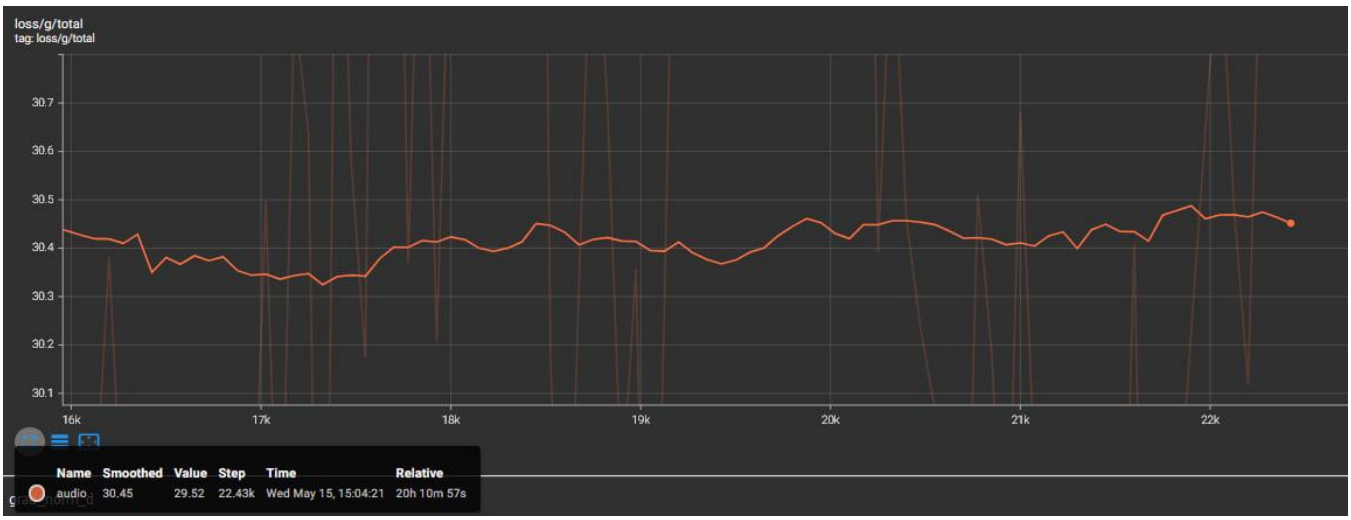


Рисунок 3.11 – графік «loss/g/total» від 210ї до 300ї епохи, повільне зростання вказує на overfitting

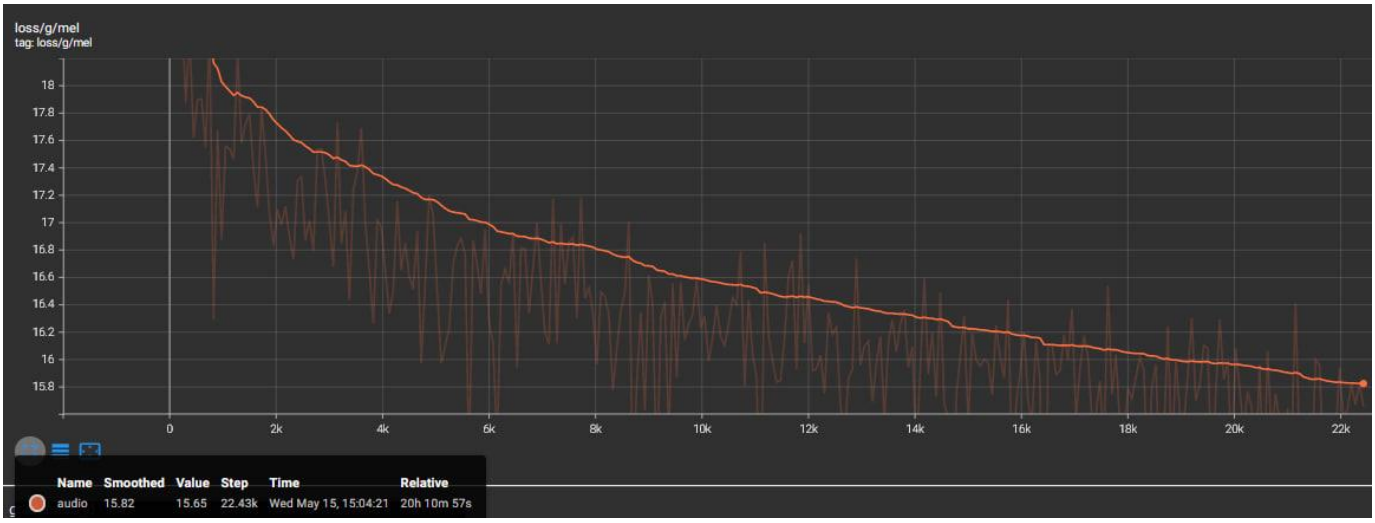


Рисунок 3.12 – графік «loss/g/mel» на 300ій епосі тренування

Name	Owner	Last modified	File size
rvcLogs.zip	me	May 14, 2024 me	423.4 MB
G_2333333.pth	me	May 15, 2024 me	422.7 MB
D_2333333.pth	me	May 15, 2024 me	817.4 MB
config.json	me	May 15, 2024 me	2 KB
audio.pth	me	May 15, 2024 me	53.6 MB
audio_e210_s15750.pth	me	May 15, 2024 me	53.7 MB
added_IVF2071_Flat_nprobe_1_audio_v2.index	me	May 15, 2024 me	243.4 MB

Рисунок 3.13 – середовище Drive після проведення тренування моделі

3.3.5 Демонстрація роботи моделі

У процесі виведення для RVC моделей можна виділити кілька важливих залежностей.

1. Якість звуку сильно впливає на якість створеної моделі, записи з великою кількістю шумів негативно впливають на якість виведення. Так само, якість аудіо яке планується конвертувати впливатиме на результати конверсії.

2. Толерантність моделі до емоційних проявів залежить від присутнього у наборі даних емоційного забарвлення.

3. Сумісність голосу впливає на результати конверсії – при значних відмінностях у просодії мовців спостерігається виникнення шумів та неприроднє мовлення.

З цього можна зробити висновок, що якість клонування голосу залежить не тільки від початкового набору даних чи пройдених епох тренування.

Для виконання процесу виведення, було використано локальний клієнт з графічним інтерфейсом з репозиторію «Mangio-RVC-Fork» (рис. 3.14).

Рисунок 3.14 – Інтерфейс клієнту «Mangio-RVC-Fork»

Далі наведено текстовий опис виконаних тестів та оцінки реалістичності клонованого голосу по десятибальній шкалі, сформовані опитуванням п’ятьох добровольців.

1. Базовий тест працездатності моделі з використанням аудіо довжиною 4 секунди, згенерованого TTS моделю «Дмитро(чоловічій)» автора robinhad (URL: https://huggingface.co/spaces/robinhad/ukrainian-tts (дата звернення 26.05.2024)). Модель з точністю повторила неприродню вимову і наголоси, що сильно знизило реалістичність запису. Оцінка – 7.6/10 балів.

2. Тест моделі з конвертацією голосу, записаного в домашніх умовах. У якості вхідного аудіо – десятисекундний запис чоловічого голосу. На фоні голосу чутно сторонній звуки на третій та дев'ятій секунді запису (цвірінькання папуги), які не вплинули на якість конвертованого аудіо. Оцінка – 9.6/10 балів.

3. Тест моделі з конвертацією голосу з емоційним забарвленням. У якості вхідного аудіо – 54 секунди запису виразного читання «Енеїди» Івана Котляревського. Перед конвертуванням, проведено попередню обробку аудіо, щоб прибрати фонові шуми і нормалізувати гучність. На конвертованому аудіо присутні 3 незначні артефакти на 4й, 47й та 50й секундах. Оцінка – 8/10.

4. Тест з конвертацією вокалу, з просодією, подібною до моделі. У якості вхідного аудіо – фрагмент з пісні «Yakuza 0 – Baka Mitai Українською (Ukrainian Cover) [UkrTrashDub]» (URL: https://www.youtube.com/watch?v=k36VeGjzcYs (дата звернення 26.05.2024)). Вокал відділено від музики за допомогою моделей з проекту UVR5 (Ultimate Vocal Remover 5) (URL: https://github.com/Anjok07/ultimatevocalremovergui (дата звернення 26.05.2024)).

Після обробки оригінального аудіо, на третій секунді присутній гучний сторонній звук (скрипка). Прийнято рішення залишити його, для демонстрації

того, як модель підбирає «найбільш схожий» звук при відсутності подібних до нього.

На вихідному аудіо присутні декілька артефактів, зокрема гучний звук на третій секунді. Зважаючи на те, що модель не була натренована специфічно для конвертації вокалу, автор вважає отриманий результат успішним. Оцінка – 8.2/10

5. Тест з конвертацією вокалу, з просодією, не подібною до моделі. У якості вхідного аудіо – фрагмент з пісні «Карпато-русинські коломийки – Ай Руна Полонина» (URL: https://www.youtube.com/watch?v=n3iVQqfZkJo (дата звернення 26.05.2024)). Обробка вхідного аудіо ідентична попередньому тесту.

Враховуючи неймовірно різні просодії виконавця пісні та автора моделі, результат очікувано поганий. Цей тест виконано для демонстрації «несумісності» голосів, тож оцінювання не проводиться.

Ознайомитися з вхідними та вихідними аудіозаписами можна на YouTube каналі автора. (URL: https://youtu.be/aF_cjdd0u-g (дата звернення 02.06.2024))

3.3.6 Демонстрація роботи RVC у реальному часі

Для демонстрації роботи RVC моделі у реальному часі було виконано локальний запуск клієнта «w-okada voice changer» (URL: https://github.com/w-okada/voice-changer) (рис. 3.15). Клієнт було обрано заради зручного інтерфейсу для налаштування параметрів конверсії.

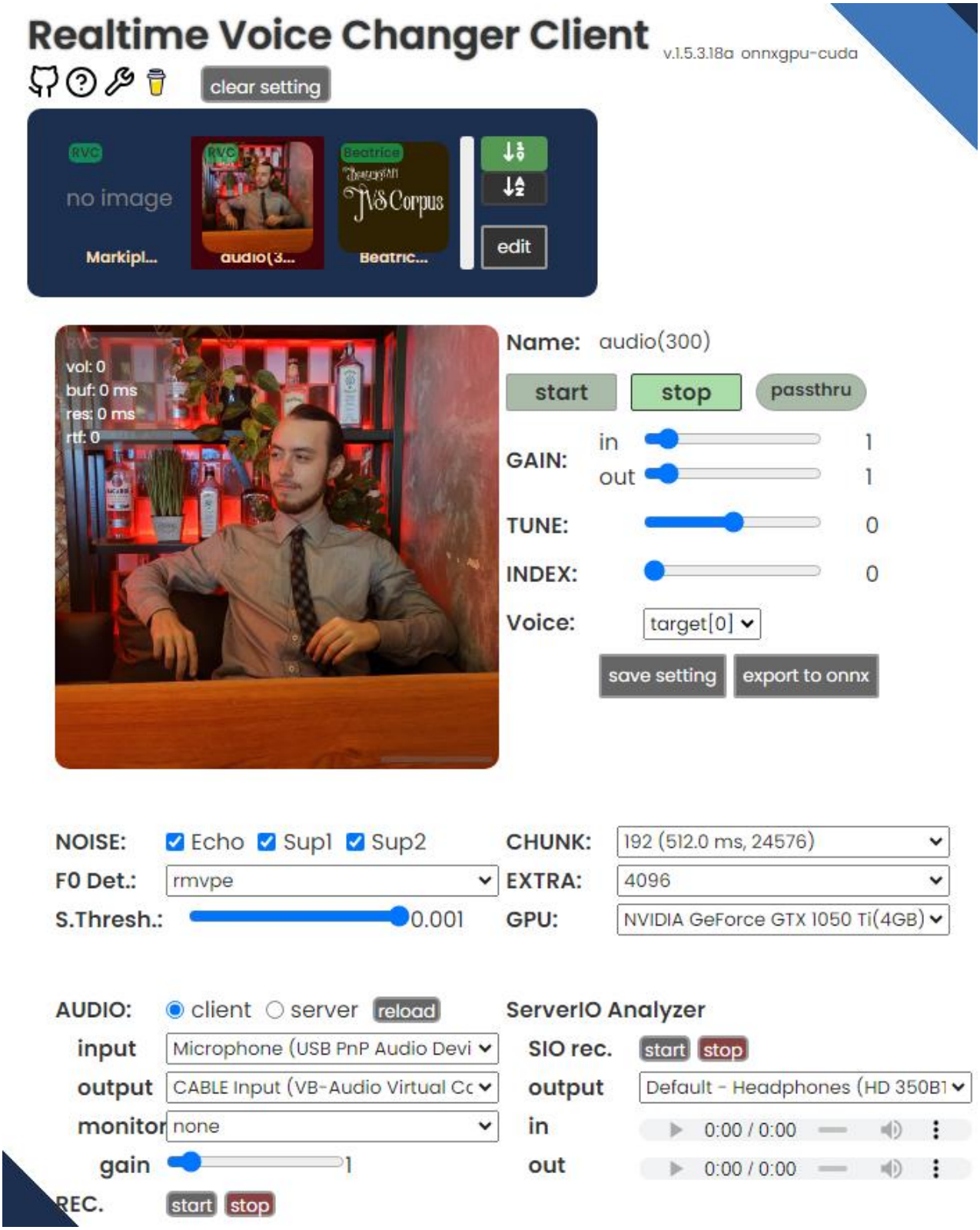


Рисунок 3.15 – Інтерфейс клієнту «w-okada’s voice changer»

Наступні параметри є важливими.

1. CHUNK відповідає за розмір фрагменту мовлення, який конвертується. Більші значення зменшують навантаження на систему, проте негативно впливають на швидкість та якість клонування голосу. Значення у 192 є стандартним, і дає середню якість із затримкою (buff.) у пів секунди.

2. EXTRA визначає, скільки попереднього конвертованого аудіо додавати до входу під час перетворення наступного фрагменту аудіо. Чим більше значення, тим вища точність, але тим довша затримка (res.), що сповільнює процес конверсії і навантажує систему.

3. S.Thresh (Speech Treshold) – бар’єр гучності для розпізнавання аудіо і процесу конвертації. Модель реагує навіть на найменші шуми, конвертуючи їх відповідно – у «дихання», і дана опція дозволяє уникнути навантаження на систему через таку постійну конверсію.

4. F0 Det (F0 Extractor) – вибір алгоритму для здобуття висоти голосу (pitch extraction).

При конверсії у реальному часі виникає затримка, яка визначається як сума buff. (buffer) та res. (residual). На відео демонстрації (URL: https://youtu.be/VKhVGO9ELfA (дата звернення 02.06.2024)), затримка відображується на зображені моделі, середня значення для конкретної конверсії становить приблизно 750 ms.

При середньо-низьких налаштуваннях якості, на слабкій системі (бюджетний комп'ютер 2015го року), отримано гарний результат конверсії. Втім, при використанні разом з іншими інтенсивними процесами, наприклад у голосовому чаті Discord, навантаження на систему, особливо на процесор, значно підвищується, погіршуючи якість клонування голосу. Цю проблему можна вирішити за допомогою використання двох персональних комп’ютерів, один із якої виконуватиме роль серверної машини.

3.4 Висновки до розділу 3

У даному розділі детально розглянуто необхідний технологічний стек для створення голосових моделей. Після короткої демонстрації роботи XTTSv2, було створено власний набір даних та розпочато тренування моделі RVC. Після завершення процессу навчання, виконано аналіз метрик з Tensorflow.

Роботу моделі перевірено на п’яти різних прикладах, якість моделі оцінена п’ятьма добровольцями. Створено відеоматеріали для демонстрації роботи моделі, як при звичайній конверсії, так і при конверсії у реальному часі.

4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В даному розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на дослідженні демографічного стану.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

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

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

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки методу клонування голосу. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

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

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір самої програми.

F_2 – якісний аналіз даних.

F_3 – вибір середовища.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Python.

б) ElevenLabs.

Функція F_2.

а) Застосування вбудованих функцій.

б) Створення своїх обчислень значень.

Функція F_3:

а) Локальне тренування.

б) Хмарні сервіси.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

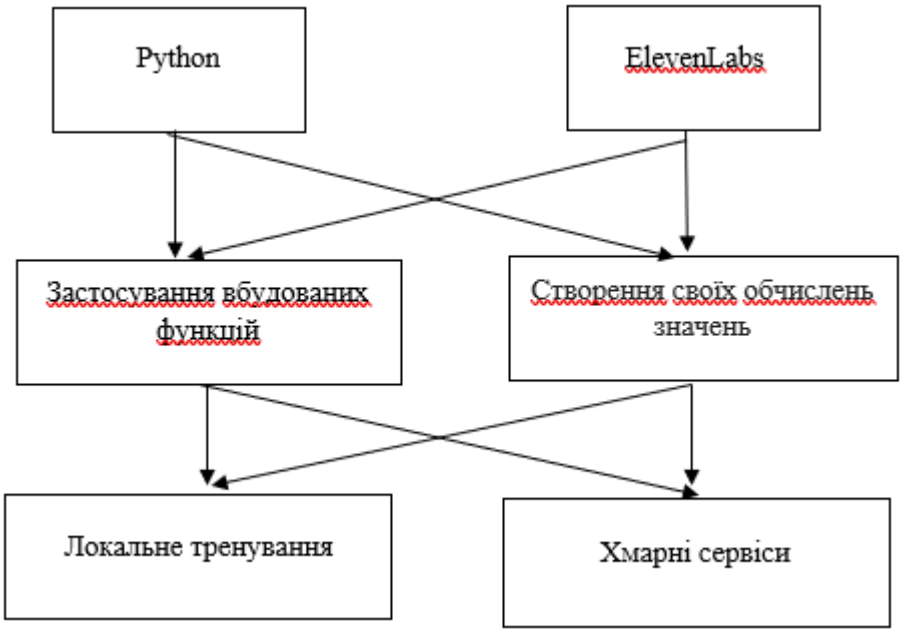


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Загальнодоступна програма, доступність багатьох бібліотек	Необхідність повної реалізації алгоритму
	Б	Швидкість, зрозумілість використання	Комерційна – потребує щомісячних витрат
F_2	*А*	Доступність та легкість при написанні	Іноді не відповідає задачі яку треба розв’язати
	Б	Ідеально описують усі необхідні характеристики	Достатньо затратно реалізовувати свої алгоритми для подальшої реалізації
F_3	*А*	Відсутність щомісячних витрат, контроль над процесом.	Потрібність у початковій інвестиції
	Б	Відсутність великих одноразових витрат, доступ до найсучасніших ресурсів	Обмеження у доступних ресурсах при безкоштовному користуванні або щомісячні витрати

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Перевагу даємо загальнодоступності. Для спрощення роботи варіант Б має бути відкинутий.

Функція F_2:

Програма допускає обрання обох варіантів. Можливо використати варіанти А чи Б.

Функція F_3:

Реалізація першого варіанту є сприйнятливою для програми. Це варіант Б.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а – F_2а – F_3б

F_1а – F_2б – F_3б

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

4.3 Обґрунтування системи параметрів програмного продукту

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

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

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

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	60	80	110
Об’єм пам’яті	X2	Мб	60	50	30
Час попередньої обробки даних	X3	мс	80	70	60
Потенційний об’єм програмного коду	X4	кількість рядків коду	35	25	20

За даними таблиці 4.3 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

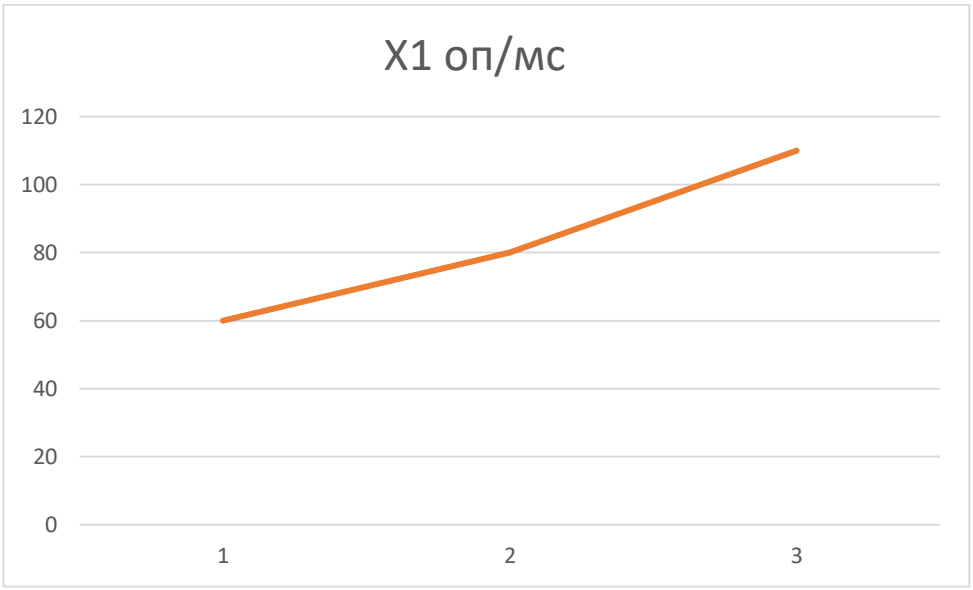


Рисунок 4.2 – X1, швидкодія мови програмування

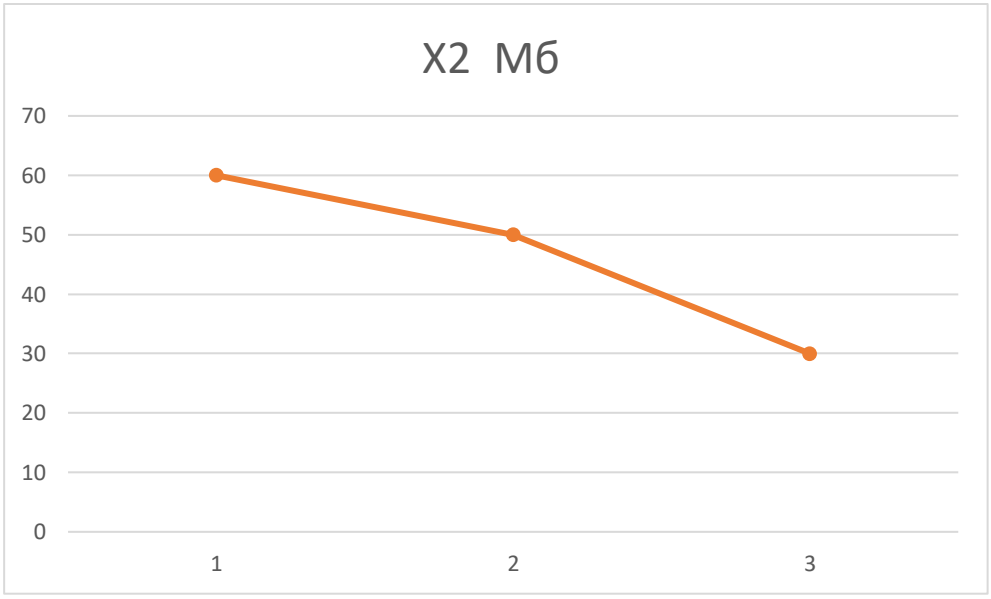


Рисунок 4.3 – X2, об’єм пам’яті

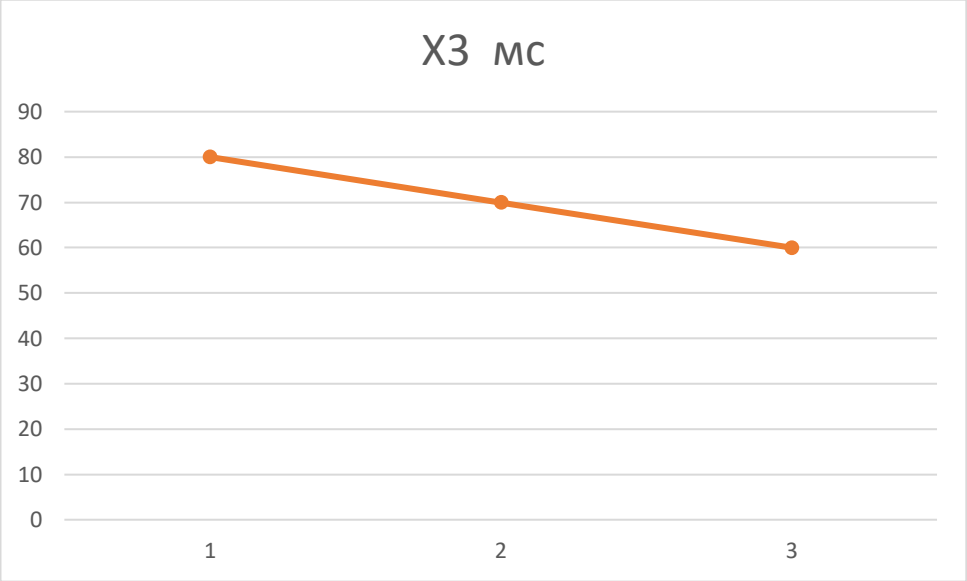


Рисунок 4.4 – X3, час попередньої обробки даних

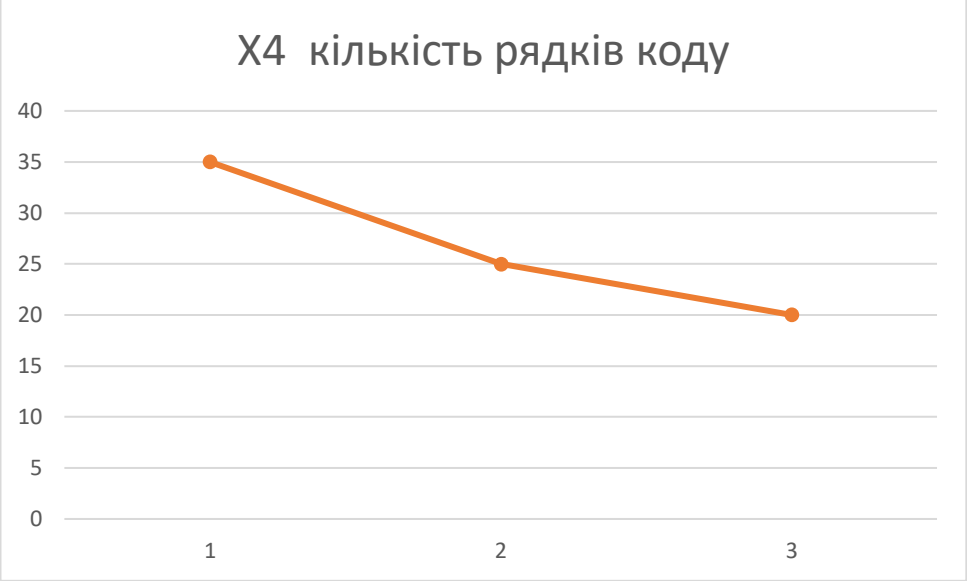


Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка

програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

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

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

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	1	2	2	1	1	1	2	10	-7,5	56,25
X2	Об’єм пам’яті	Mb	3	4	3	3	4	3	4	24	6,5	42,25

Продовження таблиці 4.3

X3	Час попередньої обробки даних	мс	2	1	1	2	2	2	1	11	-6,5	42,25
X4	Потенційний об'єм програмного коду	Кількість рядків коду	4	3	4	4	3	4	3	25	7,5	56,25
	Разом		10	10	10	10	10	10	10	70	0	197

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

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

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.1)$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5 \quad (4.2)$$

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

$$\Delta_i = R_i - T. \quad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197. \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 197}{7^2(4^3 - 4)} = 0{,}754 > W_k = 0{,}67. \quad (4.5)$$

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

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

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

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

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

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

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

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (4.8)$$

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

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \quad (4.9)$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \qquad (4.10)$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	1,5	0,5	0,5	3,5	0,21	12,25	0,18	42,875	0,16
X2	0,5	1	1,5	1,5	4,5	0,28	20,25	0,31	91,125	0,34
X3	1,5	0,5	1	0,5	3,5	0,21	12,25	0,18	42,875	0,16
X4	1,5	0,5	1,5	1	4,5	0,28	20,25	0,31	91,125	0,34
Всього:					16	1	65	1	268	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

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

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

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \tag{4.11}$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	100	25	0,18	4.2
F3	А	X2	87	29	0,41	8,48
	Б	X3	27	19	0,24	3,3
F4	А	X4	25	23	0,36	8,1

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \tag{4.12}$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 4{,}2 + 8{,}48 + 8{,}1 = 20{,}78 ;$$

$$K_{K2} = 4{,}2 + 3{,}3 + 8{,}1 = 15{,}6 .$$

Як видно з розрахунків, кращим є 2 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

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

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

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

Загальна трудомісткість обчислюється як:

$$\mathrm{Т_О} = \mathrm{Т_Р} \cdot \mathrm{К_П} \cdot \mathrm{К_{СК}} \cdot \mathrm{К_М} \cdot \mathrm{К_{СТ}} \cdot \mathrm{К_{СТ.М}}, \qquad (4.13)$$

де $Т_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

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

$К_М$ – коефіцієнт рівня мови програмування;

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

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

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $Т_Р$ =37 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.8. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 37 \cdot 1.8 \cdot 0.9 = 59{,}94 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =29 людино-днів, $К_П$ =0.9, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$Т_2 = 29 \cdot 0.9 \cdot 0.8 = 20.88 \text{ людино-днів.}$$

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

$T_I = (59{,}94 + 20.88 + 4.8 + 20.88) \cdot 8 = 852$ людино-годин.

$T_{II} = (59{,}94 + 20.88 + 6.91 + 20.88) \cdot 8 = 868{,}88$ людино-годин.

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 22100 грн., один аналітик в області даних з окладом 24800. Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{грн.}, \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_Ч = \frac{22100 + 22100 + 24800}{3 \cdot 21 \cdot 8} = 136{,}90 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_Ч \cdot Т_i \cdot К_Д, \quad (4.16)$$

де $С_Ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

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

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 136.90 \cdot 852 \cdot 1.2 = 139966,56$ грн.

II. $С_{ЗП} = 136.90 \cdot 868.88 \cdot 1.2 = 142739,61$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 139966,56 \cdot 0.22 = 30792,64$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 142739,61 \cdot 0.22 = 31402,71$ грн.

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

Так як одна ЕОМ обслуговує одного програміста з окладом 22100 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 22100 \cdot 0,2 = 53040 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 53040 \cdot (1 + 0.2) = 63648 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 63648 \cdot 0,22 = 14002.56 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.4 \cdot 0.25 \cdot 10000 = 3500 \text{ грн.},$$

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

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.4 \cdot 10000 \cdot 0.08 = 1120 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 120 – 12 – 16) \cdot 8 \cdot 0.35 = 607.6 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 607.6 \cdot 0,25 \cdot 0,3 \cdot 5,23 = 238,33 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 10000 \cdot 0,67 = 6700 \quad \text{грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \qquad (4.17)$$

$С_{ЕКС} = 63648 + 14002,56 + 3500 + 1120 + 238,33 + 6700 = 89208,89$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 89208,89 / 607,6 = 146,82 \quad \text{грн/год.}$$

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

$$С_М = С_{М-Г} \cdot Т, \qquad (4.18)$$

I. $С_М = 146,82 \cdot 852 = 125090,64$ грн.

II. $С_М = 146,82 \cdot 868.88 = 127568,96$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67, \quad (4.19)$$

I. $С_Н = 125090,64 \cdot 0,67 = 83810,72$ грн.

II. $С_Н = 127568,96 \cdot 0,67 = 85471,20$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \quad (4.20)$$

I. $С_{ПП} = 139966,56 + 30792,64 + 125090,64 + 83810,72 = 379660,56$ грн.

II. $С_{ПП} = 142739,61 + 31402,71 + 127568,96 + 85471,20 = 387182,48$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{Кj} / С_{Фj}, \quad (4.21)$$

$К_{ТЕР1} = 22,38 / 379660,56 = 5,8947392 \cdot 10^{-5}$,

$К_{ТЕР2} = 16,08 / 387182,48 = 4,1530804 \cdot 10^{-5}$.

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 5{,}8947392 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 5{,}8947392 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

– Вибір мови програмування – Python;

– Реалізація важливої постановки з допомогою вбудованих функцій;

– Використання локальних ресурсів.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

4.8 Висновки до розділу 4

У даному розділі було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних

функцій програмного продукту, а також знайдено параметри, які його характеризують.

ВИСНОВКИ

Клонування голосу існує щонайменше з 1998 року, але розвиток глибокого навчання та нейронних мереж швидко підвищив якість відтворення. Оцінюючи, чи варті переваги клонування голосу ризиків, важливо пам'ятати про те, як інші технології раніше були інтегровані в людський спосіб життя. Від писемності до інтернету, від камери до фотошопу – немає жодної технології, що визначає прогрес, яка б не була використана як на користь, так і для шкоди.

Такі захворювання, як бічний аміотрофічний склероз (БАС), апраксія та черепно-мозкові травми можуть зробити людей невербальними і нездатними спілкуватися за допомогою мови. Зокрема Стівен Гокінг, який жив з БАС і чий синтезатор мовлення, як відомо, був роботизованим за тональністю, мав можливість скористатися клонуванням голосу для персоналізації, проте не зробив цього на знак пошани до Денніса Клатта.

Подібно до того, як ми зберігаємо яйцеклітини і сперму для тих, кому загрожує безпліддя, люди тепер можуть «зберігати» свій голос. Якщо трапиться нещасний випадок або розвинеться хвороба, яка перешкоджатиме використанню мови, голос можна буде відтворити і комп'ютеризувати за допомогою нейронних мереж.

Втрата голосу - це не лише фізичне явище, але й лінгвістичне. Дослідження, опубліковане Австралійським національним університетом у 2022 році, показало, що мовне розмаїття перебуває під загрозою. З 6511 мов, якими зараз розмовляють, близько 1500, ймовірно, зникнуть до кінця цього століття. (URL: https://www.nature.com/articles/s41559-021-01604-y) Мова - це соціальний конструкт, тому вона неминуче живе, змінюється і вмирає залежно від використання.

Клонування голосів носіїв мови, що залишилися, дає можливість відродити її серед молодих поколінь, які прагнуть міцніших зв'язків зі своїми історичними спільнотами. Нинішні методи збереження ґрунтуються на записі зразків мови від мізерної кількості носіїв. Моделі клонування, що постійно вдосконалюються, можуть бути використані для автоматичного створення фонетично точних зразків зникаючих мов і діалектів на основі обмежених і коротких зразків.

Клонування голосу може стати трансформаційним для розваг. Доповіді на конференції TED могли б виголошувати Бертран Рассел або Амелія Ерхарт, а занурення в історію можна було б посилити за допомогою озвучених голосових клонів. І хоча це, зрозуміло, суперечливе явище, клонування голосу може підтримати тяглість оповіді після смерті актора.

Однак такий розвиток подій потребував би широкого кола складних розмов про право власності на голос. Можливість створювати синтетичні голоси, які практично не відрізняються від справжніх, за допомогою безкоштовних моделей з відкритим вихідним кодом викликає тривогу. Клонування голосу вже можна використовувати для обходу систем голосової автентифікації та отримання несанкціонованого доступу до особистих облікових записів.

З точки зору інтелектуальної власності, існує неоднозначність щодо того, чи можна захищати голоси відповідно до чинного законодавства про авторське право. Це потенційно залишає місце для порушення через несанкціоноване клонування. У 2004 році колишній бігун Девід Бедфорд подав успішний позов проти інформаційної компанії «118 118» за використання його зображення - двох вусатих бігунів у білих жилетах - без його дозволу (URL: https://www.theguardian.com/media/2004/jan/27/newmedia.advertising).

«118 118» стверджували, що їхні бігуни були засновані на комічному зображенні типових спортсменів 70-х років. Але чи можна перенести це рішення

на право власності на голос? Звук є перехідним і тимчасовим, тоді як зображення є статичним і постійним: зіставити перше з кимось механічно важче, ніж друге.

На думку автора, інтеграція клонування голосу в особисте і професійне життя, ймовірно, призведе до соціальних змін. Так само як існує необхідність слідкувати за вираженням себе у соціальних мережах, таких як LinkedIn або Facebook, і ретельно обирати, які фотографії опублікувати в Instagram, люди, ймовірно, будуть уважніше ставитися до того, як і що говорять у завантажених відео: адже, зрештою, голос може бути інтелектуальною власністю.

Як і у випадку з будь-якою трансформаційною технологією, відповідальна розробка і впровадження клонування голосу вимагатиме постійного балансу між використанням його переваг і пом'якшенням ризиків. Прагматичний підхід передбачає впровадження надійних етичних принципів, правових рамок і технологічних гарантій. Це не зможе повністю запобігти злочинному використанню, але забезпечить переважне використання клонування голосу, як і інших технологій, на користь.

Дана робота присвячена огляду процесам синтезу мовлення та клонування голосу. Від теоретичного підґрунтя у вигляді оцифрування аудіо та інтерпретації отриманої з нього інформації у розділі 1, до сучасних методів клонування голосу у реальному часі.

У роботі детально розглянуто процес створення RVC моделі, зокрема необхідний технологічний стек, принцип роботи, процес тренування, тощо.

Переваги даного виду моделей полягають у природному звучанні, яке легко отримати завдяки попередньо тренованим моделям. З тої є самої причини, RVC не вимагає великої кількості даних. Відсутність сильного навантаження на систему дозволяє користуватися RVC навіть на старих, бюджетних комп’ютерах, а потрібне обладнання для тренування моделі можна легко замінити за допомогою хмарних сервісів, таких як Google Collaboratory та Paperspace.

Недоліки RVC полягають у обмеженій гнучкості, адже модель часто не може знайти ідеальний фрагмент для перетворення специфічного звуку, і замінює його найближчим сусідом, через що страждає якість конверсії. RVC також потребує відносно багато місця для зберігання файлів (від 30 ГБ). Найбільший недолік RVC полягає у тому, що мовець повинен підлаштовуватися під просодію клонованого голосу, інакше конвертоване аудіо спотворюється.

У процесі роботи було створено модель власного голосу автора. За результатами серії експериментів, визначено просодії з якими конвертація модель відбувається найбільш успішно.

Актуальність роботи полягає у дослідженні принципово нової технології клонування голосу, яка потребує лише одного типу даних для виконання поставленої задачі. Виконана робота може бути підґрунтям до робіт на тему MMVC (Many-to-Many Voice Conversion, клонування декількох мовців), або досліджень нових методів непаралельної конверсії голосу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Anjok07. Ultimate Vocal Remover GUI. Website: Github. 2023. Дата звернення: 26 травня 2024. URL: https://github.com/Anjok07/ultimatevocalremovergui

2. Applio community. Training. Website: Applio Wiki. 2023. Дата звернення: 14 травня 2024. URL: https://applio-page.fandom.com/wiki/Training

3. Berrak Sisman, Junichi Yamagishi, Simon King, Haizhou Li. An Overview of Voice Conversion and its Challenges: From Statistical Modeling to Deep Learning. 17 листопада 2020. Дата звернення: 17 травня 2024. URL: https://arxiv.org/abs/2008.03648

4. Coqui AI. XTTS-v2. 2023. Website: HuggingFace. Дата звернення: 18 травня 2024. URL: https://huggingface.co/coqui/XTTS-v2/blob/main/README.md

5. Diederik P. Kingma, Max Welling. An Introduction to Variational Autoencoders. 11 грудня 2019. Дата звернення: 14 травня 2024. URL: https://arxiv.org/abs/1906.02691

6. Ethan Manilow, Prem Seetharaman, Justin Salamon. Open-Source Tools & Data for Music Source Separation. Website: GitHub. 2020. Дата звернення: 17 травня 2024. URL: https://source-separation.github.io/tutorial/basics/representations.html

7. Facebook, Inc and its affiliates. Welcome to Faiss Documentation. - 2024. Дата звернення: 24 травня 2024. URL: https://faiss.ai/index.html

8. Ganga Babu. Embracing AI voice cloning technology. Website: LinkedIn. 21 жовтня 2023. Дата звернення: 13 травня 2024. URL: https://www.linkedin.com/pulse/embracing-ai-voice-cloning-technology-ganga-babu-ugeic

9. Gölge Eren. XTTS v2 Notes. 11 листопада 2023. Дата звернення: 15 травня 2024. URL: http://erogol.com/2023/11/11/xtts-v2-release-and-notes

10. Haojie Wei, Xueke Cao, Tangpeng Dan, Yueguo Chen. RMVPE: A Robust Model for Vocal Pitch Estimation in Polyphonic Music. 28 червня 2023. Дата звернення: 27 травня 2024. URL: https://arxiv.org/abs/2306.15412v2

11. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Networks. 10 червня 2014. Дата звернення: 12 травня 2024. URL: https://arxiv.org/abs/1406.2661

12. Jeff. A. Bilmes, Yongcheng Lin, Nenad Mladenovic, Ganapati K. Panda Cepstral Coefficient. Website: ScienceDirect. 2020. Дата звернення: 15 травня 2024. URL: https://www.sciencedirect.com/topics/computer-science/cepstral-coefficient

13. Julia, Eddy, Poopmaster Raid, Light, Faze Masta, Alexolotl, Delik, SimplCup, Litsa. AiHub DOCS. Website: AiHub. 2023. Дата звернення: 15 травня 2024. URL: https://docs.aihub.wtf/

14. Jungil Kong, Jaehyeon Kim, Jaekyoung Bae. HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis. 12 жовтня 2020. Дата звернення: 20 травня 2024. URL: https://arxiv.org/abs/2010.05646

15. LinkedIn community. How can WaveNet enhance voice-based interactions and services in different domains and scenarios?. Website: LinkedIn. 2023. Дата звернення: 12 травня 2024. URL: https://www.linkedin.com/advice/3/how-can-wavenet-enhance-voice-based-interactions

16. Mangio 621. Mangio-RVC-Fork. Website: Github. 2023. Дата звернення: 19 травня 2024. URL: https://www.youtube.com/watch?v=n3iVQqfZkJo

17. Music best clean version. Карпато русинська коломийки Ай Руна Полонина з Текстом Хіт (тік ток) 2023. Website: YouTube. 2023. Дата звернення: 26 травня 2024. URL: https://www.youtube.com/watch?v=n3iVQqfZkJo

18. Nadare. RVCのモデルを日本語向けに事前学習する. Website: Qiita. 24 травня 2023. Дата звернення: 20 травня 2024. URL: https://qiita.com/nadare/items/306521c6010bf3efb115

19. NVIDIA. CUDA Python. Website: NVIDIA Developer. Updated: 2024. Дата звернення: 22 травня 2024. URL: https://developer.nvidia.com/cuda-python

20. Prudhviraju Srivatsavaya. LSTM vs GRU. Website: Medium. 24 липня 2023 р.. Дата звернення: 12 травня 2024. URL: https://medium.com/@prudhviraju.srivatsavaya/lstm-vs-gru-c1209b8ecb5a

21. robinhad. ukrainian-tts. 2023. Дата звернення: 26 травня 2024. URL: https://huggingface.co/spaces/robinhad/ukrainian-tts

22. RVC-Boss. RVC-Project. Website: Github. 2023. Дата звернення: 20 травня 2024. URL: https://github.com/RVC-Project

23. Stewart John Q. An electrical analogue of the vocal organs. Nature. 1922.

24. UkrTrashDub 2.0. Yakuza 0 – Baka Mitai Українською (Ukrainian Cover) [UkrTrashDub] [Онлайновий]. Website: YouTube. 2021. Дата звернення: 26 травня 2024. URL: https://www.youtube.com/watch?v=k36VeGjzcYs

25. Unreal Speech. XTTS-V2 Ultimate guide. Website: Unreal Speech. 5 січня 2024. Дата звернення: 17 травня 2024. URL: https://blog.unrealspeech.com/xtts-v2-ultimate-guide/

26. Various contributors. Mangio-RVC-Fork. Website: Github. 28 Липня 2023. Дата звернення: 16 травня 2024. URL: https://github.com/Mangio621/Mangio-RVC-Fork

27. Wei-Ning Hsu, Benjamin Bolte, Yao-Hung HuBERT Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, Abdelrahman Mohamed. HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units. 14 червня 2021. Дата звернення: 16 травня 2024. URL: https://arxiv.org/abs/2106.07447

28. w-okada voice-changer. 2023. Website: Github. Дата звернення: 02 червня 2024. URL: https://github.com/w-okada/voice-changer

29. Нікітаєв Нікіта. Website: Google Drive. 25 Травня 2024. URL: https://drive.google.com/drive/folders/1HuR_sP8muEubT4yir1NNFOMnWqEBYv5V?usp=sharing

30. Нікітаєв Нікіта. demoRVC. Website: YouTube. 02 Червня 2024. URL: https://youtu.be/aF_cjdd0u-g

31. Нікітаєв Нікіта. RVC REALTIME Demo. Website: YouTube. 03 червня 2024. URL: https://www.youtube.com/watch?v=VKhVGO9ELfA

32. Нікітаєв Нікіта. XTTSv2Demo. Website: YouTube. 2 Червня 2024. URL: https://youtu.be/QorGOu8RiJE

33. NVIDIA. Tacotron 2. Website: PyTorch. 2023. Дата звернення: 16 травня 2024. URL: https://pytorch.org/hub/nvidia_deeplearningexamples_tacotron2/

34. The PyTorch Foundation. PyTorch. Website: PyTorch. Дата звернення: 16 травня 2024. URL: https://pytorch.org/hub

35. Various contributors. Audio Transformers course. Website: HuggingFace. 2023. Дата звернення: 11.05.2024. URL: https://huggingface.co/learn/audio-course/chapter0/introduction

36. Python Software Foundation. Python. 2024. URL: https://www.python.org/doc/

37. Librosa development team. Librosa documentation. Website: Librosa. 2023. URL: https://librosa.org/doc/latest/index.html

38. Owen Gibson. Runner wins 118 118 image rights battle. 2004. URL: https://www.theguardian.com/media/2004/jan/27/newmedia.advertising

39. Bromham, L., Dinnage, R., Skirgård, H. et al. Global predictors of language endangerment and the future of linguistic diversity. Nat Ecol Evol **6**, 163–173. 2022. DOI: https://doi.org/10.1038/s41559-021-01604-y

ДОДАТОК А. ЛІСТИНГ ПРОГРАМИ

```
#@title Залежності

import subprocess


packages = ['build-essential', 'python3-dev', 'ffmpeg', 'aria2']

pip_packages = ['pip', 'setuptools', 'wheel', 'faiss-gpu', 'fairseq', 'ffmpeg',
'ffmpeg-python', 'praat-parselmouth', 'pyworld', 'numpy==1.23.5',
'numba==0.56.4', 'librosa==0.9.2']

print("Updating and installing system packages...")

for package in packages:

  print(f"Installing {package}...")

  subprocess.check_call(['apt-get', 'install', '-qq', '-y', package])


print("Updating and installing pip packages...")

subprocess.check_call(['pip', 'install', '--upgrade'] + pip_packages)


print('Packages up to date.')

firsttry = True


#@title РЕПО

import os


os.chdir('/content/')


if(os.path.exists("/content/Mangio-RVC-Fork")):

  print("RVC уже встановлено.")

else:

  !git clone -b pr-optimization --single-branch
https://github.com/alexlnkp/Mangio-RVC-Tweaks.git

  os.rename("/content/Mangio-RVC-Tweaks", "/content/Mangio-RVC-Fork")

  !git clone https://github.com/maxrmorrison/torchcrepe.git
```

```
  !mv torchcrepe/torchcrepe Mangio-RVC-Fork/
  !rm -rf torchcrepe  # Delete the torchcrepe repository folder

os.chdir('/content/Mangio-RVC-Fork')
now_dir = "/content/Mangio-RVC-Fork"
os.makedirs(os.path.join(now_dir, "logs"), exist_ok=True)
os.makedirs(os.path.join(now_dir, "weights"), exist_ok=True)

#@title GPU
import torch

ngpu = torch.cuda.device_count()
gpu_infos = []
mem = []
if_gpu_ok = False

if torch.cuda.is_available() or ngpu != 0:
  for i in range(ngpu):
    gpu_name = torch.cuda.get_device_name(i)
    if any(
        value in gpu_name.upper()
        for value in ["10", "16", "20", "30", "40", "A2", "A3", "A4", "P4",
"A50", "500", "A60", "70", "80", "90", "M4", "T4", "TITAN"]
    ):
      if_gpu_ok = True
      print("GPU знайдено: %s" % gpu_name)
      gpu_infos.append("%s\t%s" % (i, gpu_name))
      mem.append(int(torch.cuda.get_device_properties(i).total_memory / 1024 /
1024 / 1024 + 0.4))

if if_gpu_ok and len(gpu_infos) > 0:
  gpu_info = "\n".join(gpu_infos)
```

```
else:
  raise Exception("GPU не знайдено")
gpus = "-".join(i[0] for i in gpu_infos)


#@title Drive
from google.colab import drive
if not os.path.exists('/content/drive'):
  drive.mount('/content/drive')
else:
  print('Drive уже підключено.')


os.makedirs('/content/drive/MyDrive/rvcDisconnected', exist_ok=True)


#@title Didn't ask.


#HuBERT/RMVPE
!aria2c --console-log-level=error -c -x 16 -s 16 -k 1M
https://huggingface.co/Kit-Lemonfoot/RVC_DidntAsk/resolve/main/HuBERT_base.pt -d
/content/Mangio-RVC-Fork -o HuBERT_base.pt
!aria2c --console-log-level=error -c -x 16 -s 16 -k 1M
https://huggingface.co/Kit-Lemonfoot/RVC_DidntAsk/resolve/main/rmvpe.pt -d
/content/Mangio-RVC-Fork -o rmvpe.pt


#FM JSONs
!rm -rf /content/Mangio-RVC-Fork/configs/32k.json
!rm -rf /content/Mangio-RVC-Fork/configs/40k.json
!rm -rf /content/Mangio-RVC-Fork/configs/48k.json
!aria2c --console-log-level=error -c -x 16 -s 16 -k 1M
https://huggingface.co/Kit-Lemonfoot/RVC_DidntAsk/resolve/main/32k.json -d
/content/Mangio-RVC-Fork/configs -o 32k.json
!aria2c --console-log-level=error -c -x 16 -s 16 -k 1M
https://huggingface.co/Kit-Lemonfoot/RVC_DidntAsk/resolve/main/40k.json -d
/content/Mangio-RVC-Fork/configs -o 40k.json
!aria2c --console-log-level=error -c -x 16 -s 16 -k 1M
https://huggingface.co/Kit-Lemonfoot/RVC_DidntAsk/resolve/main/48k.json -d
/content/Mangio-RVC-Fork/configs -o 48k.json
```

```
#@title CSVDB

import csv


if not os.path.isdir("csvdb/"):

  os.makedirs("csvdb")

  frmnt, stp = open("csvdb/formanting.csv", "w", newline=""),
open("csvdb/stop.csv", "w", newline="")

  csv_writer = csv.writer(frmnt, delimiter=",")

  csv_writer.writerow([False, 1.0, 1.0])

  csv_writer = csv.writer(stp, delimiter=",")

  csv_writer.writerow([False])

  frmnt.close()

  stp.close()


global DoFormant, Quefrency, Timbre

DoFormant, Quefrency, Timbre = False, 1.0, 1.0


#@title Параметри тренування

now_dir = "/content/Mangio-RVC-Fork"

experiment_name = "experiment_name" #@param {type:"string"}

path_to_training_folder = "/content/dataset/"

model_architecture = "v2" #@param ["v1","v2"] {allow-input: false}

target_sample_rate = "40k" #@param ["32k", "40k", "48k"] {allow-input: false}

speaker_id = 0 #@param {type:"integer"}

pitch_extraction_algorithm = "rmvpe" #@param ["harvest", "crepe", "mangio-
crepe", "rmvpe"] {allow-input: false}

crepe_hop_length = 64 #@param {type:"integer"}

pitch_guidance = True #@param {type:"boolean"}


cpu_threads = !nproc

cpu_threads = int(cpu_threads[0])
```

```
exp_dir = f"{now_dir}/logs/{experiment_name}"


assert crepe_hop_length!=None, "crepe_hop_length порожнє."

assert crepe_hop_length>0, "Hop length більше 0."

assert crepe_hop_length<=512, "Save frequency менше 512."


if(experiment_name == "experiment_name"):

  print("Warning: Your experiment name should be changed to the name of your
dataset.")


#@title Попередньо тренована модель

pretrain_type = "original" #@param ["original", "OV2Super", "RIN_E3", "ItaIla",
"SnowieV3", "SnowieV3xRIN_E3", "TITAN", "custom"] {allow-input: false}

g="" #@param {type:"string"}

d="" #@param {type:"string"}

assert 'model_architecture' in locals(), "Налаштувати параметри тренування"


if pretrain_type!="original" and model_architecture!="v2":

  model_architecture="v2"

  print("Тільки RVC v2 для нових моделей.")


if pretrain_type=="OV2Super" and target_sample_rate=="48k":

  target_sample_rate="40k"

  print("OV2Super підтримує <= 40k.")

if pretrain_type=="RIN_E3" and target_sample_rate!="40k":

  target_sample_rate="40k"

  print("RIN_E3 підтримує <= 40k.")

if pretrain_type=="ItaIla" and target_sample_rate!="32k":

  target_sample_rate="32k"

  print("ItaIla підтримує <= 32k.")

if pretrain_type=="SnowieV3xRIN_E3" and target_sample_rate!="40k":

  target_sample_rate="40k"

  print("SnowieV3 x RIN_E3 підтримує <= 40k.")
```

```
print("Setting up...")

if pretrain_type=="original" and model_architecture=="v2":

  g = f"https://huggingface.co/Kit-
Lemonfoot/RVC_DidntAsk/resolve/main/f0G{target_sample_rate}.pth"

  d = f"https://huggingface.co/Kit-
Lemonfoot/RVC_DidntAsk/resolve/main/f0D{target_sample_rate}.pth"

if pretrain_type=="original" and model_architecture=="v1":

  g = f"https://huggingface.co/Kit-
Lemonfoot/RVC_DidntAsk/resolve/main/v1/f0G{target_sample_rate}.pth"

  d = f"https://huggingface.co/Kit-
Lemonfoot/RVC_DidntAsk/resolve/main/v1/f0D{target_sample_rate}.pth"

if pretrain_type=="OV2Super" and target_sample_rate=="40k":

  g =
f"https://huggingface.co/ORVC/Ov2Super/resolve/main/f0Ov2Super{target_sample_rat
e}G.pth"

  d =
f"https://huggingface.co/ORVC/Ov2Super/resolve/main/f0Ov2Super{target_sample_rat
e}D.pth"

if pretrain_type=="OV2Super" and target_sample_rate=="32k":

  g =
f"https://huggingface.co/poiqazwsx/Ov2Super32kfix/resolve/main/f0Ov2Super32kD.pt
h"

  d =
f"https://huggingface.co/poiqazwsx/Ov2Super32kfix/resolve/main/f0Ov2Super32kD.pt
h"

if pretrain_type=="RIN_E3":

  g = "https://huggingface.co/MUSTAR/RIN_E3/resolve/main/RIN_E3_G.pth"

  d = "https://huggingface.co/MUSTAR/RIN_E3/resolve/main/RIN_E3_D.pth"

if pretrain_type=="SnowieV3":

  g = f"https://huggingface.co/MUSTAR/SnowieV3.1-
{target_sample_rate}/resolve/main/G_SnowieV3.1_{target_sample_rate}.pth"

  d = f"https://huggingface.co/MUSTAR/SnowieV3.1-
{target_sample_rate}/resolve/main/D_SnowieV3.1_{target_sample_rate}.pth"

if pretrain_type=="ItaIla":

  g = "https://huggingface.co/TheStinger/itaila/resolve/main/ItaIla_32k_G.pth"

  d = "https://huggingface.co/TheStinger/itaila/resolve/main/ItaIla_32k_D.pth"

if pretrain_type=="SnowieV3xRIN_E3":
```

```
  g = "https://huggingface.co/MUSTAR/SnowieV3.1-X-RinE3-
40K/resolve/main/G_Snowie-X-Rin_40k.pth"

  d = "https://huggingface.co/MUSTAR/SnowieV3.1-X-RinE3-
40K/resolve/main/D_Snowie-X-Rin_40k.pth"

if pretrain_type=="TITAN":

  g = f"https://huggingface.co/blaise-
tk/TITAN/resolve/main/models/medium/{target_sample_rate}/pretrained/G-
f0{target_sample_rate}-TITAN-Medium.pth"

  d = f"https://huggingface.co/blaise-
tk/TITAN/resolve/main/models/medium/{target_sample_rate}/pretrained/D-
f0{target_sample_rate}-TITAN-Medium.pth"


pretrained_base = "pretrained/" if model_architecture == "v1" else
"pretrained_v2/"

unpt = f"_{pretrain_type}" if pretrain_type!="original" else ""


print("Завантаження...")

!aria2c --console-log-level=error -q -c -x 16 -s 16 -k 1M {g} -d
/content/Mangio-RVC-Fork/{pretrained_base} -o f0G{target_sample_rate}{unpt}.pth

!aria2c --console-log-level=error -q -c -x 16 -s 16 -k 1M {d} -d
/content/Mangio-RVC-Fork/{pretrained_base} -o f0D{target_sample_rate}{unpt}.pth


print("Модель завантажено!")



#@title Завантаження датасету

dataset = "zipfile.zip"  #@param {type:"string"}


import os

import shutil


directories=[]


def sanitize_directory(directory):

  for filename in os.listdir(directory):

    file_path = os.path.join(directory, filename)
```

```
    if os.path.isfile(file_path):
      if filename == ".DS_Store" or filename.startswith("._") or not
filename.endswith(('.wav', '.flac', '.mp3', '.ogg', '.m4a')):
        os.remove(file_path)
    elif os.path.isdir(file_path):
      if(filename == "__MACOSX"):
        shutil.rmtree(file_path)
        continue
      directories.append(file_path)
      sanitize_directory(file_path)


dataset_path = '/content/drive/MyDrive/rvcDisconnected/' + dataset
final_directory = '/content/dataset'
temp_directory = '/content/temp_dataset'


if os.path.exists(final_directory):
  print("Dataset folder already found. Wiping...")
  shutil.rmtree(final_directory)
if os.path.exists(temp_directory):
  print("Temporary folder already found. Wiping...")
  shutil.rmtree(temp_directory)


if not os.path.exists(dataset_path):
  raise Exception(f'I can\'t find {dataset} in
{os.path.dirname(dataset_path)}.')


os.makedirs(final_directory, exist_ok=True)
os.makedirs(temp_directory, exist_ok=True)
#Oops.
!unzip -d "{temp_directory}" -B "{dataset_path}"
print("Sanitizing...")
sanitize_directory(temp_directory)
```

```
if(len(directories) == 0):
  print("Dataset Type: Audio Files (Single Speaker)")
  expDir=os.path.join(final_directory, experiment_name)
  os.makedirs(expDir, exist_ok=True)
  for r, _, f in os.walk(temp_directory):
    for name in f:
      !cp "{temp_directory}/{name}" "{expDir}"
elif(len(directories) == 1):
  #If there's only one directory, we're dealing with a single speaker.
  #Rename the folder to experiment_name and move it to /dataset/.
  print("Dataset Type: Single Speaker")
  fi = os.path.join(temp_directory, experiment_name)
  os.rename(directories[0], fi)
  shutil.move(fi, final_directory)

else:
  #If anything else, we're dealing with multispeaker.
  #Move all folders to /dataset/ indiscriminately.
  print("Dataset Type: Multispeaker")
  for fi in directories:
    shutil.move(fi, final_directory)

shutil.rmtree(temp_directory)

print("Датасет завантажено.")

#@title Попередня обробка і Feature Extraction

import os
import subprocess

assert cpu_threads>0, "CPU threads not allocated correctly."
```

```
sr = int(target_sample_rate.rstrip('k'))*1000

pttf = path_to_training_folder + experiment_name

os.makedirs(f"{exp_dir}", exist_ok=True)


cmd = f"python trainset_preprocess_pipeline_print.py \"{pttf}\" {sr}
{cpu_threads} \"{exp_dir}\" 1"

print(cmd)

!$cmd


gpuList = gpus.split("-")

cmd = f"python extract_f0_print.py \"{exp_dir}\" {cpu_threads}
{pitch_extraction_algorithm} {crepe_hop_length}"

print(cmd)

!$cmd


leng = len(gpus)

cmd = f"python extract_feature_print.py \"device\" {leng} 0 0 \"{exp_dir}\"
{model_architecture}"

print(cmd)

!$cmd


#@title Save Зберегти оброблені дані в Google Drive

loc = "%s/logs/%s" % (now_dir, experiment_name)

!zip -r rvcLogs.zip "{loc}"

DATASET_PATH_DRIVE = "/content/drive/MyDrive/rvcDisconnected/" + experiment_name

!mkdir -p "{DATASET_PATH_DRIVE}"

!cp /content/Mangio-RVC-Fork/rvcLogs.zip "{DATASET_PATH_DRIVE}"


#@title Завантажити датасет з Google Drive

import os


BACK_UP_DATASET_PATH = "/content/drive/MyDrive/rvcDisconnected/" +
experiment_name
```

```
ok=True
if(os.path.exists("/content/Mangio-RVC-Fork/logs/"+experiment_name+"/2a_f0")):
  print("Dataset files already loaded, skipping.")
  ok=False

if ok:
  !unzip "{BACK_UP_DATASET_PATH}/rvcLogs.zip" -d /

#@title Завантажити модель з Drive (для продовження тренування)
import os

DATASET_PATH_DRIVE = "/content/drive/MyDrive/rvcDisconnected/" + experiment_name
DATASET_PATH_COLAB = "/content/Mangio-RVC-Fork/logs/" + experiment_name
STEPCOUNT = 000 #@param {type:"integer"}

print("Copying model files...")
!cp "{DATASET_PATH_DRIVE}/D_{STEPCOUNT}.pth" "{DATASET_PATH_COLAB}"
!cp "{DATASET_PATH_DRIVE}/G_{STEPCOUNT}.pth" "{DATASET_PATH_COLAB}"
!cp "{DATASET_PATH_DRIVE}/config.json" "{DATASET_PATH_COLAB}"

print("Copying Tensorboard TFEVENT files...")
for r, _, f in os.walk(DATASET_PATH_DRIVE):
  for name in f:
    if(name.startswith("events.out.tfevents")):
      !cp "{DATASET_PATH_DRIVE}/{name}" "{DATASET_PATH_COLAB}"

#@title Індексне тренування

import os
import sys
import traceback
```

```
import numpy as np
import faiss

#from sklearn.cluster import MiniBatchKMeans

exp_dir = "%s/logs/%s" % (now_dir, experiment_name)
os.makedirs(exp_dir, exist_ok=True)
feature_dir = (
    "%s/3_feature256" % (exp_dir)
    if model_architecture == "v1"
    else "%s/3_feature768" % (exp_dir)
)
print(feature_dir)
if not os.path.exists(feature_dir):
  raise Exception("Feature Extraction")
listdir_res = list(os.listdir(feature_dir))
if len(listdir_res) == 0:
  raise Exception("Feature Extraction")
try:
  from sklearn.cluster import MiniBatchKMeans
except:
  !pip install -U numpy
  sys.exit()
else:
  print("Numpy detected.")

infos=[]
npys=[]
for name in sorted(listdir_res):
  phone = np.load("%s/%s" % (feature_dir, name))
  npys.append(phone)
big_npy = np.concatenate(npys, 0)
```

```
big_npy_idx = np.arange(big_npy.shape[0])
np.random.shuffle(big_npy_idx)
if big_npy.shape[0] > 2e5 or force_mbkm:
  print("Trying doing kmeans %s shape to 10k centers." % big_npy.shape[0])
  try:
    big_npy = (
        MiniBatchKMeans(
            n_clusters=10000,
            verbose=True,
            batch_size=256,
            compute_labels = False,
            init="random"
        )
        .fit(big_npy)
        .cluster_centers_

    )
  except:
    info = traceback.format_exc()
    print(info)

np.save("%s/total_fea.npy" % exp_dir, big_npy)
n_ivf = min(int(16*np.sqrt(big_npy.shape[0])), big_npy.shape[0] // 39)
print("%s,%s" % (big_npy.shape, n_ivf))
index = faiss.index_factory(256 if model_architecture == "v1" else 768,
"IVF%s,Flat" % n_ivf)
print("Training index...")
index_ivf = faiss.extract_index_ivf(index)
index_ivf.nprobe = 1
index.train(big_npy)
faiss.write_index(
    index,
```

```
    "%s/trained_IVF%s_Flat_nprobe_%s_%s_%s.index" % (exp_dir, n_ivf,
index_ivf.nprobe, experiment_name, model_architecture)

)

print("Adding...")

batch_size_add = 8192

for i in range(0, big_npy.shape[0], batch_size_add):

  index.add(big_npy[i:i+batch_size_add])

faiss.write_index(

    index,

    "%s/added_IVF%s_Flat_nprobe_%s_%s_%s.index"

    % (exp_dir, n_ivf, index_ivf.nprobe, experiment_name, model_architecture)

)


npr = index_ivf.nprobe


print("Saving files to Drive...")

DATASET_PATH_DRIVE = "/content/drive/MyDrive/rvcDisconnected/" + experiment_name

if(not os.path.exists(DATASET_PATH_DRIVE)):

  !mkdir -p "{DATASET_PATH_DRIVE}"

DATASET_PATH_COLAB = "/content/Mangio-RVC-Fork/logs/" + experiment_name

if(save_extra_files_to_drive):

  !cp "{DATASET_PATH_COLAB}/total_fea.npy" "{DATASET_PATH_DRIVE}"

  !cp
"{DATASET_PATH_COLAB}/trained_IVF{n_ivf}_Flat_nprobe_{npr}_{experiment_name}_{mo
del_architecture}.index" "{DATASET_PATH_DRIVE}"

!cp
"{DATASET_PATH_COLAB}/added_IVF{n_ivf}_Flat_nprobe_{npr}_{experiment_name}_{mode
l_architecture}.index" "{DATASET_PATH_DRIVE}"



import os

import math

from random import shuffle
```

```
assert 'model_architecture' in locals(), "Hold up! You need to run the  \"Set
Training Variables\" cell first."



assert 'pretrain_type' in locals(), "You need to download a pretrain! Please run
the \"Download Pretrained Model\" cell before continuing."



#@title Тренування

save_frequency = 10 #@param {type:"integer"}

total_epochs = 500 #@param {type:"integer"}

batch_size = 8 #@param {type:"integer"}

save_only_latest_ckpt = True #@param {type:"boolean"}

cache_all_training_sets = False #@param {type:"boolean"}

save_small_final_model = True #@param {type:"boolean"}

use_manual_stepToEpoch = False #@param {type:"boolean"}

manual_stepToEpoch = 000 #@param {type:"integer"}

enable_autosave = False #@param {type:"boolean"}


pretrained_base = "pretrained/" if model_architecture == "v1" else
"pretrained_v2/"

unpt = f"_{pretrain_type}" if pretrain_type!="original" else ""


pretrainedD = f"{pretrained_base}f0D{target_sample_rate}{unpt}.pth"

pretrainedG = f"{pretrained_base}f0G{target_sample_rate}{unpt}.pth"


#Log interval

log_interval = 1

liFolderPath = os.path.join(exp_dir, "1_16k_wavs")

if(os.path.exists(liFolderPath) and os.path.isdir(liFolderPath)):

  wav_files = [f for f in os.listdir(liFolderPath) if f.endswith(".wav")]

  if wav_files:
```

```
    sample_size = len(wav_files)
    log_interval = math.ceil(sample_size / batch_size)
    if log_interval > 1:
      log_interval += 1


if log_interval > 250 and not use_manual_stepToEpoch:
  log_interval = 200


if use_manual_stepToEpoch:
  log_interval = manual_stepToEpoch


#Credit to Sonphantrung for dreaming this up.
if enable_autosave:
  !sed -i
"s/weights\/\%s/\/content\/drive\/MyDrive\/rvcDisconnected\/$experiment_name\/\%
s/g" -i {now_dir}/train/process_ckpt.py
  !sed -i
"s/.\/logs/\/content\/drive\/MyDrive\/rvcDisconnected\/$experiment_name\/\%s/g"
-i {now_dir}/train/utils.py


cmd = "python train_nsf_sim_cache_sid_load_pretrain.py -e \"%s\" -sr %s -f0 %s -
bs %s -g %s -te %s -se %s %s %s -l %s -c %s -sw %s -v %s -li %s" % (
    experiment_name,
    target_sample_rate,
    1,
    batch_size,
    0,
    total_epochs,
    save_frequency,
    "-pg %s" % pretrainedG if pretrainedG != "" else "\b",
    "-pd %s" % pretrainedD if pretrainedD != "" else "\b",
    1 if save_only_latest_ckpt else 0,
    1 if cache_all_training_sets else 0,
    1 if save_small_final_model else 0,
```

```
    model_architecture,
    log_interval,
)
print(cmd)

gt_wavs_dir = f"{exp_dir}/0_gt_wavs"
feature_dir = (
  f"{exp_dir}/3_feature256"
  if model_architecture == "v1"
  else f"{exp_dir}/3_feature768"
)
f0_dir = f"{exp_dir}/2a_f0"
f0nsf_dir = f"{exp_dir}/2b-f0nsf"
names = (
  set([name.split(".")[0] for name in os.listdir(gt_wavs_dir)])
  & set([name.split(".")[0] for name in os.listdir(feature_dir)])
  & set([name.split(".")[0] for name in os.listdir(f0_dir)])
  & set([name.split(".")[0] for name in os.listdir(f0nsf_dir)])
)
opt = []
for name in names:
  opt.append(
    "%s/%s.wav|%s/%s.npy|%s/%s.wav.npy|%s/%s.wav.npy|%s"
    % (
      gt_wavs_dir.replace("\\", "\\\\"),
      name,
      feature_dir.replace("\\", "\\\\"),
      name,
      f0_dir.replace("\\", "\\\\"),
      name,
      f0nsf_dir.replace("\\", "\\\\"),
      name,
```

```
      speaker_id,
    )
  )
fea_dim = 256 if model_architecture == "v1" else 768
for _ in range(2):
  opt.append(
f"{now_dir}/logs/mute/0_gt_wavs/mute{target_sample_rate}.wav|{now_dir}/logs/mute
/3_feature{fea_dim}/mute.npy|{now_dir}/logs/mute/2a_f0/mute.wav.npy|{now_dir}/lo
gs/mute/2b-f0nsf/mute.wav.npy|{speaker_id}"
  )
shuffle(opt)
with open(f"{exp_dir}/filelist.txt", "w") as f:
  f.write("\n".join(opt))
print("Mute filelist written. Best of luck training!")



%cd /content/Mangio-RVC-Fork
%load_ext tensorboard
%tensorboard --logdir /content/Mangio-RVC-Fork/logs


os.chdir('/content/Mangio-RVC-Fork')
!$cmd


#@title Експорт моделі
import os


DATASET_PATH_DRIVE = "/content/drive/MyDrive/rvcDisconnected/" + experiment_name
DATASET_PATH_COLAB = "/content/Mangio-RVC-Fork/logs/" + experiment_name
if(not os.path.exists(DATASET_PATH_DRIVE)):
  !mkdir -p "{DATASET_PATH_DRIVE}"


skip_models = False #@param {type:"boolean"}
```

```
manual_save = False #@param {type:"boolean"}
STEPCOUNT = 000 #@param {type:"integer"}
EPOCHCOUNT = 000 #@param {type:"integer"}

finished=False
potential="/content/Mangio-RVC-Fork/weights/"+experiment_name+".pth"
if os.path.exists(potential):
  finished = True

print("Detecting latest model...")
if(not manual_save):
  currentMax = 0
  for r, _, f in os.walk("/content/Mangio-RVC-Fork/weights/"):
    for name in f:
      if(name.endswith(".pth") and (name!=experiment_name+".pth")):
        if(name.find(experiment_name)==-1):
          continue
        #Determine Epochcount+Stepcount Phase 1
        pot = name.split('_')
        ep=pot[len(pot)-2][1:]
        #If what we got from the epoch count section of the filename isn't a
number, multiple completed models are in weights.
        #Skip it if that happens.
        if(not ep.isdecimal()):
          continue
        #Determine Epochcount+Stepcount Phase 2
        ep=int(ep)
        if ep>currentMax:
          currentMax=ep
          step=pot[len(pot)-1].split('.')
          step=int(step[0][1:])
          EPOCHCOUNT=ep
```

```
          STEPCOUNT=step


TSTEP = STEPCOUNT
if(not skip_models):
  if(save_only_latest_ckpt):
    TSTEP=2333333
  !cp "{DATASET_PATH_COLAB}/D_{TSTEP}.pth" "{DATASET_PATH_DRIVE}"
  !cp "{DATASET_PATH_COLAB}/G_{TSTEP}.pth" "{DATASET_PATH_DRIVE}"
  !cp "{DATASET_PATH_COLAB}/config.json" "{DATASET_PATH_DRIVE}"


print("Copying Tensorboard TFEVENT files...")
for r, d, f in os.walk(DATASET_PATH_COLAB):
  for name in f:
    if(name.startswith("events.out.tfevents") and
os.path.exists(os.path.join(DATASET_PATH_COLAB, name))):
      !cp "{DATASET_PATH_COLAB}/{name}" "{DATASET_PATH_DRIVE}"


print("Copying weight file...")
if(finished):
  !cp "{potential}" "{DATASET_PATH_DRIVE}"
else:
  !cp "/content/Mangio-RVC-
Fork/weights/{experiment_name}_e{EPOCHCOUNT}_s{STEPCOUNT}.pth"
"{DATASET_PATH_DRIVE}"
```

ДОДАТОК Б. ГРАФІЧНІ МАТЕРІАЛИ

ЗАСТОСУВАННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ КЛОНУВАННЯ ТА ГЕНЕРАЦІЇ ГОЛОСУ У РЕАЛЬНОМУ ЧАСІ

ВИКОНАВ: НІКІТАЄВ НІКІТА ВІТАЛІЙОВИЧ

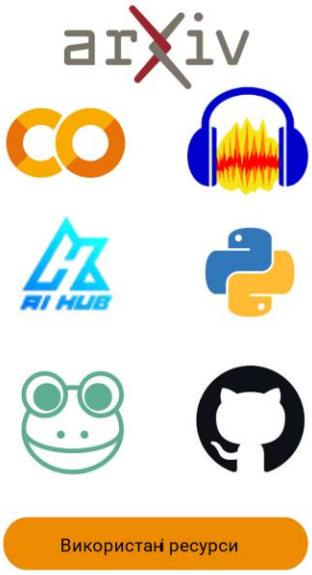
Студент 4-го курсу, групи КА-05

Керівник: канд. фіз.-мат. наук, доцент Барановська Леся Валеріївна

Актуальність

Дослідження у сфері синтезу мовленнярозпочалисяше у 1922му році, а клонуванняголосуіснує щонайменшез 1998року, але швидкий розвиток нейронних мереж покращив якість відтворенняу рекордний час. Наразі, клонуванняголосує перспективноютехнологією у сферах розваг та охорониздоров'я. В той же час, як і більшість новітніх технологій, VC моделі змушують замислитисянад важливими правовимипитаннями, наприклад щодо авторського права на голос. Відкритий доступ до цих технологій також дає зловмисникамзмогу легко користуватисяодним з найефективніших методів маскуванняособистості.

Знання принципуроботи даних технології є важливимуу наш час. На думку автора, інтеграція клонуванняголосув особисте і професійнежиття, призведедо соціальних змін.



Використані ресурси

Мета та об'єкт роботи

Мета роботи - дослідження застосування нейронних мереж для синтезу мовлення та клонування голосу, зокрема у режимі реального часу. Ключова задача полягає у демонстрації прогресу нейронних мереж у обраній сфері, з особливою увагою до моделей на основі пошуку (RVC).

Об'єкти роботи - власний набір даних, що складається з 34 хвилин високоякісного аудіо та попередньо тренована модель "original" з проекту RVC.

Постановка задачі

Поставлена задача - розробка та демонстрація роботи моделі RVC на прикладі власного голосу автора

01 Провести дослідження щодо теорії обробки оцифрованих звукових сигналів. Особливу увагу приділити мел-спектрограмам.

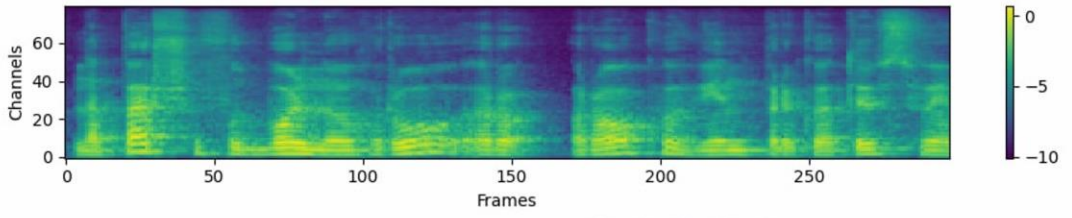
02 Проаналізуватиструктуру сучасних моделей TTS та VC, досягти розуміннярархітектури обраних моделей.

03 Створити власний набір даних Реалізувати тренування моделі у хмарному середовищі.Зробити демонстраціюроботи моделі.

04 Проаналізувати проведену роботу, зробити висновки, як з приводу її виконання, так і з приводу подальшого розвитку досліджених технологій

Мел-спектрограма

У звичайній спектрограмі частотна вісь відповідає лінійній шкалі, що вимірюється в герцах (Гц). Однак слухове сприйняття людини демонструє логарифмічну чутливість, коли нижчі частоти розрізняються гостріше, ніж вищі. Для апроксимації цієї нелінійної чутливості використовується шкала мела - шкала частот сприйняття. Створення мел-спектрограми передбачає використання короткочасного перетворення Фур'є для сегментації звуку на короткі інтервали, таким чином отримуючи серію частотних спектрів, як у стандартній спектрограмі



Мел-спектрограма власного набору даних

Модель TTS

Text-to-speech (текст у голос) - це метод синтезу голосу, який передбачає конвертацію тексту у аудіо

Переваги

Висока точність
Ефективність
Доступність
Масштабованість

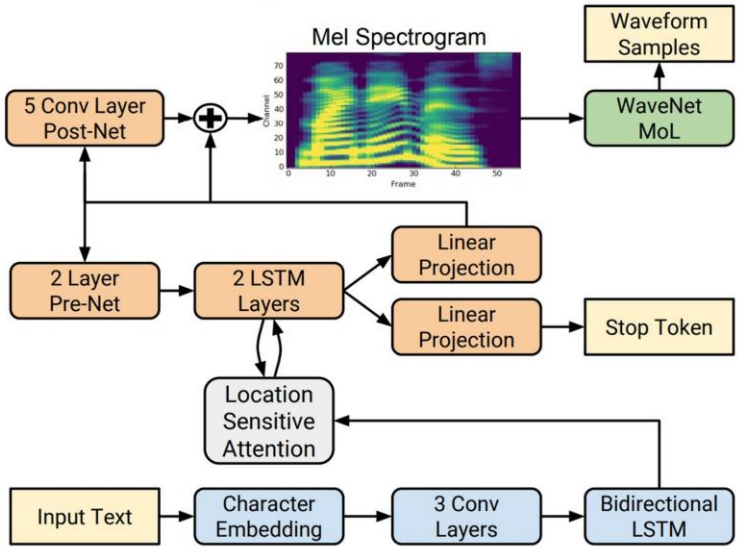
Недоліки

Природність голосу
Відсутність емоційного забарвлення
Паралельне перетворення

06

Діаграма TTS VC

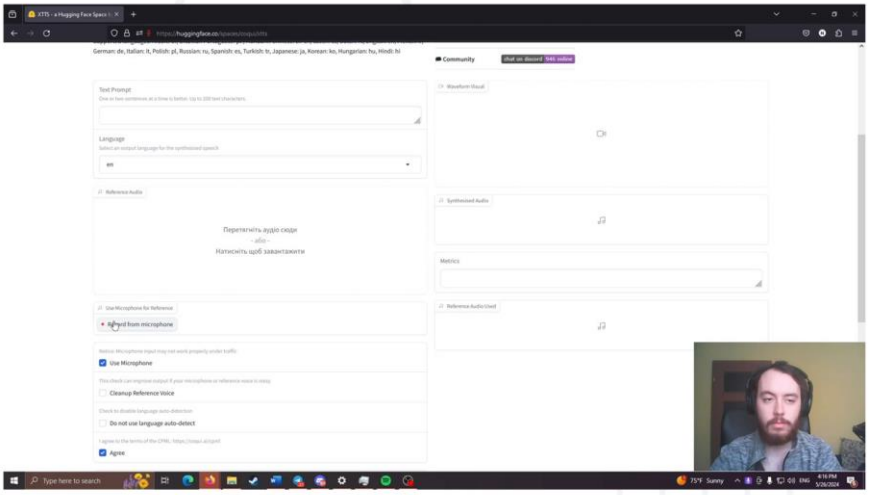
На прикладі Tacotron2



07

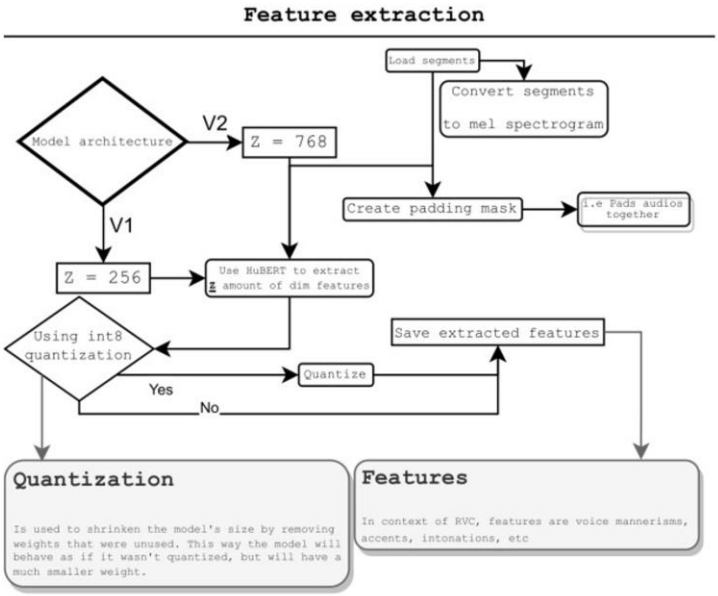
Демонстрації роботи XTTSv2

01



08

Алгоритм вилучення особливостей голосу



09

Модель RVC

Retrieval-based Voice Conversion (клонування голосу на основі пошуку) - це метод синтезу голосу, який передбачає пошук і модифікацію наявних зразків мовлення.

Переваги

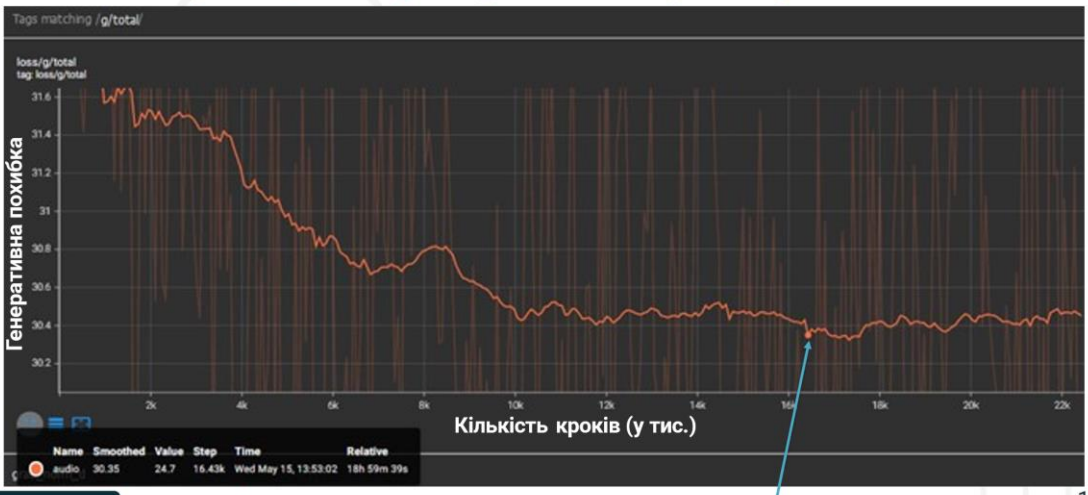
- Висока точність
- Природність
- Ефективність
- Низьке використання ресурсів
- Відсутність потреби у великих наборах даних
- Непаралельне перетворення

Недоліки

- Залежність від pretrain
- Граничні артефакти
- Масштабованість
- Специфіка використання ресурсів

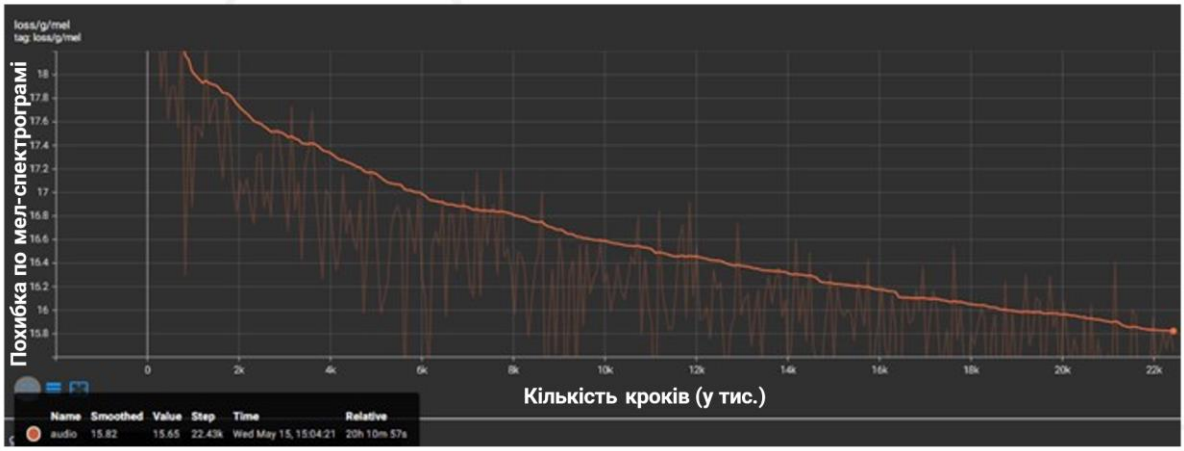
10

Демонстрація графіків тренування



11

Демонстрація графіків тренування



Демонстрації роботи RVC

02

Контрольний запис

Демонстрації роботи RVC Realtime

03

Середня затримка – 720 ms

Висновки

Відповідальність за етичне використання моделі лягає на її творця!

Дана робота присвячена аналізу процесів синтезу мовлення та клонування голосу за допомогою нейронних мереж. У роботі детально розглянуто сучасні методи синтезу та клонування голосу, зокрема у реальному часі, описано потенційні варіанти використання. Безпосередньо створено модель клонування голосу за технологією RVC, наведено демонстрації, досліджено та описано необхідний технологічний стек, переваги та недоліки даної технології.

Отримана модель виявилася результативнішою за першочергові очікування, непогано спрацьовуючи навіть з конвертацією вокалу. Актуальність роботи полягає у дослідженні принципово нової технології клонування голосу, яка потребує лише одного типу даних для виконання поставленої задачі. RVC успішно застосовується у сфері розваг (переважно на YouTube), проте має потенціал використання у сфері охорони здоров'я та для збереження вимираючих мов.

Подальші дослідження можуть полягати у покращенні створеної моделі, шляхом вдосконалення набору даних. Також перспективним напрямом є дослідження MMVC (many -to -many voice conversion) .

ДЯКУЮ

ЗА УВАГУ!