ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»

Факультет комп’ютерно-інформаційних технологій

Кафедра комп’ютерних інформаційних систем і технологій

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Тема «Розробка десктопного застосунку для клонування голосу на основі аудіо зразків з використанням технологій машинного навчання»

Спеціальність: Інженерія програмного забезпечення

Виконав студент групи ІК-9-22-Б1ІПЗ(4.0ддс)-1 Шнітов Іван Валерійович

Керівник: к.т.н., професор

Рецензент: к.т.н., доцент

Допущено до захисту

Завідувач кафедри _____ д.е.н., професор Кавун С.В.

(підпис)

Київ 2026

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці десктопного застосунку для клонування голосу (voice cloning) на основі аудіо зразків з використанням технологій машинного навчання (machine learning). У роботі розглядаються теоретичні основи синтезу мовлення (speech synthesis), аналіз предметної області, проєктування архітектури застосунку, вибір технологій реалізації, таких як Python та бібліотеки для глибокого навчання (deep learning). Запропонований застосунок дозволяє копіювати характеристики голосу з існуючого аудіо файлу та генерувати новий аудіо контент на основі введеного тексту, що має потенціал застосування в освіті, медіа та розважальній індустрії. Робота демонструє етапи створення альфа–версії програмного продукту без повного тренування моделі, з акцентом на інтеграцію готових рішень.

Ключові слова: клонування голосу, синтез мовлення, машинне навчання, глибоке навчання, текст у мовлення,

ABSTRACT

The qualification work is dedicated to the development of a desktop application for voice cloning based on audio samples using machine learning technologies. The work examines the theoretical foundations of speech synthesis, analysis of the subject area, design of the application architecture, selection of implementation technologies such as Python and deep learning libraries. The proposed application allows copying voice characteristics from an existing audio file and generating new audio content based on input text, which has potential applications in education, media, and entertainment industry. The work demonstrates the stages of creating an alpha version of the software product without full model training, with an emphasis on integrating ready–made solutions.

Keywords: voice cloning, speech synthesis, machine learning, Python, deep learning, TTS (Text–to–Speech).

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі швидкого розвитку штучного інтелекту (artificial intelligence, AI) та цифрових технологій, клонування голосу (voice cloning) стає одним з ключових напрямків у сфері обробки аудіо сигналів. Згідно з прогнозами експертів, до 2026 року ринок технологій синтезу мовлення (text–to–speech, TTS) та voice cloning досягне обсягів у мільярди доларів, зважаючи на зростання попиту в галузях медіа, освіти, телекомунікацій та розваг. Наприклад, такі технології вже застосовуються в створенні аудіокниг, віртуальних асистентів (virtual assistants) та персоналізованого контенту, дозволяючи відтворювати голоси відомих осіб або адаптувати мовлення для людей з вадами мовлення.

Актуальність розробки десктопного застосунку для клонування голосу зумовлена кількома факторами. По–перше, зростання доступності інструментів машинного навчання (machine learning, ML) для неспеціалістів, таких як бібліотеки PyTorch або TensorFlow, робить можливим створення персоналізованих рішень без значних обчислювальних ресурсів. По–друге, етичні та практичні виклики, пов'язані з deepfakes (глибокі фейки – технології створення фальшивих медіа за допомогою AI), вимагають розвитку відкритих інструментів для освітніх та конструктивних цілей. У контексті України, де IT–сектор активно розвивається, такі проекти сприяють інтеграції локальних фахівців у глобальні тренди, наприклад, у створенні україномовного контенту з використанням voice cloning для збереження культурної спадщини або дубляжу фільмів.

Крім того, пандемія COVID–19 та перехід до онлайн–освіти підкреслили потребу в інструментах, які дозволяють генерувати аудіо з тексту з персоналізованим голосом, наприклад, для лекцій чи подкастів. За даними досліджень Gartner, до 2025–2026 років понад 50% цифрового контенту буде

створюватися з використанням AI, включаючи voice cloning, що робить тему не лише актуальною, але й перспективною для подальших досліджень. Однак, існуючі комерційні рішення, як ElevenLabs чи Respeecher, часто є платними та обмеженими, тому розробка відкритого десктопного застосунку на базі Python дозволить демократизувати доступ до цих технологій.

Метою кваліфікаційної роботи є розробка альфа–версії десктопного застосунку для клонування голосу на основі аудіо зразків з використанням технологій машинного навчання, з акцентом на опис етапів проєктування, реалізації та тестування без повного тренування моделі.

Для досягнення поставленої мети потрібно виконати наступні завдання:

1. Провести аналіз предметної області, включаючи огляд теоретичних основ voice cloning та існуючих рішень
2. Розробити функціональні та нефункціональні вимоги до застосунку, а також сценарії використання (use cases).
3. Проєктувати архітектуру застосунку, включаючи архітектурний каркас (framework) на базі Python.
4. Обрати технології реалізації, такі як бібліотеки для обробки аудіо (наприклад, Librosa) та моделі глибокого навчання на базі архітектури SV2TTS (speaker encoder GE2E, синтезатор Tacotron, вокодер WaveRNN) як рішення з відкритим кодом.
5. Описати методику тестування та потенційні результати, з фокусом на unit–тестах (одиночні тести) та ручному тестуванні.

Ці завдання дозволять створити концептуальну основу для застосунку, демонструючи потенціал технології voice cloning у простому, доступному форматі.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ТЕОРЕТИЧНІ ОСНОВИ

1.1. Огляд технологій клонування голосу

Клонування голосу є одним із найбільш активно досліджуваних напрямів у сфері штучного інтелекту. Суть цієї технології полягає у відтворенні індивідуальних характеристик людського голосу – таких як тембр, інтонаційні патерни, темп мовлення та акцентуаційні особливості – за допомогою програмних систем. На відміну від традиційного синтезу мовлення, де генерується умовно нейтральний голос диктора, клонування орієнтоване на точну імітацію конкретної людини, що відкриває широкі можливості для персоналізованих рішень.

З точки зору ринкової динаміки, галузь демонструє стрімкий розвиток. Аналітичні компанії фіксують, що обсяг світового ринку технологій клонування голосу у 2025 році перевищив позначку у 2,4 мільярди доларів США [11]. Прогнози на найближчі роки свідчать про подальше зростання: за різними оцінками, до 2026 року показник може сягнути 2,9–3,0 мільярди доларів, а до 2031 року – перевищити 9,5 мільярди при середньорічному темпі зростання близько 25% [12]. Такий стрімкий приріст пояснюється одночасним розширенням як технологічних можливостей, так і практичного попиту у сферах медіавиробництва, освіти, телекомунікацій та індустрії розваг.

Принципово важливим є те, що сучасні технології клонування голосу пройшли значний шлях еволюції від ранніх, доволі примітивних методів. Можна виділити три основні підходи, що застосовувалися на різних етапах розвитку галузі.

Перший – конкатенативний підхід – ґрунтується на поєднанні попередньо записаних фонетичних фрагментів у єдиний аудіопотік. Цей метод забезпечував відносно природне звучання, проте вимагав великої бібліотеки записів та

погано справлявся з нестандартними інтонаціями чи новими словами.

Другий – параметричний підхід – передбачає математичне моделювання голосового апарату людини. Параметри, що описують характеристики джерела звуку та артикуляційного тракту, передаються акустичному синтезатору. Перевагою є компактність і гнучкість, проте синтезований голос нерідко сприймається як механічний та позбавлений природної виразності.

Третій і на сьогодні домінуючий – нейронний підхід – використовує архітектури глибокого навчання для навчання на реальних аудіозаписах і генерації нових звукових послідовностей. Саме нейронні методи забезпечили якісний стрибок у реалістичності синтетичного мовлення, скорочуючи так звану «проблему uncanny valley» – відчуття неприродності, що виникає при майже, але не зовсім реалістичному відтворенні. За оцінками аналітиків, нейронне клонування охоплює близько 64% ринку і демонструє найвищі темпи зростання серед усіх методів.

Говорячи про практичне застосування, варто відзначити кілька ключових сфер. У галузі доступності технологія дозволяє людям, що втратили голос через хворобу чи травму, зберегти персоналізований синтетичний аналог власного мовлення. У медіавиробництві voice cloning застосовується для дубляжу, озвучення аудіокниг, рекламних матеріалів та відеоігор без необхідності залучати акторів для кожної нової фрази. Віртуальні асистенти набувають більшої персоналізованості завдяки можливості «звучати» голосом конкретної людини. Нарешті, у сфері збереження культурної спадщини технологія дозволяє відтворювати голоси відомих осіб з архівних записів низької якості.

Серед провідних комерційних рішень, що визначають стан галузі станом на 2025–2026 роки, можна виділити кілька платформ. ElevenLabs вважається одним із найякісніших інструментів для англомовного ринку: платформа підтримує клонування на основі лише 30 секунд аудіозапису, генерує голос більш ніж 30 мовами і забезпечує реалістичні паузи та зміни інтонації [10]. Resemble AI робить акцент на етичних аспектах і вбудовує приховані цифрові маркери в синтезовані записи для їх подальшої ідентифікації. Fish Audio

вирізняється можливістю роботи з мінімальним зразком – достатньо 1–3 секунд аудіо – і підтримує понад 50 мов та акцентів. Серед рішень із відкритим кодом варто відзначити Coqui XTTS та Bark, що дозволяють розробникам інтегрувати можливості клонування у власні проекти на Python.

Паралельно з технологічним розвитком загострюються питання етичного регулювання. Використання голосових клонів для маніпуляцій, шахрайства або поширення дезінформації спонукає дослідників і регуляторів розробляти стандарти підтвердження згоди, механізми виявлення синтетичного контенту та законодавчі рамки для відповідального застосування технології [20].

1.2. Теоретичні основи клонування голосу

Для розуміння принципів роботи систем клонування голосу необхідно розглянути теоретичну базу, на якій вони побудовані. Ця база охоплює елементи обробки аудіосигналів, математичного моделювання та архітектур нейронних мереж.

Обробка аудіосигналів та екстракція ознак

Будь–яка система клонування голосу починає роботу з аналізу вхідного аудіозапису. Сирий звуковий сигнал – послідовність числових значень амплітуди у часі – є надто складним для безпосереднього використання як вхід нейронної мережі. Тому застосовуються методи перетворення сигналу у компактніші та інформативніші представлення.

Одним із найпоширеніших є Short–Time Fourier Transform (STFT) – короткочасне перетворення Фур'є, що розкладає сигнал на частотні складові в невеликих часових вікнах. Результатом є спектрограма – двовимірне представлення сигналу у координатах «час–частота». Проте оскільки людське слухове сприйняття є нелінійним щодо частоти (людина краще розрізняє низькі частоти), на практиці застосовується мел–шкала – логарифмічна шкала частот, що наближена до особливостей слуху. Побудована на її основі мел–спектрограма (mel–spectrogram) є стандартним проміжним представленням

у більшості сучасних TTS–систем.

Окрім спектрограм, широко застосовуються мел–кепстральні коефіцієнти (MFCC, mel–frequency cepstral coefficients) – компактний набір числових характеристик, що кодує спектральну форму голосового сигналу. MFCC традиційно використовувалися в класичних системах розпізнавання мовлення і досі зустрічаються як додаткові ознаки у нейромережевих системах.

Іншою важливою характеристикою є просодія – сукупність ритмічних та інтонаційних властивостей мовлення, що включає частоту основного тону (pitch), тривалість звуків та гучність. Просодія визначає «емоційне забарвлення» і природність синтетичного голосу, тому її правильне моделювання є критично важливим для переконливого клонування.

Архітектури нейронних мереж у клонуванні голосу

Сучасні системи клонування голосу, як правило, складаються з кількох взаємопов'язаних нейромережевих компонентів, кожен із яких виконує специфічну функцію.

Енкодер диктора (speaker encoder) – відповідає за «розпізнавання» індивідуальних характеристик голосу з аудіозразка та кодування їх у вектор фіксованої розмірності, що отримав назву ембедінг диктора (speaker embedding). Цей вектор є своєрідним «відбитком голосу», що використовується для кондиціювання синтезу. Зазвичай енкодер навчається окремо від інших модулів за завданням верифікації диктора.

Синтезатор (synthesizer) – приймає текст у вигляді послідовності фонем або символів та ембедінг диктора і генерує відповідну мел–спектрограму. Одним із найвпливовіших рішень у цьому класі є Tacotron – архітектура на базі рекурентних мереж типу LSTM з механізмом уваги (attention mechanism), яка навчилась автоматично вирівнювати текст і аудіо без ручної розмітки. Синтезатор (synthesizer) – приймає текст у вигляді послідовності фонем або символів та ембедінг диктора і генерує відповідну мел–спектрограму. Найвпливовішим і широко застосовуваним рішенням у цьому класі є Tacotron – архітектура на базі рекурентних мереж типу LSTM з механізмом уваги (attention

mechanism), яка навчилась автоматично вирівнювати текст і аудіо без ручної розмітки фонем. Механізм уваги дозволяє моделі динамічно визначати, яка частина вхідної текстової послідовності відповідає поточному моменту у спектрограмі, що є принциповою відмінністю від попередніх підходів із ручним вирівнюванням. Саме Tacotron і його варіанти лежать в основі пайплайну SV2TTS (Transfer Learning from Speaker Verification to Multispeaker Text–To–Speech Synthesis), що реалізований у даній роботі.

Вокодер (vocoder) – завершальний компонент, що перетворює мел–спектрограму у хвильову форму (waveform) – сирий аудіосигнал, придатний для відтворення. Класичним підходом є алгоритм Гріфіна–Ліма (Griffin–Lim), проте він дає відчутні артефакти. Серед нейронних вокодерів особливе місце займає WaveRNN – авторегресивна рекурентна архітектура [2], що генерує аудіо вибірку за вибіркою, використовуючи приховані стани RNN для моделювання залежностей між сусідніми вибірками. Незважаючи на авторегресивну природу, WaveRNN досягає генерації, близької до реального часу, завдяки низці архітектурних оптимізацій: розрідженим вагам, подвійній softmax–голові для старших і молодших бітів та ефективному batch–розгортанню. Саме WaveRNN використовується як вокодер у архітектурі SV2TTS, що реалізована в даній роботі.У ряді архітектур застосовується підхід GAN (Generative Adversarial Network) – генеративно–змагальна мережа. Ідея полягає у спільному навчанні двох нейронних мереж: генератора, що намагається синтезувати реалістичне аудіо, та дискримінатора, що намагається відрізнити синтетичний сигнал від реального. Змагаючись між собою, обидві мережі покращуються: генератор стає все більш переконливим, а дискримінатор – все більш чутливим. Цей підхід особливо ефективний для вокодерів, де він дозволяє уникнути розмитості, характерної для методів на основі середньоквадратичної похибки.

Альтернативою GAN є дифузійні моделі (diffusion models) – відносно новий клас генеративних архітектур, що навчаються поступово відновлювати сигнал зі стану шуму. Системи на базі дифузії, зокрема та, що використовується

в Tortoise TTS, демонструють стабільну якість і менш схильні до проблем, типових для GAN (нестабільність тренування, «режим колапсу»). Недоліком є більш повільна генерація, проте це компенсується вищою надійністю результату.

Повне тренування моделей голосового синтезу вимагає великих датасетів і тривалих обчислень. У реальних задачах широко застосовується трансферне навчання (transfer learning): модель спочатку навчається на великій кількості даних (загальна TTS–задача), а потім адаптується до конкретного диктора з використанням невеликої кількості зразків. Саме цей підхід дозволяє сучасним системам клонувати голос з 30–60 секунд аудіо замість годин записів.

На теоретичному рівні активно досліджуються методи виявлення синтетичного мовлення. Аудіо водяні знаки (audio watermarks) – це приховані сигнали, вбудовані в синтезоване аудіо, які практично не впливають на якість звучання, але можуть бути виявлені спеціальним детектором. Паралельно розробляються класифікатори синтетичного мовлення (speech deepfake detectors), що навчаються розпізнавати характерні артефакти клонованих голосів.

1.3. Аналіз існуючих рішень

Аналіз поточного стану ринку засобів клонування голосу дозволяє виявити основні тенденції та визначити нішу для розробки нового програмного рішення. Станом на 2025–2026 роки існуючі продукти можна умовно розподілити на три категорії: комерційні хмарні платформи, спеціалізовані корпоративні рішення та інструменти з відкритим вихідним кодом.

Комерційні хмарні платформи:

ElevenLabs є беззаперечним лідером споживчого сегмента і вважається фактичним стандартом якості для англомовного голосу. Платформа пропонує API та веб–інтерфейс для клонування голосу на основі зразка від 30 секунд до кількох хвилин. Підтримується понад 32 мови і понад 1000 готових синтетичних голосів. Сильною стороною ElevenLabs є висока природність,

включаючи реалістичні паузи, зміни темпу та емоційні відтінки. Обмеженням є платна модель монетизації: безкоштовний рівень суттєво обмежений за обсягом генерації.

Resemble AI позиціонується як рішення для корпоративного використання з акцентом на безпеку та відстежуваність. Платформа вбудовує невидимі цифрові маркери у кожен синтезований запис, що дозволяє підтвердити його походження. Підтримується швидке клонування та налаштування емоційного забарвлення, проте вартість використання висока.

Descript орієнтований насамперед на редагування аудіо та відеоконтенту. Інструмент Overdub дозволяє виправляти помилки у записах, замінюючи окремі слова синтетичним голосом диктора – що особливо зручно для подкастерів та відеовиробників. Проте можливості повноцінного клонування обмежені порівняно з конкурентами.

Fish Audio виділяється серед конкурентів можливістю роботи з надкороткими зразками – достатньо 1–3 секунд вхідного аудіо. Підтримується понад 50 мов і діалектів, а система демонструє хорошу передачу емоційного забарвлення. Це робить Fish Audio привабливим для застосувань у медіа та мультимовних проектах, однак для обробки в реальному часі може знадобитися потужне апаратне забезпечення.

Soundverse AI та аналогічні платформи акцентують на застосуванні в музичній індустрії, де клонування голосу використовується для бек–вокалу, демонстраційних записів та реставрації архівних матеріалів.

Coqui XTTS (eXtended Text–to–Speech) є однією з найвпливовіших бібліотек відкритого доступу для мультимовного клонування голосу. Система підтримує понад 17 мов, включаючи багато європейських, і дозволяє клонувати голос з кількох секунд аудіо. Бібліотека написана на Python і добре інтегрується з PyTorch, що робить її зручною для включення у власні проекти. Основним обмеженням є вимогливість до обчислювальних ресурсів: без GPU час синтезу може бути неприйнятно великим.

Real–Time–Voice–Cloning є відкритою реалізацією архітектури SV2TTS.

Система складається з трьох окремих модулів із попередньо натренованими вагами: speaker encoder на базі GE2E (Generalized End–to–End loss) [4], синтезатор на базі Tacotron [3] та вокодер на базі WaveRNN. Ключовою особливістю є можливість клонування голосу з короткого аудіозразка (від 5 секунд) без будь–якого додаткового тренування – лише на основі кондиціювання через голосовий ембедінг. Попередньо натреновані контрольні точки (pre–trained checkpoints) завантажуються автоматично з HuggingFace при першому запуску, що дозволяє розгорнути повноцінний пайплайн клонування без необхідності тренування з нуля [9]. Саме цей проект обрано як технічну основу для реалізації застосунку у даній роботі.

Bark від компанії Suno – відносно новий open–source інструмент, що підтримує генерацію мовлення зі збереженням емоцій, паразитних звуків («е–е», сміх тощо) і навіть фоновий шум. Це надає синтезованому голосу особливої живості, проте модель є великою і повільною без GPU.

RVC (Retrieval–based Voice Conversion) відрізняється від класичних TTS–підходів: замість перетворення тексту в мовлення, RVC конвертує голос одного диктора у голос іншого, використовуючи механізм пошуку за зразками. Це дозволяє зберігати природні характеристики оригінального запису та є популярним у музичних застосуваннях.

Порівнюючи розглянуті рішення за основними критеріями, можна виявити наступні закономірності. Комерційні платформи забезпечують вищу якість і зручність використання, проте є платними, закритими та орієнтованими на хмарне розгортання. Open–source інструменти надають повний контроль над процесом і можуть працювати локально, але вимагають технічної підготовки для налаштування та потужного обладнання для комфортного використання. Жодне з розглянутих open–source рішень не пропонує зручного десктопного інтерфейсу для кінцевого користувача – це і є основна ніша для розробки застосунку, що розглядається у даній роботі.

1.4. Проблеми та перспективи розвитку

Попри вражаючий прогрес у сфері клонування голосу, технологія стикається з низкою нерозв'язаних проблем, що обмежують її широке впровадження.

Якість і надійність залишаються ключовим викликом. Хоча найкращі комерційні системи демонструють чудові результати на якісних вхідних зразках, якість суттєво знижується при наявності фонового шуму, реверберації або нестандартних акустичних умов запису. Синтезовані голоси нерідко демонструють характерні артефакти у складних фонетичних контекстах, на межах речень або при вимові рідкісних слів.

Обчислювальна вимогливість є серйозним бар'єром для локального розгортання. Повноцінне тренування сучасних голосових моделей потребує кількох GPU та тижнів обчислень, що недоступно для більшості розробників і дослідників. Навіть інференс – генерація аудіо за допомогою вже навченої моделі – може займати від кількох секунд до хвилин на звичайному ноутбуці. Це робить реалістичним лише застосування попередньо натренованих рішень у навчальних проектах.

Упередженість даних (data bias) є систематичною проблемою галузі. Більшість великих датасетів, на яких навчаються системи синтезу мовлення, сформовані з аудіозаписів носіїв стандартних варіантів мов (переважно американського англійського, стандартного мандаринського тощо). Внаслідок цього системи значно гірше справляються з малопоширеними мовами, регіональними діалектами та нестандартними акцентами – що фактично означає технологічне відчуження частини мовної спільноти. Для української мови ця проблема є особливо актуальною, оскільки україномовних датасетів для навчання TTS значно менше, ніж для провідних світових мов.

Емоційна автентичність – ще одна незрілість технології. Синтезувати природну інтонацію нейтрального диктора сучасні системи навчилися досить добре, проте відтворити тонкі емоційні нюанси – іронію, стриману радість, втому – залишається складним завданням. Емоційно насичені голоси нерідко

звучать перебільшено або неприродно.

Етичні та правові виклики є, мабуть, найбільш серйозними з нетехнічних проблем. Технологія клонування голосу має очевидний потенціал для зловживань: від шахрайських дзвінків від імені родичів або керівників до масштабних кампаній дезінформації з фальшивими аудіозаявами публічних осіб. У 2024–2025 роках у багатьох країнах зафіксовано численні випадки телефонного шахрайства з використанням синтетичних голосів. Правове регулювання у більшості юрисдикцій ще тільки формується, що створює правову невизначеність для розробників і користувачів технології [13]. Додатковим ускладненням є питання авторського права на голос: чи є голос людини об'єктом правового захисту, і чи потрібна її згода для створення синтетичного клону.

Незважаючи на зазначені проблеми, перспективи галузі виглядають дуже оптимістично, і дослідницька спільнота активно працює над їх вирішенням.

У технологічному плані очікується подальше зниження вимог до обсягу вхідних даних: вже зараз провідні системи клонують голос з 30 секунд, і ця межа, ймовірно, буде знижена до кількох секунд або навіть до однієї фрази. Оптимізація архітектур і методів дистиляції моделей дозволить суттєво прискорити генерацію, роблячи синтез у реальному часі доступним навіть на мобільних пристроях.

Зростатиме підтримка малопоширених мов завдяки розвитку методів крос–мовного трансферного навчання та цілеспрямованому збору нових датасетів. Для України це відкриває можливість розвитку якісних україномовних голосових технологій – аудіокниг, навчальних матеріалів, асистентів.

В аспекті застосувань очікується активна інтеграція клонування голосу з іншими AI–технологіями: системами автоматичного перекладу, відеогенерацією та аватарами для відеоконференцій. Мультимодальні системи, що одночасно синтезують голос і мімічні рухи обличчя, стануть стандартом для медіавиробництва і персоналізованої комунікації.

У сфері безпеки та довіри прогнозується стандартизація механізмів виявлення синтетичного контенту та законодавче закріплення вимог щодо маркування AI–генерованого аудіо. Розвиток криптографічних методів верифікації автентичності аудіозаписів дозволить ефективніше протидіяти дезінформації.

Загалом, клонування голосу перетворюється з нішевої дослідницької теми на масову технологію, що потребує не лише технічного, а й правового, етичного та соціального осмислення. Розробка десктопних застосунків, що надають доступ до цієї технології широкому колу користувачів, є важливим кроком у напрямі її демократизації – за умови відповідального підходу до питань конфіденційності та використання.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ДЕСКТОПНОГО ЗАСТОСУНКУ ДЛЯ КЛОНУВАННЯ ГОЛОСУ

Визначення вимог до програмного забезпечення є фундаментальним етапом проєктування, оскільки саме на цьому кроці формується чітке розуміння того, що система повинна робити і в яких умовах вона має функціонувати. У розробці програмного забезпечення прийнято розрізняти дві великі категорії вимог: функціональні, що описують конкретну поведінку системи, та нефункціональні, що визначають якісні характеристики її роботи. Обидві категорії однаково важливі: функціональні вимоги визначають, що система робить, а нефункціональні – як добре вона це робить. Ігнорування будь–якої з категорій призводить або до продукту з неповним набором можливостей, або до системи, що технічно виконує свої функції, але є незручною, повільною або ненадійною на практиці.

2.1. Функціональні і нефункціональні вимоги до застосунку

Функціональні вимоги до десктопного застосунку для клонування голосу були сформовані на основі аналізу типових сценаріїв використання технології та огляду аналогічних рішень, проведеного у першому розділі. Нижче наведено деталізований перелік функціональних вимог, згрупованих за логічними модулями системи.

Основою будь–якої системи клонування голосу є можливість завантаження вхідного аудіозапису, що слугує еталоном характеристик голосу.

Застосунок повинен підтримувати імпорт аудіофайлів у щонайменше двох найпоширеніших форматах: WAV (Waveform Audio File Format) та MP3 (MPEG Audio Layer III). Формат WAV є пріоритетним, оскільки він зберігає аудіо без стиснення з втратами, що забезпечує вищу якість вхідних даних для моделі. Підтримка MP3 необхідна з практичних міркувань, оскільки більшість побутових записів зберігається саме в цьому форматі.

Система повинна автоматично перевіряти завантажений файл на відповідність мінімальним вимогам якості. Зокрема, тривалість аудіозразка не повинна бути меншою за 5 секунд – цей поріг продиктований технічними вимогами speaker encoder архітектури SV2TTS, для якого надто короткі зразки не дозволяють сформувати репрезентативний ембедінг диктора. Рекомендованою є тривалість від 30 секунд до 3 хвилин: цього достатньо для якісного клонування без зайвого навантаження на пам'ять. Система також повинна перевіряти частоту дискретизації (sample rate) і при необхідності автоматично виконувати ресемплінг: до 16000 Гц для модуля encoder та до 22050 Гц для модуля vocoder, відповідно до вимог кожного компонента пайплайну SV2TTS.

Додатково у перспективних версіях застосунку доцільно реалізувати можливість запису аудіозразка безпосередньо через мікрофон. Це усунуло б необхідність попередньої підготовки файлу і суттєво спростило б процес для кінцевого користувача.

Застосунок повинен надавати користувачеві зручний інтерфейс для введення тексту, який буде синтезовано клонованим голосом. Текстове поле має підтримувати введення багаторядкового тексту довільної довжини, з підтримкою кирилиці та латиниці. Для коректної обробки тексту модель потребує його попередньої нормалізації: розгортання абревіатур, чисел, дат і символів у їх словесні еквіваленти. Наприклад, рядок «25.12.2024, 14:30» повинен бути перетворений на «двадцять п'яте грудня дві тисячі двадцять четвертого року, чотирнадцята година тридцять хвилин» перед передачею у синтезатор.

Підтримка мовного введення визначається можливостями базової TTS-моделі. У поточній реалізації пайплайну SV2TTS використовується synthesizer на базі Tacotron, навчений на англомовних датасетах, тому система коректно обробляє текст англійською мовою. Розширення підтримки до інших мов, зокрема до української, є перспективним напрямом розвитку і потребує або доналаштування базової моделі на відповідних даних, або заміни модуля synthesizer на мультимовну альтернативу — наприклад, Coqui XTTS v2, що підтримує понад 17 мов. Детальніше цей напрям розглядається у розділі 3.4.

Для зручності роботи корисним доповненням буде функція збереження та завантаження текстових шаблонів – можливість зберігати часто використовувані тексти для повторного застосування без необхідності повторного введення.

Центральна функція застосунку – генерація синтетичного аудіо з урахуванням характеристик завантаженого голосового зразка. Після запуску синтезу система повинна відображати прогрес виконання через відповідний індикатор, оскільки операція може тривати від кількох секунд до кількох десятків секунд залежно від апаратного забезпечення і довжини тексту.

Важливою функцією є можливість налаштування параметрів синтезу перед генерацією. До таких параметрів належать: швидкість мовлення, що дозволяє пришвидшити або сповільнити темп синтезованого голосу; рівень «варіативності» (variability або temperature) – параметр, що впливає на ступінь відхилення від найімовірнішого виходу моделі (нижчі значення дають більш монотонне, але передбачуване мовлення, вищі – більш природне, але з ризиком помилок); а також налаштування просодії (prosody), тобто загального емоційного забарвлення синтезованої фрази.

Система повинна підтримувати повторну генерацію без необхідності повторно завантажувати зразок чи вводити текст – досить натиснути кнопку ще раз, щоб отримати нову варіацію синтезу з тими самими параметрами.

Після завершення генерації система повинна надавати інтерактивний аудіоплеєр для прослуховування результату безпосередньо в застосунку, без необхідності відкривати зовнішні програми. Плеєр повинен підтримувати базові

функції: відтворення, паузу, перемотування та регулювання гучності.

За умови задоволення результатом, користувач повинен мати можливість зберегти згенероване аудіо у файл. Застосунок повинен підтримувати експорт у формати WAV та MP3, з можливістю налаштування якості стиснення для MP3. Також корисною є функція пакетної генерації – обробка кількох різних текстів з одним голосовим профілем послідовно, із збереженням результатів у окрему директорію.

Модуль керування профілями голосів

Для зручності повторного використання система повинна підтримувати збереження голосових профілів – ембедінгів, витягнутих з аудіозразка, – у вигляді файлів на диску. Це дозволить завантажувати раніше збережений голос одним кліком, без повторної обробки вхідного аудіо. Профіль повинен мати ім'я, задане користувачем, і відображатися у бібліотеці збережених голосів.

З огляду на потенціал зловживання технологією, застосунок повинен включати механізм отримання явної згоди користувача перед початком синтезу. При першому запуску система відображає попередження про відповідальне використання та вимагає підтвердження того, що користувач має право використовувати завантажений голосовий зразок. Ця функція не є технічним обмеженням, проте вона формує культуру відповідального застосування і може слугувати елементом доказу добросовісності у разі правових питань.

Нефункціональні вимоги (non–functional requirements) фокусуються на якості та продуктивності системи. Реалізований пайплайн SV2TTS функціонує у середовищі Jupyter Notebook на платформах Windows та Linux. Мінімальні вимоги до апаратного забезпечення: процесор з тактовою частотою не нижче 2 ГГц, не менше 8 ГБ RAM, не менше 3 ГБ вільного місця на диску для розміщення моделей (encoder.pt — 17.1 МБ, synthesizer.pt — 371 МБ, vocoder.pt — 53.8 МБ) і тимчасових файлів. Наявність GPU з підтримкою CUDA від NVIDIA не є обов'язковою, але є вирішальною для прийнятного часу генерації: на конфігурації з GPU RTX 3060 час синтезу типового тексту складає 10–30 секунд (RTF 0.3–0.8), тоді як на CPU час генерації зростає до кількох хвилин.

Якість результату оцінюється за метриками MOS (Mean Opinion Score), SIM (Speaker Similarity) та WER (Word Error Rate) [14, 6]. Надійність забезпечується автоматичною перевіркою наявності файлів моделей при завантаженні та верифікацією їх цілісності через хеш-суми бібліотекою huggingface_hub.

Нефункціональні вимоги визначають якісні характеристики системи і часто мають не менший вплив на практичну цінність продукту, ніж його функціональність. Для десктопного застосунку клонування голосу можна виокремити кілька основних груп нефункціональних вимог.

Для досягнення прийнятного користувацького досвіду час генерації синтетичного аудіо не повинен перевищувати 20 секунд для тексту обсягом до 200 слів при наявності GPU (graphics processing unit). На системах без GPU ліміт може бути розширений до 60 секунд, проте у цьому разі система повинна чітко інформувати користувача про очікуваний час очікування. Завантаження та попередня обробка аудіозразка не повинна займати більше 5 секунд. Запуск самого застосунку (час від відкриття до готовності до роботи) не повинен перевищувати 10 секунд на рекомендованому апаратному забезпеченні.

Застосунок повинен коректно функціонувати на системах з такою мінімальною конфігурацією: процесор з тактовою частотою не нижче 2 ГГц (4+ ядра бажано), оперативна пам'ять не менше 8 ГБ (рекомендовано 16 ГБ), вільне місце на диску не менше 3 ГБ (для розміщення моделей і тимчасових файлів). Наявність GPU з підтримкою CUDA (Compute Unified Device Architecture) від NVIDIA не є обов'язковою, але значно покращує продуктивність і рекомендована для регулярного використання. Застосунок повинен підтримувати операційні системи Windows 10/11 та Linux (Ubuntu 20.04+).

Система повинна коректно обробляти всі передбачувані помилкові ситуації, не завершуючи роботу аварійно. До таких ситуацій належать: спроба завантаження файлу непідтримуваного формату, надто короткий аудіозразок, недостатній обсяг вільної оперативної пам'яті, відсутність доступу до директорії для збереження файлів. У кожному випадку система повинна

відображати зрозуміле повідомлення про помилку з рекомендаціями щодо її усунення. Усі помилки повинні фіксуватися у файлі журналу (log–файлі) з достатньою деталізацією для подальшого діагностування.

Зручність використання (Usability)

Інтерфейс повинен бути організований таким чином, щоб користувач зміг виконати базову операцію – завантажити зразок, ввести текст і отримати результат – протягом не більше 5 хвилин без звернення до документації. Усі основні елементи керування повинні мати підказки (tooltips), що з'являються при наведенні курсора. Критичні дії (наприклад, перезапис збереженого профілю) повинні супроводжуватися діалогом підтвердження.

Оскільки голосові дані є чутливою біометричною інформацією, застосунок повинен обробляти їх виключно локально, без передачі на зовнішні сервери (за винятком режиму хмарної обробки, що має бути явно увімкненим користувачем). Тимчасові аудіофайли, що створюються під час обробки, повинні автоматично видалятися після завершення роботи сесії. Збережені голосові профілі повинні зберігатися у директорії, доступній лише поточному системному користувачу.

2.2. Сценарії варіантів використання

Сценарії варіантів використання (use cases) є одним із центральних інструментів об'єктно–орієнтованого аналізу та проєктування. Вони описують взаємодію зовнішніх учасників (акторів) із системою в контексті досягнення конкретної цілі. На відміну від технічних специфікацій, use cases формулюються з точки зору користувача і дозволяють перевірити, чи справді функціональні вимоги покривають реальні потреби. У даній роботі для опису сценаріїв використовується методологія UML (Unified Modeling Language), що є стандартом у сфері проєктування програмного забезпечення.

Перед описом конкретних сценаріїв необхідно визначити акторів – учасників, що взаємодіють із системою. В даному застосунку можна виокремити два основних актори.

Користувач (User) – основний актор, фізична особа, що безпосередньо взаємодіє з графічним інтерфейсом застосунку. Цей актор є неоднорідним і охоплює різні профілі: розробник або дослідник, що тестує можливості технології; контент–мейкер або подкастер, що використовує застосунок для створення аудіоматеріалів; освітній фахівець, що генерує озвучений навчальний контент. Незважаючи на різницю в мотивації, всі ці підтипи взаємодіють із системою через однаковий інтерфейс.

Система (System) – внутрішній актор, що представляє автоматизовані процеси застосунку: обробку аудіо, взаємодію з ML–моделлю, управління файлами. Виокремлення системи як актора важливо для опису сценаріїв, де певні дії виконуються автоматично без участі користувача.

Детальні сценарії використання:

Сценарій UC–01: Клонування голосу з аудіозразка та генерація мовлення

Мета: Користувач бажає згенерувати аудіо зі своїм текстом, озвучене клонованим голосом з наявного запису.

Передумови: Застосунок запущений і готовий до роботи. Користувач має аудіофайл у форматі WAV або MP3 тривалістю не менше 10 секунд.

Основний сценарій (Basic Flow):

1. Користувач переходить на вкладку «Імпорт аудіо» в головному вікні застосунку.
2. Натискає кнопку «Завантажити файл» або перетягує файл у відповідну область (drag–and–drop).
3. Система перевіряє формат файлу, його тривалість і доступність для читання.
4. Система відображає ім'я файлу, його тривалість та форму хвилі (waveform) у мініатюрному віконці попереднього перегляду.
5. Система у фоновому режимі виконує попередню обробку: ресемплінг до 16000 Гц для модуля encoder через функцію preprocess_wav, нормалізацію амплітуди, видалення тиші, витягування ембедінгу диктора через

encoder.embed_utterance. Отриманий вектор-ембедінг розмірністю 256 зберігається в пам'яті для подальшого кондиціювання синтезатора.

6. Після завершення попередньої обробки у верхній частині вікна з'являється індикатор «Голосовий профіль готовий».
7. Користувач переходить на вкладку «Введення тексту» і вводить бажаний текст у текстове поле.
8. Користувач налаштовує опціональні параметри синтезу: швидкість мовлення, рівень варіативності.
9. Натискає кнопку «Генерувати».
10. Система відображає анімований прогрес–бар і текстовий статус («Завантаження моделі...», «Синтез аудіо...», «Завершення...»).
11. Після завершення генерації автоматично відкривається вкладка «Результат» з аудіоплеєром.
12. Користувач прослуховує згенероване аудіо.

Альтернативний сценарій А1 (файл не відповідає вимогам): На кроці 3 система виявляє, що файл закороткий (менше 10 секунд) або має непідтримуваний формат. Система відображає діалогове вікно з описом проблеми і рекомендацією: наприклад, «Аудіозразок занадто короткий. Рекомендована тривалість – від 30 секунд». Сценарій повертається до кроку 2.

Альтернативний сценарій А2 (відсутність GPU): На кроці 5 або 10 система виявляє, що CUDA-сумісний GPU недоступний (torch.cuda.is_available() повертає False) [15]. Пайплайн автоматично перемикається на виконання на CPU — PyTorch забезпечує цей перехід без змін у коді моделей. Система відображає попередження про значно більший час обробки (від кількох хвилин залежно від довжини тексту). Сценарій продовжується у штатному режимі.

Постумови: Синтезоване аудіо доступне для прослуховування та збереження. Попередньо оброблений ембедінг диктора збережено в тимчасовому кеші для можливих повторних генерацій у рамках поточної сесії.

Сценарій UC–02: Збереження голосового профілю

Мета: Користувач бажає зберегти витягнутий голосовий ембедінг для повторного використання у майбутніх сесіях.

Передумови: Сценарій UC–01 виконаний успішно, система має готовий ембедінг диктора у пам'яті.

Основний сценарій:

1. Після успішної генерації (або після завантаження аудіозразка) користувач натискає кнопку «Зберегти голосовий профіль».
2. Система відображає діалогове вікно з полем для введення імені профілю (наприклад, «Голос Марії»).
3. Користувач вводить ім'я і підтверджує збереження.
4. Система серіалізує ембедінг у файл формату .pkl або аналогічний і зберігає його у стандартній директорії профілів.
5. У бібліотеці голосів з'являється новий запис з іменем профілю, датою збереження та іконкою попереднього перегляду.

Альтернативний сценарій (ім'я вже існує): Якщо профіль із введеним іменем вже існує, система запитує підтвердження перезапису або пропонує ввести інше ім'я.

Сценарій UC–03: Завантаження збереженого голосового профілю

Мета: Користувач бажає скористатися раніше збереженим голосовим профілем без повторного завантаження аудіозразка.

Передумови: У бібліотеці голосів є щонайменше один збережений профіль.

Основний сценарій:

1. Користувач відкриває розділ «Бібліотека голосів» у боковій панелі застосунку.
2. Система відображає список збережених профілів із іменами та датами.
3. Користувач обирає потрібний профіль одним кліком.
4. Система завантажує ембедінг із файлу в оперативну пам'ять.
5. У верхній частині вікна з'являється індикатор «Голосовий профіль завантажено: [ім'я]».

6. Користувач переходить до введення тексту і генерації (кроки 7–12 сценарію UC–01).

Альтернативний сценарій (файл профілю не знайдено або пошкоджено): Система відображає відповідне повідомлення про помилку і пропонує видалити пошкоджений запис з бібліотеки або вказати розташування файлу вручну.

Сценарій UC–04: Експорт синтезованого аудіо

Мета: Користувач бажає зберегти результат генерації у вигляді аудіофайлу на диску.

Передумови: Генерація аудіо виконана успішно (UC–01 завершений).

Основний сценарій:

1. На вкладці «Результат» користувач натискає кнопку «Зберегти аудіо».
2. Система відкриває стандартний системний діалог вибору директорії та імені файлу.
3. У випадаючому меню «Формат файлу» користувач обирає WAV або MP3.
4. Якщо обрано MP3, стає активним додатковий повзунок якості (бітрейт від 64 до 320 кбіт/с).
5. Користувач підтверджує збереження.
6. Система виконує пост–обробку (нормалізація гучності) і зберігає файл у зазначеному місці.
7. З'являється спливне повідомлення (toast notification) «Файл успішно збережено» з посиланням на директорію.

Альтернативний сценарій (помилка запису): Якщо система не може записати файл (наприклад, через відсутність прав доступу), відображається повідомлення про помилку. Система автоматично пропонує зберегти файл у стандартній директорії завантажень користувача.

Сценарій UC–05: Повторна генерація з модифікованими параметрами

Мета: Користувач не задоволений результатом першої генерації і бажає змінити параметри для отримання кращого результату.

Передумови: Перша генерація виконана (UC–01 завершений). Ембедінг диктора знаходиться в кеші.

Основний сценарій:

1. На вкладці «Результат» користувач натискає кнопку «Редагувати параметри».
2. Система повертає користувача на вкладку налаштувань із попередньо збереженими значеннями.
3. Користувач змінює параметри: наприклад, зменшує швидкість мовлення або підвищує рівень варіативності.
4. Натискає «Генерувати знову».
5. Оскільки ембедінг вже знаходиться в кеші, пропуск кроку попередньої обробки зменшує час генерації.
6. Новий результат відображається поряд із попереднім, що дозволяє їх порівняти.
7. Користувач обирає кращий варіант або продовжує експериментувати.

2.3. Проєктування архітектури застосунку

Проєктування архітектури є одним із найвідповідальніших етапів розробки програмного забезпечення. Саме на цьому кроці закладаються рішення, що визначатимуть не лише поточну функціональність системи, а й її здатність до подальшого розвитку, масштабування та підтримки. Погано спроєктована архітектура перетворює кожне нове вдосконалення на дорогу і ризиковану операцію, тоді як продумана структура дозволяє гнучко реагувати на нові вимоги. У контексті десктопного застосунку для клонування голосу архітектурне рішення повинно враховувати специфіку задачі: необхідність інтеграції важких ML–моделей, асинхронну природу тривалих обчислень, потребу в чіткому розмежуванні відповідальності між компонентами та забезпечення можливості заміни базової моделі без переробки всієї системи.

2.3.1 Проєктування архітектурного каркасу

В основу архітектури застосунку покладено лінійний трирівневий

пайплайн (pipeline), що відповідає структурі архітектури SV2TTS (Transfer Learning from Speaker Verification to Multispeaker Text–To–Speech Synthesis)[1]. На відміну від класичних багатошарових підходів, де логіка розподілена між рівнями представлення, контролерами і даними, пайплайн клонування голосу має чітко виражену послідовну природу: кожен компонент отримує результат попереднього і передає свій вихід наступному. Це визначає архітектурне рішення – три незалежних модулі з чітко визначеними інтерфейсами між ними, що разом реалізують повний цикл від аудіозразка до синтезованого мовлення.

Перший модуль – encoder – відповідає за аналіз голосового зразка і кодування його індивідуальних характеристик у компактний числовий вектор. На вхід модуль отримує аудіосигнал у форматі WAV або MP3, виконує його попередню обробку (ресемплінг до 16 кГц, нормалізацію амплітуди, видалення тиші) і передає через нейронну мережу speaker encoder, навчену за алгоритмом GE2E (Generalized End–to–End loss) [4]. На виході – вектор–ембедінг фіксованої розмірності 256, що є «відбитком голосу» і використовується для кондиціювання наступного модуля.

Алгоритм GE2E навчає encoder таким чином, щоб ембедінги одного і того ж диктора були максимально близькими між собою у просторі, тоді як ембедінги різних дикторів – максимально віддаленими. Це досягається через функцію втрат, що одночасно максимізує косинусну подібність всередині групи одного диктора і мінімізує подібність між групами різних дикторів. Завдяки цьому підходу ембедінг стає стабільним і репрезентативним навіть при варіації умов запису – різному фоновому шумі, темпі мовлення чи емоційному забарвленні.

Попередня обробка аудіо у межах модуля encoder реалізована через функцію preprocess_wav, яка послідовно виконує ресемплінг вхідного сигналу до стандартної частоти 16000 Гц, нормалізацію амплітуди та видалення незначущих ділянок тиші на початку і в кінці запису. Обчислення мел–спектрограми для передачі у нейронну мережу виконується через функцію wav_to_mel_spectrogram з параметрами, що відповідають умовам навчання

моделі: 40 мел–смуг, вікно 25 мс, крок 10 мс.

Другий модуль – **synthesizer** – реалізує перетворення тексту у мел–спектрограму з урахуванням характеристик цільового голосу. Модуль на базі архітектури Tacotron [3] приймає одночасно текстову послідовність і голосовий ембедінг від першого модуля. Механізм уваги (attention mechanism) забезпечує вирівнювання між текстом і спектрограмою без ручної фонетичної розмітки. На виході – двовимірна мел–спектрограма (80 мел–смуг), що кодує часово–частотні характеристики синтезованого мовлення.

Кондиціювання синтезатора на голосовий ембедінг є ключовим механізмом, що забезпечує клонування: ембедінг конкатенується з прихованим станом декодера на кожному кроці генерації спектрограми, завдяки чому модель «знає», який голос потрібно відтворити, незалежно від змісту тексту. Цей підхід дозволяє використовувати одну навчену модель для необмеженої кількості дикторів без будь–якого додаткового тренування – достатньо надати аудіозразок і отримати відповідний ембедінг.

Взаємодія з модулем synthesizer реалізована через клас Synthesizer, що завантажує попередньо натреновані ваги і надає метод synthesize_spectrograms. Метод приймає список текстів і список відповідних ембедінгів у вигляді батчу, що дозволяє обробляти кілька запитів одночасно для підвищення ефективності використання GPU.

Третій модуль – vocoder – перетворює мел–спектрограму у відтворюваний аудіосигнал. Модуль на базі WaveRNN генерує аудіовибірки авторегресивно, використовуючи спектрограму як умову. На виході – аудіосигнал з частотою дискретизації 22050 Гц у форматі PCM, придатний для безпосереднього відтворення або збереження у файл.

WaveRNN генерує кожну вибірку окремо, умовно на всіх попередніх вибірках і на відповідному кадрі спектрограми. Для прискорення цього процесу застосовується техніка upsampling: спектрограма просторово розтягується до розмірності аудіосигналу через мережу UpsampleNetwork, що складається з ResNet–блоків. Це дозволяє уникнути повторного обчислення ознак

спектрограми на кожному кроці генерації і суттєво прискорює процес.
Завантаження модуля vocoder виконується через функцію load_model, а синтез аудіо – через infer_waveform, яка приймає мел–спектрограму і повертає одновимірний масив аудіовибірок. Отриманий сигнал додатково нормалізується за амплітудою перед збереженням, щоб гарантувати відсутність кліпінгу і стандартний рівень гучності вихідного файлу.

Організація пайплайну і середовище виконання:

Взаємодія між модулями організована послідовно і реалізована у середовищі Jupyter Notebook, де кожен етап пайплайну відповідає окремій клітинці з коментарями українською мовою. Такий підхід забезпечує прозорість процесу: на кожному кроці можна спостерігати проміжні результати – форму хвилі після обробки, вектор ембедінгу, згенеровану спектрограму – що є важливим як для розуміння роботи системи, так і для діагностики можливих проблем.

Середовище Jupyter Notebook обрано як основне робоче середовище з кількох причин. По–перше, воно забезпечує інтерактивність: кожну клітинку можна виконати окремо і одразу побачити результат – відтворити аудіо, відобразити графік спектрограми, перевірити розмірність ембедінгу. По–друге, notebook дозволяє органічно поєднувати виконуваний код із текстовими поясненнями у форматі Markdown, що робить його одночасно і інструментом розробки, і документацією реалізації. По–третє, Jupyter Lab підтримує виконання на GPU через стандартний PyTorch CUDA–інтерфейс без будь–яких додаткових налаштувань, що критично важливо для прийнятного часу генерації.

Попередньо натреновані ваги всіх трьох модулів (encoder.pt, synthesizer.pt, vocoder.pt) завантажуються автоматично з репозиторію HuggingFace при першому запуску через функцію ensure_default_models і зберігаються локально у директорії saved_models/default/. Загальний обсяг завантажених моделей складає близько 440 МБ.

2.4. Проєктування інтерфейсу користувача

Якість інтерфейсу користувача безпосередньо визначає ефективність роботи із застосунком. При проєктуванні потенційного десктопного інтерфейсу доцільно дотримуватися чотирьох ключових UX-принципів: мінімалізму (відображення лише елементів, необхідних для поточного кроку), відповідності реальному робочому потоку (завантаження зразка → введення тексту → генерація → збереження), негайного зворотного зв'язку на кожну дію користувача та запобігання помилкам (деактивація недоступних елементів до виконання попередніх кроків).

Головне вікно потенційного застосунку організовано за принципом горизонтального розподілу: вузька ліва панель навігації (≈200 пікселів) із переходами між розділами «Голосові профілі», «Новий синтез» та «Налаштування», і правий робочий простір, що змінює вміст залежно від обраного розділу.

Центральний екран «Новий синтез» реалізований як вертикальна послідовність кроків: зона drag-and-drop для завантаження аудіозразка з відображенням хвильової форми після імпорту; текстове поле з лічильником символів та вибором мови; розділ розширених параметрів (швидкість мовлення, варіативність), прихований за принципом прогресивного розкриття; кнопка «Генерувати» з індикатором прогресу та можливістю скасування; блок результату з вбудованим аудіоплеєром і кнопками збереження.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ДЕСКТОПНОГО ЗАСТОСУНКУ ДЛЯ КЛОНУВАННЯ ГОЛОСУ

3.1. Вибір технологій

Вибір технологічного стеку є одним із найважливіших рішень на етапі реалізації програмного продукту. Від нього залежить не лише складність розробки, а й продуктивність готового застосунку, його сумісність із цільовими платформами та можливість подальшого розширення. У контексті реалізації пайплайну клонування голосу вибір технологій ускладнюється специфічними вимогами: необхідністю ефективної роботи з аудіоданими, інтеграцією важких ML–моделей, забезпеченням прийнятного часу генерації на наявному апаратному забезпеченні та зручністю відтворення і демонстрації результатів. Нижче детально обґрунтовано вибір кожної технології за окремими функціональними напрямами.

Мова програмування: Python 3.12

Головною мовою реалізації обрано Python версії 3.12. Ця мова є де–факто стандартом у сфері машинного навчання та наукових обчислень, що зумовлено кількома чинниками.

По–перше, Python має найбагатшу екосистему бібліотек для задач штучного інтелекту та обробки сигналів. Практично кожен значущий інструмент у цій галузі – від фреймворків глибокого навчання до спеціалізованих аудіобібліотек – або написаний на Python, або має Python–інтерфейс. Це різко знижує вартість інтеграції сторонніх компонентів і дозволяє зосередитись на логіці застосунку, а не на низькорівневих деталях

взаємодії між бібліотеками.

По–друге, Python забезпечує кросплатформенність без суттєвих зусиль з боку розробника. Один і той самий код, написаний і налагоджений під Windows, як правило, без змін або з мінімальними адаптаціями запрацює на Linux і macOS. Для проекту, де відтворюваність результатів є важливою вимогою, це є вагомою перевагою.

По–третє, Python пропонує зручні можливості для прискореного прототипування та дослідження. Лаконічний синтаксис, динамічна типізація та інтерактивні середовища виконання (Jupyter Notebook) дозволяють швидко перевіряти ідеї і переходити від концепту до робочого коду. Для навчального та дослідницького проекту, яким є бакалаврська робота, це особливо важливо: можливість виконати окремий крок пайплайну і миттєво побачити його результат у вигляді графіка або відтвореного аудіо суттєво прискорює розробку і полегшує розуміння роботи системи.

Слід також зазначити відому слабкість Python – відносно низьку продуктивність «чистого» коду порівняно зі скомпільованими мовами, як C++ або Rust. Однак у контексті даного застосунку це обмеження фактично не є проблемою, оскільки обчислювально інтенсивні операції – синтез аудіо, операції з тензорами, ресемплінг – виконуються у бібліотеках, написаних на C або C++ і скомпільованих з оптимізаціями. Python виступає лише організаційним «клеєм», що координує роботу цих компонентів.

Фреймворк глибокого навчання: PyTorch 2.x

Для реалізації та виконання ML–моделей обрано PyTorch версії 2.x [5]. PyTorch є одним із двох провідних фреймворків глибокого навчання поряд із TensorFlow/Keras, і обидва мають широке застосування у промисловості та науці. Проте для даного проекту PyTorch є переважнішим вибором з таких причин.

Динамічний обчислювальний граф (dynamic computational graph) PyTorch дозволяє будувати і модифікувати нейронні мережі «на льоту», під час

виконання програми. На відміну від статичного підходу TensorFlow 1.x, де граф компілювався заздалегідь, динамічна модель значно зручніша для налагодження і дослідження. Це особливо важливо при роботі з чужими моделями, де потрібно розібратися у їх внутрішній роботі, відстежити форму тензорів на кожному шарі та ідентифікувати джерело можливих помилок.

Домінування PyTorch у дослідницькій спільноті є ще одним вагомим аргументом. Більшість сучасних публікацій у сфері синтезу мовлення та клонування голосу супроводжуються реалізацією саме на PyTorch. Репозиторій Real–Time–Voice–Cloning, що є технічною основою даної роботи, повністю побудований на PyTorch, включаючи всі три модулі пайплайну: encoder, synthesizer і vocoder. Це означає, що код без додаткових перетворень інтегрується у проект.

Підтримка CUDA і прискорення на GPU є принциповою вимогою для прийнятного часу генерації. PyTorch забезпечує прозору підтримку виконання обчислень на GPU через CUDA. Переведення моделі і даних на GPU зводиться до виклику методу .to('cuda') або .cuda(), що суттєво спрощує реалізацію порівняно з ручним управлінням GPU–пам'яттю. При відсутності GPU PyTorch автоматично перемикається на виконання на CPU без жодних змін у коді моделі – змінюється лише пристрій виконання, але не логіка. У даній роботі використовується відеокарта NVIDIA RTX 3060 з 12 ГБ відеопам'яті, що забезпечує комфортний час генерації аудіо в межах 10–30 секунд для типових текстів.

Важливою перевагою PyTorch 2.x є механізм torch.compile, введений у цій версії, що дозволяє компілювати моделі для прискорення виконання без зміни коду. Хоча у даному проекті ця можливість не застосовується безпосередньо, вона є перспективним напрямом оптимізації для майбутніх версій.

Архітектура SV2TTS: encoder, synthesizer, vocoder

Основою реалізації є репозиторій Real–Time–Voice–Cloning, що містить повну реалізацію архітектури SV2TTS з попередньо натренованими моделями.

Репозиторій організований у вигляді трьох незалежних Python–пакетів, кожен з яких відповідає одному модулю пайплайну.

Пакет encoder реалізує speaker encoder на базі алгоритму GE2E. Ключовими модулями є encoder/inference.py, що надає публічний API для завантаження моделі і обчислення ембедінгів, та encoder/audio.py, що містить функції попередньої обробки аудіо – preprocess_wav для нормалізації і ресемплінгу та wav_to_mel_spectrogram для обчислення мел–спектрограми як входу нейронної мережі. Попередньо натреновані ваги encoder (encoder.pt, розмір 17.1 МБ) завантажуються з HuggingFace і дозволяють відразу отримувати якісні голосові ембедінги без будь–якого додаткового тренування.

Пакет synthesizer реалізує синтезатор на базі Tacotron. Публічний API надається через клас Synthesizer у synthesizer/inference.py, що інкапсулює завантаження моделі і метод synthesize_spectrograms для батчевої генерації мел–спектрограм. Попередньо натреновані ваги synthesizer (synthesizer.pt, розмір 371 МБ) є найбільшим компонентом пайплайну і завантажуються автоматично. Synthesizer виконується на GPU при його наявності, що забезпечується автоматичним визначенням пристрою через torch.cuda.is_available().

Пакет vocoder реалізує WaveRNN vocoder. API надається через vocoder/inference.py з функціями load_model і infer_waveform. Модель WaveRNN (vocoder.pt, розмір 53.8 МБ) є відносно компактною порівняно з synthesizer і забезпечує швидку генерацію аудіосигналу з мел–спектрограми. Архітектура моделі реалізована у vocoder/models/fatchord_version.py і включає мережу UpsampleNetwork для просторового узгодження спектрограми з аудіосигналом та двошарову GRU–мережу для авторегресивної генерації вибірок.

Управління залежностями: uv і pyproject.toml

Для управління залежностями проекту використовується інструмент uv від компанії Astral – сучасний менеджер пакетів для Python, написаний на Rust. На

відміну від традиційного pip, uv значно швидше розв'язує і встановлює залежності завдяки паралельному завантаженню і кешуванню.

Конфігурація залежностей зберігається у файлі pyproject.toml, що є сучасним стандартом для Python–проектів згідно зі специфікацією PEP 517/518 і замінює застарілий формат requirements.txt.

Файл pyproject.toml описує як основні залежності проекту (PyTorch, librosa, soundfile, numpy тощо), так і опціональні групи залежностей – зокрема, група cuda для встановлення версій бібліотек із підтримкою GPU. Це дозволяє одною командою uv sync —extra cuda встановити повне середовище виконання з усіма необхідними залежностями, включаючи правильну версію PyTorch для наявної версії CUDA.

Обробка аудіо: librosa і soundfile

Для аналізу і попередньої обробки аудіосигналів застосовується librosa – одна з найвідоміших Python–бібліотек для роботи зі звуком. Librosa надає зручний API для широкого спектру операцій: завантаження аудіофайлів різних форматів, ресемплінгу до цільової частоти дискретизації, обчислення спектрограм і MFCC, видалення тиші та аналізу характеристик сигналу [7, 16].
У контексті даного проекту librosa виконує такі конкретні функції: завантаження аудіофайлу з автоматичним визначенням частоти дискретизації через librosa.load з параметром sr=None; ресемплінг до стандартних 16 кГц для encoder через librosa.resample з іменованими параметрами orig_sr і target_sr; обчислення мел–спектрограми для візуалізації вхідного сигналу через librosa.feature.melspectrogram з параметрами y і sr; видалення тиші через librosa.effects.trim з налаштованим пороговим значенням.

Варто зазначити, що librosa зазнала суттєвих змін API у версіях 0.9 і вище: ряд функцій перейшов від позиційних аргументів до іменованих, а деякі застарілі функції були видалені. Зокрема, функція librosa.resample у нових версіях приймає аргументи виключно як іменовані (orig_sr, target_sr), а librosa.feature.melspectrogram вимагає передачі аудіосигналу через параметр y=

замість позиційного аргументу [16]. При роботі з репозиторієм, що орієнтований на старіші версії librosa, необхідно враховувати ці зміни і за потреби адаптувати виклики функцій у вихідному коді модулів.

Для читання і запису аудіофайлів у форматі WAV використовується бібліотека soundfile – легковагий інструмент із чистим API, що повертає аудіодані безпосередньо у вигляді NumPy–масивів. Функція soundfile.write використовується для збереження синтезованого аудіо у файл після завершення пайплайну [17].

Обробка числових даних: NumPy

NumPy є фундаментальною бібліотекою для числових обчислень у Python і використовується на всіх етапах пайплайну як стандартний формат обміну даними між компонентами. Аудіосигнали представляються як одновимірні масиви NumPy типу float32, мел–спектрограми – як двовимірні масиви, голосові ембедінги – як одновимірні масиви розмірності 256.

Інтеграція між NumPy і PyTorch є безшовною: масиви NumPy конвертуються у тензори PyTorch через torch.from_numpy без копіювання пам'яті, а результати роботи моделей конвертуються назад через метод .numpy(). Це забезпечує ефективне переміщення даних між компонентами пайплайну без зайвих витрат пам'яті.

Слід зазначити важливий аспект сумісності: NumPy 2.0 видалив ряд застарілих функцій, що існували у попередніх версіях виключно як псевдоніми. Зокрема, функція np.cumproduct була видалена на користь np.cumprod. Репозиторій Real–Time–Voice–Cloning у файлі vocoder/models/fatchord_version.py використовує застарілу назву np.cumproduct, що спричиняє AttributeError при використанні NumPy 2.x. Виправлення зводиться до заміни виклику на актуальний np.cumprod – функціонально ідентичні, різниця лише у назві.

Декодування аудіо: ffmpeg

Окремою системною залежністю є ffmpeg – мультиплатформенний

інструмент командного рядка для роботи з аудіо і відео. Бібліотека librosa використовує ffmpeg як бекенд для декодування аудіофайлів у форматах, що не підтримуються нативно (зокрема, MP3). Без встановленого ffmpeg спроба завантажити MP3–файл через librosa.load завершиться помилкою.

На Windows ffmpeg встановлюється окремо від Python–залежностей: завантажується бінарний дистрибутив з офіційного сайту або встановлюється через пакетний менеджер winget командою winget install ffmpeg. Після встановлення директорія з виконуваними файлами ffmpeg повинна бути додана до системної змінної PATH, щоб librosa могла знаходити виконуваний файл автоматично. Перевірку коректності встановлення можна виконати командою ffmpeg –version у терміналі.

Середовище виконання: Jupyter Lab

Для інтерактивного виконання і демонстрації пайплайну обрано Jupyter Lab – сучасне веб–середовище для роботи з Jupyter Notebook. На відміну від класичного Jupyter Notebook, Jupyter Lab надає повноцінне середовище розробки з файловим менеджером, вкладками, підтримкою Markdown і зручними інструментами навігації по коду.

Jupyter Lab запускається командою jupyter lab з директорії проекту, після чого автоматично відкривається у браузері. Файл notebook (demo.ipynb), розміщений у кореневій директорії репозиторію, стає доступним у файловому менеджері і може бути відкритий для інтерактивного виконання. Кожна клітинка notebook виконується незалежно командою Shift+Enter, що дозволяє покроково проходити пайплайн і спостерігати проміжні результати.

Важливою перевагою Jupyter Lab є вбудована підтримка відображення аудіо і графіків безпосередньо у клітинках notebook. Функція IPython.display.Audio дозволяє відтворювати аудіофайли прямо у браузері без необхідності відкривати зовнішній плеєр, а matplotlib забезпечує відображення графіків форми хвилі, мел–спектрограм і векторів ембедінгів як статичних зображень у клітинках. Це робить notebook не лише інструментом виконання, а

й інтерактивною документацією пайплайну, де текстові пояснення, код і результати розміщені поруч.

Завантаження моделей: HuggingFace Hub

Для автоматичного завантаження попередньо натренованих моделей використовується бібліотека huggingface_hub та репозиторій CorentinJ/SV2TTS на платформі HuggingFace [8]. Функція ensure_default_models, реалізована у utils/default_models.py, перевіряє наявність файлів моделей у локальній директорії saved_models/default/ і за їх відсутності автоматично завантажує їх з HuggingFace.

Загальний обсяг завантажуваних моделей складає близько 442 МБ: encoder.pt – 17.1 МБ, synthesizer.pt – 371 МБ, vocoder.pt – 53.8 МБ. Завантаження виконується один раз при першому запуску; при наступних запусках моделі завантажуються з локального диску, що забезпечує швидкий старт без необхідності підключення до мережі. Бібліотека huggingface_hub автоматично верифікує цілісність завантажених файлів через хеш–суми, що гарантує коректність моделей навіть при перерваному завантаженні.

Візуалізація: Matplotlib

Для візуалізації проміжних і кінцевих результатів пайплайну використовується бібліотека matplotlib – стандартний інструмент для побудови графіків у Python. У контексті даного проекту matplotlib застосовується для відображення форми хвилі (waveform) вхідного аудіозразка і синтезованого результату, мел–спектрограми вхідного сигналу і згенерованої спектрограми, а також вектора голосового ембедінгу у вигляді стовпчикової діаграми.

Відображення спектрограм реалізується через підмодуль librosa.display, що надає зручну функцію specshow для візуалізації двовимірних спектральних даних з правильним масштабом осей часу і частоти. Порівняльні графіки, що відображають оригінальний і синтезований аудіосигнал поруч, реалізуються через matplotlib.pyplot.subplots з кількома підграфіками. Усі графіки

зберігаються у PNG–файли через plt.savefig для подальшого включення до документації.

3.2. Реалізація архітектурного каркасу

Реалізація архітектурного каркасу є практичним втіленням архітектурних рішень, описаних у розділі 2.3. На цьому етапі абстрактна схема компонентів перетворюється на конкретну структуру модулів, файлів і функцій. Каркас визначає «скелет» застосунку: яким чином пов'язані між собою компоненти пайплайну, як організований потік даних від завантаження аудіозразка до отримання синтезованого результату і як система реагує на помилки та нестандартні ситуації.

Файлова структура проекту

Файлова структура проекту організована відповідно до трирівневого пайплайну SV2TTS і відображає природний розподіл відповідальності між модулями. Кореневий рівень репозиторію містить такі компоненти:

```
Real-Time-Voice-Cloning/
│
├── encoder/              # Модуль кодування голосу (GE2E)
│   ├── inference.py      # Публічний API: load_model, embed_utterance
│   ├── audio.py          # Обробка аудіо: preprocess_wav, wav_to_mel_spectrogram
│   ├── params_data.py    # Параметри обробки аудіо
│   ├── params_model.py   # Параметри архітектури моделі
│   └── models/
│       └── lstm.py       # Архітектура LSTM speaker encoder
│
├── synthesizer/          # Модуль синтезу спектрограми (Tacotron)
│   ├── inference.py      # Клас Synthesizer: synthesize_spectrograms
│   ├── models/
│   │   └── tacotron.py   # Архітектура Tacotron
```

```
│   └── utils/
│       ├── text.py          # Обробка тексту: text_to_sequence
│       ├── symbols.py       # Набір символів для синтезатора
│       ├── cleaners.py      # Нормалізація тексту
│       └── numbers.py       # Розгортання числівників
│
├── vocoder/                 # Модуль синтезу аудіо (WaveRNN)
│   ├── inference.py         # Публічний API: load_model, infer_waveform
│   └── models/
│                └── fatchord_version.py  # Архітектура WaveRNN і
UpsampleNetwork
│
├── utils/                   # Допоміжні утиліти
│   ├── default_models.py    # Завантаження моделей з HuggingFace
│   └── argutils.py          # Утиліти командного рядка
│
├── saved_models/            # Директорія збережених моделей
│   └── default/
│       ├── encoder.pt       # Ваги speaker encoder (17.1 МБ)
│       ├── synthesizer.pt   # Ваги Tacotron synthesizer (371 МБ)
│       └── vocoder.pt       # Ваги WaveRNN vocoder (53.8 МБ)
│
├── samples/                 # Вбудовані аудіозразки для тестування
│
├── demo.ipynb               # Jupyter Notebook — основний файл реалізації
├── demo_cli.py              # Альтернативний CLI-інтерфейс
├── demo_toolbox.py          # Альтернативний PyQt5 GUI
└── pyproject.toml           # Конфігурація залежностей (uv)
```

Така структура дотримується принципу єдиної відповідальності на рівні пакетів: кожна директорія верхнього рівня містить код, що відноситься до однієї

чітко визначеної задачі пайплайну. Це полегшує навігацію по кодовій базі, спрощує розуміння ролі кожного компонента і мінімізує взаємні залежності між модулями.

Основним файлом реалізації є demo.ipynb — Jupyter Notebook, розміщений у кореневій директорії репозиторію. Саме цей файл містить повний пайплайн клонування голосу, реалізований покроково з коментарями українською мовою і проміжними візуалізаціями результатів кожного етапу.

Реалізація модуля encoder

Модуль encoder надає два ключових публічних інтерфейси через файл encoder/inference.py: функцію load_model для завантаження попередньо натренованих ваг і функцію embed_utterance для обчислення голосового ембедінгу.

Функція load_model(weights_fpath) завантажує збережену модель з файлу .pt у пам'ять і переводить її на доступний пристрій виконання (GPU або CPU). Стан моделі зберігається у глобальній змінній модуля _model, що реалізує неявний патерн Singleton — модель завантажується один раз і перевикористовується для всіх наступних обчислень ембедінгів без повторного читання з диску.

Функція embed_utterance(wav, using_partials=True) є основним методом модуля і реалізує повний ланцюжок обчислення ембедінгу. При вхідному сигналі, коротшому за мінімальний розмір (визначається параметром min_coverage), сигнал доповнюється нулями до необхідної довжини. Далі сигнал розбивається на перекриваючі фрагменти (partials) з кроком, що визначається глобальним параметром partials_n_frames. Для кожного фрагмента обчислюється мел-спектрограма через wav_to_mel_spectrogram, після чого батч спектрограм передається у нейронну мережу через embed_frames_batch. Отримані ембедінги фрагментів усереднюються і нормалізуються до одиничної довжини, утворюючи фінальний 256-вимірний вектор.

Нормалізація ембедінгу до одиничної довжини (L2-нормалізація) є принциповою деталлю: вона гарантує, що косинусна подібність між

ембедінгами є коректною метрикою відстані і що результати порівнянні між собою незалежно від тривалості вхідного зразка. Норма ембедінгу, отриманого від embed_utterance, завжди дорівнює 1.0.

Попередня обробка аудіо реалізована у файлі encoder/audio.py. Функція preprocess_wav(fpath_or_wav, source_sr=None) є універсальним препроцесором, що приймає як шлях до файлу, так і вже завантажений масив NumPy. При передачі шляху функція автоматично завантажує аудіо через librosa. Якщо вхідна частота дискретизації відрізняється від цільової (16000 Гц), виконується ресемплінг через librosa.resample з іменованими параметрами orig_sr і target_sr. Завершальним кроком є видалення тиші через WebRTC VAD (Voice Activity Detection) — детектор голосової активності, що ефективніше виявляє межі мовлення порівняно з простим пороговим відсіканням за амплітудою.

Функція wav_to_mel_spectrogram(wav) обчислює мел-спектрограму, що використовується як вхід нейронної мережі encoder. На відміну від спектрограм для synthesizer, тут застосовуються специфічні параметри, що відповідають умовам навчання моделі: 40 мел-смуг (а не 80, як у synthesizer), вікно 25 мс і крок 10 мс. Ця відмінність є важливою деталлю: передача спектрограми з неправильними параметрами у модель encoder призведе до некоректних ембедінгів.

Реалізація модуля synthesizer

Модуль synthesizer реалізований у вигляді класу Synthesizer у файлі synthesizer/inference.py. Клас ініціалізується шляхом передачі шляху до файлу з вагами моделі: synthesizer = Synthesizer(model_fpath). При ініціалізації відбувається завантаження моделі Tacotron з файлу .pt і її переведення на пристрій виконання (GPU при наявності, інакше CPU), про що повідомляє рядок Synthesizer using device: cuda або cpu у виводі.

Основним методом класу є synthesize_spectrograms(texts, embeddings, states=None). Метод приймає список рядків тексту і відповідний список голосових ембедінгів у вигляді NumPy-масивів. Тексти попередньо перетворюються у числові послідовності через text_to_sequence з

synthesizer/utils/text.py, що виконує нормалізацію тексту і його кодування у числові ідентифікатори символів. Ембедінги конвертуються у тензори PyTorch і передаються у модель разом із текстовими послідовностями.

Модель Tacotron генерує мел-спектрограму авторегресивно, кадр за кадром, доки не досягне маркера кінця послідовності (stop token). На кожному кроці декодер отримує попередньо згенерований кадр спектрограми, поточний стан уваги та голосовий ембедінг. Механізм уваги обчислює вагові коефіцієнти для кожного символу вхідного тексту, формуючи контекстний вектор, що визначає, яка частина тексту відповідає поточному моменту у спектрограмі. Результатом є двовимірний масив NumPy розмірності (80, T), де 80 — кількість мел-смуг, T — кількість часових кадрів, що залежить від довжини тексту.

Нормалізація тексту у synthesizer/utils/cleaners.py виконує ряд послідовних перетворень: приведення до нижнього регістру, видалення зайвих пробілів і спеціальних символів, розгортання абревіатур і скорочень. Функція normalize_numbers у synthesizer/utils/numbers.py перетворює числові вирази у їх словесні еквіваленти англійською мовою. Бібліотека unidecode транслітерує нелатинські символи у латинський алфавіт, що дозволяє обробляти текст, який містить символи різних мов, хоча якість синтезу для нелатинських мов при цьому знижується.

Реалізація модуля vocoder

Модуль vocoder реалізований у vocoder/inference.py і надає два функціональних інтерфейси. Функція load_model(weights_fpath, verbose=True) завантажує ваги WaveRNN і ініціалізує модель. При завантаженні виводиться повідомлення Building Wave-RNN, після чого модель переводиться на GPU через .cuda() при наявності відповідного пристрою.

Архітектура WaveRNN реалізована у vocoder/models/fatchord_version.py і складається з двох основних компонентів. Мережа UpsampleNetwork приймає мел-спектрограму і просторово розтягує її до розмірності аудіосигналу, використовуючи серію ResNet-блоків (MelResNet) для перетворення ознак і транспонованих згорток для збільшення часового масштабу. Параметр

upsample_factors визначає коефіцієнти збільшення масштабу на кожному кроці; добуток усіх коефіцієнтів дорівнює hop_length — кількості аудіовибірок на кадр спектрограми. Саме цей добуток обчислюється через np.cumprod(upsample_scales)[-1] у конструкторі класу.

Основна мережа WaveRNN реалізує авторегресивну генерацію: на кожному кроці вона отримує попередню аудіовибірку, відповідний стовпець розтягнутої спектрограми і прихований стан двошарової GRU-мережі. Вихідний шар через softmax розподіл генерує ймовірності для дискретних значень вибірки. Для прискорення генерації застосовується техніка dual softmax head: старші і молодші 8 біт вибірки генеруються окремими головами, що зменшує розмір вихідного шару з 65536 до 512 класів при збереженні 16-бітної якості аудіо.

Функція infer_waveform(mel, normalize=True, batched=True, target=8000, overlap=800) виконує генерацію аудіосигналу з мел-спектрограми. При batched=True спектрограма розбивається на перекриваючі фрагменти, що обробляються паралельно на GPU, а результати з'єднуються з перекриттям для усунення артефактів на стиках. Параметр target визначає розмір батчу в кадрах спектрограми, overlap — розмір зони перекриття. Повернений масив NumPy містить аудіовибірки у форматі float32 з частотою дискретизації 22050 Гц.

Реалізація notebook — повний пайплайн

Файл demo.ipynb реалізує повний пайплайн клонування голосу у вигляді послідовності клітинок Jupyter Notebook. Структура notebook відображає природну послідовність етапів пайплайну і забезпечує максимальну прозорість для розуміння роботи системи.

Перша клітинка виконує перевірку середовища виконання: визначає наявність і характеристики GPU через torch.cuda.is_available() і torch.cuda.get_device_properties(0), виводить версію Python і CUDA. Ця клітинка є діагностичним інструментом, що дозволяє переконатися у коректності встановленого середовища перед запуском ресурсоємних операцій.

Друга клітинка виконує імпорт усіх необхідних модулів: трьох основних

модулів пайплайну (encoder.inference, synthesizer.inference.Synthesizer, vocoder.inference), допоміжної функції ensure_default_models та бібліотек для роботи з даними і візуалізацією (NumPy, librosa, soundfile, matplotlib, IPython.display). Імпорт зібраний в одній клітинці для чіткого розмежування між налаштуванням середовища і виконанням пайплайну.

Третя клітинка реалізує завантаження моделей. Спочатку викликається ensure_default_models, що перевіряє наявність файлів моделей і за необхідності завантажує їх з HuggingFace. Після цього послідовно завантажуються всі три модулі через відповідні функції і класи. Кожен крок супроводжується виводом статусного повідомлення, що дозволяє відстежувати прогрес завантаження.

Четверта клітинка реалізує завантаження і попередню обробку аудіозразка. Шлях до файлу задається змінною AUDIO_PATH, що дозволяє легко замінити вхідний зразок без модифікації іншого коду. Функція librosa.load завантажує аудіо зі збереженням оригінальної частоти дискретизації, після чого encoder.preprocess_wav виконує ресемплінг до 16 кГц і видалення тиші. Клітинка виводить метадані файлу (ім'я, оригінальна і оброблена тривалість, частота дискретизації) і відтворює оригінальний зразок через IPython.display.Audio.

П'ята клітинка виконує обчислення голосового ембедінгу через encoder.embed_utterance і виводить характеристики отриманого вектора: розмірність (256,), тип даних і норму (повинна дорівнювати 1.0). Вектор ембедінгу візуалізується у вигляді стовпчикової діаграми через matplotlib, де кожен стовпець відповідає одному з 256 вимірів. Ця візуалізація є інструментом перевірки коректності роботи encoder: нормалізований ембедінг повинен мати значення переважно в діапазоні від -0.2 до 0.2 з рівномірним розподілом позитивних і негативних значень.

Шоста клітинка реалізує синтез мел-спектрограми. Текст для синтезу задається змінною TEXT, що дозволяє легко замінювати вхідний текст. Метод synthesizer.synthesize_spectrograms приймає список текстів і список ембедінгів у вигляді батчу і повертає список спектрограм; для одиночного запиту

результатом є список з одного елементу. Отримана спектрограма візуалізується через librosa.display.specshow з параметрами x_axis='time' і y_axis='mel' для коректного масштабування осей.

Сьома клітинка реалізує синтез аудіосигналу через vocoder.infer_waveform. Після генерації виконується пост-обробка: додавання невеликого буфера тиші в кінці через np.pad для запобігання обрізання звуку при відтворенні, повторна обробка через encoder.preprocess_wav для видалення зайвої тиші і нормалізація амплітуди до рівня 0.95 від максимуму. Відтворення результату виконується безпосередньо у notebook через IPython.display.Audio.

Восьма клітинка зберігає результат у файл через soundfile.write і генерує порівняльний графік, що відображає форму хвилі оригінального зразка і синтезованого результату поруч. Цей графік є зручним інструментом для візуальної оцінки відповідності темпу і структури мовлення між зразком і результатом.

Обробка помилок сумісності

При розгортанні проекту на сучасних версіях Python і залежних бібліотек виникає ряд проблем сумісності, пов'язаних із тим, що репозиторій орієнтований на версії бібліотек, актуальні на момент його створення. У процесі реалізації виявлено і усунено три такі проблеми.

Перша проблема стосується функції np.cumproduct у файлі vocoder/models/fatchord_version.py рядок 64. Ця функція була видалена у NumPy 2.0 на користь np.cumprod. При використанні NumPy 2.x виникає AttributeError: module 'numpy' has no attribute 'cumproduct'. Виправлення полягає у заміні виклику np.cumproduct(upsample_scales)[-1] на np.cumprod(upsample_scales)[-1] — функції є повністю ідентичними, різниця лише у назві.

Друга проблема стосується функції librosa.feature.melspectrogram у файлі encoder/audio.py рядок 58. У нових версіях librosa (0.9+) ця функція більше не приймає перші два аргументи позиційно — вони повинні передаватися як іменовані параметри y= і sr=. При використанні librosa 0.11.x виникає TypeError: melspectrogram() takes 0 positional arguments. Виправлення полягає у заміні

librosa.feature.melspectrogram(wav, sampling_rate, ...) на librosa.feature.melspectrogram(y=wav, sr=sampling_rate, ...).

Третя проблема стосується функції librosa.resample у файлі encoder/audio.py рядок 42. Аналогічно до попередньої проблеми, у нових версіях librosa параметри orig_sr і target_sr повинні передаватися як іменовані. При позиційній передачі виникає TypeError: resample() takes 1 positional argument but 3 were given. Виправлення — заміна librosa.resample(wav, source_sr, sampling_rate) на librosa.resample(wav, orig_sr=source_sr, target_sr=sampling_rate).

3.3. Елементи реалізації

Після визначення архітектурного каркасу та вибору технологій настає етап безпосередньої реалізації ключових функціональних елементів пайплайну. У рамках даної роботи акцент робиться на тих компонентах, що формують основний робочий ланцюжок: від завантаження аудіозразка до отримання синтезованого результату. Кожен елемент реалізований як окремий, ізольований етап у Jupyter Notebook із чітко визначеними вхідними і вихідними даними, що відповідає модульній природі архітектури SV2TTS. Нижче детально розглядається реалізація кожного ключового елементу пайплайну.

Елемент 1. Перевірка середовища виконання

Першим елементом реалізації є перевірка середовища виконання, що виконується до будь-яких обчислювальних операцій. Цей елемент є важливим практичним інструментом, оскільки продуктивність і навіть коректність пайплайну суттєво залежать від конфігурації апаратного і програмного забезпечення.

Перевірка визначає наявність GPU через torch.cuda.is_available() і при позитивному результаті виводить детальну інформацію про пристрій: назву GPU, загальний обсяг відеопам'яті в гігабайтах через torch.cuda.get_device_properties(0).total_memory, версію CUDA через torch.version.cuda. На основі результату перевірки встановлюється змінна

DEVICE, що приймає значення "cuda" або "cpu" і використовується далі для явного управління пристроєм виконання.

Значення змінної DEVICE є критично важливим для налаштування очікувань щодо часу виконання. На GPU NVIDIA RTX 3060 повний пайплайн для тексту довжиною 20–30 слів займає 10–30 секунд. На CPU аналогічна операція може займати від 5 до 15 хвилин залежно від кількості ядер процесора і тактової частоти — що робить інтерактивне використання на CPU практично непридатним для регулярної роботи.

```
import torch
import sys

print(f"Python версія: {sys.version}")

if torch.cuda.is_available():
    gpu_name = torch.cuda.get_device_name(0)
    gpu_memory = torch.cuda.get_device_properties(0).total_memory / 1e9
    print(f"GPU знайдено: {gpu_name}")
    print(f"Відеопам'ять: {gpu_memory:.1f} ГБ")
    print(f"CUDA версія: {torch.version.cuda}")
    DEVICE = "cuda"
else:
    print("GPU не знайдено — використовується CPU")
    DEVICE = "cpu"

print(f"Активний пристрій: {DEVICE}")
```

Елемент 2. Імпорт модулів пайплайну

Другим елементом є імпорт усіх необхідних модулів. Структура імпортів відображає трирівневу природу пайплайну: три рядки імпортів відповідають трьом модулям SV2TTS, решта — допоміжним бібліотекам для обробки даних і візуалізації.

```
python
from pathlib import Path
import numpy as np
import librosa
import soundfile as sf
import IPython.display as ipd
import matplotlib.pyplot as plt

from encoder import inference as encoder
from synthesizer.inference import Synthesizer
from vocoder import inference as vocoder
from utils.default_models import ensure_default_models
```

Модуль encoder імпортується цілком як простір імен, оскільки його публічний API реалізований у вигляді функцій модуля (не класу): encoder.load_model, encoder.embed_utterance, encoder.preprocess_wav. Це відрізняється від підходу synthesizer, де API реалізований через клас Synthesizer, що відображає різницю у внутрішній організації модулів репозиторію.

Важливим є те, що імпорт модулів не ініціює завантаження ML-моделей у пам'ять — це відбувається лише при явному виклику відповідних функцій завантаження. Такий підхід до відкладеної ініціалізації (lazy initialization) є принциповим для пайплайну з важкими моделями: він дозволяє виконати імпорт і перевірити наявність усіх залежностей без необхідності мати достатньо пам'яті для одночасного утримання всіх трьох моделей.

Елемент 3. Завантаження попередньо натренованих моделей

Завантаження моделей є одним із найважливіших елементів пайплайну і потребує детального розуміння механізму роботи. Функція ensure_default_models(Path("saved_models")) перевіряє наявність файлів encoder.pt, synthesizer.pt і vocoder.pt у директорії saved_models/default/. Якщо будь-який з файлів відсутній, функція автоматично завантажує його з репозиторію CorentinJ/SV2TTS на HuggingFace через API бібліотеки

huggingface_hub [9].

```
MODELS_DIR = Path("saved_models/default")
ENC_MODEL  = MODELS_DIR / "encoder.pt"
SYN_MODEL  = MODELS_DIR / "synthesizer.pt"
VOC_MODEL  = MODELS_DIR / "vocoder.pt"

print("Перевірка наявності моделей...")
ensure_default_models(Path("saved_models"))

print("Завантаження encoder...")
encoder.load_model(ENC_MODEL)

print("Завантаження synthesizer...")
synthesizer = Synthesizer(SYN_MODEL)

print("Завантаження vocoder...")
vocoder.load_model(VOC_MODEL)

print("Усі моделі завантажено успішно")
```

При успішному завантаженні encoder виводить повідомлення Loaded encoder "encoder.pt" trained to step 1564501, що підтверджує завантаження конкретної контрольної точки навчання. Synthesizer виводить Synthesizer using device: cuda або cpu, що підтверджує коректне визначення пристрою. Vocoder виводить Building Wave-RNN перед ініціалізацією архітектури моделі.

Загальний обсяг пам'яті, що займають три завантажені моделі, складає приблизно 1.2–1.5 ГБ оперативної пам'яті при виконанні на CPU або відеопам'яті при виконанні на GPU. Для GPU з 12 ГБ відеопам'яті, як у даній роботі, це є цілком прийнятним і залишає достатній резерв для батчевих обчислень під час генерації.

Елемент 4. Завантаження та попередня обробка аудіозразка

Четвертий елемент реалізує завантаження вхідного аудіофайлу і його підготовку до передачі у модуль encoder. Вхідний файл задається змінною AUDIO_PATH, що є єдиним параметром, який користувачу необхідно змінювати для роботи з різними голосами.

```
AUDIO_PATH = Path("samples/1320_00000.mp3")

if not AUDIO_PATH.exists():
    sample_files = (list(Path("samples").glob("*.mp3")) +
                    list(Path("samples").glob("*.wav")))
    if sample_files:
        AUDIO_PATH = sample_files[0]
        print(f"Використовуємо вбудований зразок: {AUDIO_PATH.name}")
    else:
        raise FileNotFoundError("Аудіо файл не знайдено.")

original_wav, original_sr = librosa.load(str(AUDIO_PATH), sr=None)
processed_wav = encoder.preprocess_wav(original_wav, original_sr)

duration_orig = len(original_wav) / original_sr
duration_proc = len(processed_wav) / encoder.sampling_rate

print(f"Файл: {AUDIO_PATH.name}")
print(f"Оригінальна частота дискретизації: {original_sr} Гц")
print(f"Тривалість оригіналу: {duration_orig:.2f} с")
print(f"Після обробки (16 кГц): {duration_proc:.2f} с")

ipd.display(ipd.Audio(original_wav, rate=original_sr))
```

Резервна логіка пошуку файлу реалізована через перевірку існування шляху і автоматичний пошук будь-якого аудіофайлу у директорії samples/ при відсутності вказаного. Це забезпечує стійкість до ситуації, коли користувач ще

не додав власний аудіофайл, і дозволяє одразу запустити пайплайн із вбудованими тестовими зразками репозиторію.

Виклик librosa.load(str(AUDIO_PATH), sr=None) з параметром sr=None є принциповим: він зберігає оригінальну частоту дискретизації файлу замість автоматичного ресемплінгу до стандартних 22050 Гц, що librosa виконує за замовчуванням. Збереження оригінальної частоти необхідне для коректної передачі цього параметра у encoder.preprocess_wav, яка самостійно виконає ресемплінг до потрібних encoder'у 16000 Гц.

Функція encoder.preprocess_wav(original_wav, original_sr) виконує два ключових кроки: ресемплінг до 16000 Гц через librosa.resample(wav, orig_sr=source_sr, target_sr=sampling_rate) і видалення тиші на початку і в кінці через WebRTC VAD. Результатом є одновимірний масив NumPy типу float32 з частотою дискретизації 16000 Гц, готовий для передачі у нейронну мережу encoder. Відтворення оригінального зразка безпосередньо у notebook через IPython.display.Audio є важливим елементом інтерактивності: воно дозволяє переконатися у коректності завантаженого файлу і оцінити якість вхідного аудіо перед запуском ресурсоємного пайплайну.

Разом із відтворенням виконується візуалізація вхідного сигналу у вигляді двох графіків: форми хвилі (waveform) через matplotlib.pyplot і мел-спектрограми через librosa.display.specshow. Форма хвилі відображає амплітуду сигналу у часі і дозволяє оцінити наявність і характер тиші, рівень гучності та загальну структуру мовлення. Мел-спектрограма відображає розподіл енергії сигналу за частотою і часом у логарифмічній шкалі, що є набагато інформативнішим представленням для аналізу мовлення.

Елемент 5. Обчислення голосового ембедінгу

П'ятий елемент реалізує центральну операцію модуля encoder — обчислення вектора голосового ембедінгу, що кодує індивідуальні характеристики диктора.

python

```
print("Кодування голосового зразка...")
speaker_embedding = encoder.embed_utterance(processed_wav)
```

```
print(f"Розмірність вектора: {speaker_embedding.shape}")
print(f"Тип даних: {speaker_embedding.dtype}")
print(f"Норма вектора: {np.linalg.norm(speaker_embedding):.4f}")

plt.figure(figsize=(12, 2))
plt.bar(range(len(speaker_embedding)), speaker_embedding,
        color="steelblue", alpha=0.7, width=1.0)
plt.title("Вектор ембедінгу диктора (256 вимірів)", fontsize=13)
plt.xlabel("Індекс виміру")
plt.ylabel("Значення")
plt.grid(True, alpha=0.3, axis="y")
plt.tight_layout()
plt.savefig("output_speaker_embedding.png", dpi=150, bbox_inches="tight")
plt.show()
```

Виведення норми вектора є практичним діагностичним кроком: результат encoder.embed_utterance завжди повинен мати норму рівно 1.0, оскільки функція виконує L2-нормалізацію перед поверненням результату. Відхилення від 1.0 свідчить про проблему у процесі обчислення або некоректну версію завантаженої моделі.

Візуалізація 256-вимірного вектора у вигляді стовпчикової діаграми дозволяє отримати інтуїтивне уявлення про структуру ембедінгу. Кожен стовпець відповідає одному виміру простору ембедінгів. Характерним є відносно рівномірний розподіл значень у діапазоні приблизно від -0.15 до 0.15 з відсутністю домінуючих вимірів — це є ознакою якісно навченого encoder, де інформація рівномірно розподілена по всіх вимірах простору.

Важливо розуміти семантику ембедінгу: два аудіозразки одного і того ж диктора, записані в різних умовах і з різним змістом, дадуть ембедінги з високою косинусною подібністю (зазвичай вище 0.85). Аудіозразки різних дикторів дадуть ембедінги з низькою подібністю (зазвичай нижче 0.5). Саме ця

властивість encoder є основою механізму клонування: synthesizer, кондиціонований на ембедінг конкретного диктора, генерує спектрограму з характеристиками саме цього голосу.

Елемент 6. Синтез мел-спектрограми

Шостий елемент реалізує найважчу з обчислювальної точки зору операцію пайплайну — синтез мел-спектрограми з тексту з урахуванням голосового ембедінгу.

python

```
TEXT = "Hello, this is a demonstration of voice cloning using deep learning."

print(f"Текст для синтезу: '{TEXT}'")
print(f"Кількість слів: {len(TEXT.split())}")

texts      = [TEXT]
embeddings = [speaker_embedding]

spectrograms    = synthesizer.synthesize_spectrograms(texts, embeddings)
mel_spectrogram = spectrograms[0]

print(f"Розмір спектрограми: {mel_spectrogram.shape}")
print(f"Частота дискретизації synthesizer: {synthesizer.sample_rate} Гц")

plt.figure(figsize=(12, 4))
plt.imshow(mel_spectrogram, aspect="auto", origin="lower", cmap="magma")
plt.colorbar(label="Амплітуда")
plt.title(f"Згенерована мел-спектрограма", fontsize=12)
plt.xlabel("Часові кадри")
plt.ylabel("Мел-смуги")
plt.tight_layout()
plt.savefig("output_generated_spectrogram.png", dpi=150, bbox_inches="tight")
```

plt.show()

Метод synthesize_spectrograms приймає дані у вигляді списків (texts і embeddings), що відображає батчевий характер API: в одному виклику можна синтезувати кілька спектрограм паралельно. У даній реалізації використовується батч розміром 1, проте API залишається незмінним. Результатом є список спектрограм; перша (і єдина у батчі) спектрограма витягується через spectrograms[0].

Розмір отриманої спектрограми (80, T) несе практичну інформацію: 80 фіксованих мел-смуг визначаються архітектурою synthesizer, тоді як кількість часових кадрів T залежить від довжини тексту і середнього темпу мовлення. Для речення із 10–12 слів типове значення T становить 400–600 кадрів. При частоті кадрів synthesizer 80 Гц (hop_length 256 при частоті дискретизації 22050 Гц) це відповідає тривалості аудіо 5–7 секунд — що узгоджується з природним темпом мовлення.

Відображення спектрограми через plt.imshow з параметрами aspect="auto" і origin="lower" є стандартним способом візуалізації для TTS-досліджень: вісь X відповідає часу (кадри зліва направо), вісь Y — частоті (мел-смуги знизу вгору, від низьких до високих частот). Колірна карта magma відображає амплітуду від темно-фіолетового (низька) до жовтого (висока). У якісно згенерованій спектрограмі добре видно вертикальні смуги, що відповідають окремим звукам, і горизонтальні формантні структури, характерні для людського мовлення.

Елемент 7. Синтез аудіосигналу та пост-обробка

Сьомий елемент реалізує перетворення мел-спектрограми у відтворюваний аудіосигнал через модуль vocoder і виконує пост-обробку результату.

python

```
print("Запуск WaveRNN vocoder...")
generated_wav = vocoder.infer_waveform(mel_spectrogram)

generated_wav = np.pad(generated_wav,
                       (0, synthesizer.sample_rate),
```

```
                mode="constant")
generated_wav = encoder.preprocess_wav(generated_wav)

max_amplitude = np.max(np.abs(generated_wav))
if max_amplitude > 0:
    generated_wav = generated_wav / max_amplitude * 0.95

duration_generated = len(generated_wav) / synthesizer.sample_rate
print(f"Тривалість: {duration_generated:.2f} с")
print(f"Частота дискретизації: {synthesizer.sample_rate} Гц")

print("Згенероване аудіо:")
ipd.display(ipd.Audio(generated_wav, rate=synthesizer.sample_rate))
```

Функція vocoder.infer_waveform(mel_spectrogram) виконує авторегресивну генерацію аудіовибірок з мел-спектрограми. Час виконання цієї операції на GPU NVIDIA RTX 3060 становить приблизно 5–15 секунд для типового речення і є найбільш тривалою частиною пайплайну після синтезу спектрограми.

Пост-обробка синтезованого аудіо включає три послідовних кроки. Перший — додавання буфера тиші в кінці через np.pad — є компенсацією характерного артефакту WaveRNN, де останні вибірки можуть бути обрізані через специфіку авторегресивної генерації. Розмір буфера дорівнює одній секунді (значення synthesizer.sample_rate вибірок). Другий крок — повторна обробка через encoder.preprocess_wav — видаляє зайву тишу на початку і в кінці, що могла утворитися після додавання буфера. Третій крок — нормалізація амплітуди — приводить максимальне абсолютне значення сигналу до рівня 0.95, що гарантує відсутність кліпінгу і стандартний рівень гучності результату.

Відтворення результату безпосередньо у notebook через IPython.display.Audio є ключовим елементом інтерактивності реалізації: воно дозволяє оцінити якість клонування відразу після генерації, без необхідності зберігати файл і відкривати зовнішній програвач.

Елемент 8. Збереження результату та порівняльна візуалізація

Восьмий елемент реалізує збереження синтезованого аудіо у файл і генерацію порівняльного графіка, що відображає оригінальний і синтезований сигнали поруч.

python

```
OUTPUT_PATH = Path("output_cloned_voice.wav")
sf.write(str(OUTPUT_PATH),
        generated_wav.astype(np.float32),
        synthesizer.sample_rate)
print(f"Результат збережено: {OUTPUT_PATH}")

fig, axes = plt.subplots(2, 1, figsize=(14, 6))

t_orig = np.linspace(0, duration_orig, len(original_wav))
axes[0].plot(t_orig, original_wav,
            linewidth=0.4, color="steelblue", alpha=0.8)
axes[0].set_title(f"Оригінальний голосовий зразок ({AUDIO_PATH.name})",
                fontsize=12)
axes[0].set_xlabel("Час (с)")
axes[0].set_ylabel("Амплітуда")
axes[0].grid(True, alpha=0.3)

t_gen = np.linspace(0, duration_generated, len(generated_wav))
axes[1].plot(t_gen, generated_wav,
            linewidth=0.4, color="coral", alpha=0.8)
axes[1].set_title(f"Синтезований голос: '{TEXT[:60]}'",
                fontsize=12)
axes[1].set_xlabel("Час (с)")
axes[1].set_ylabel("Амплітуда")
axes[1].grid(True, alpha=0.3)
```

```
plt.tight_layout()
plt.savefig("output_comparison.png", dpi=150, bbox_inches="tight")
plt.show()
```

Збереження виконується через soundfile.write з явним приведенням типу до np.float32, що є стандартним форматом для WAV-файлів з плаваючою крапкою. Функція автоматично записує заголовок WAV-файлу з коректними метаданими (частота дискретизації, кількість каналів, формат вибірок).

Порівняльний графік є практичним інструментом для первинної оцінки результату клонування. На ньому видно відповідність між структурними характеристиками оригінального і синтезованого сигналів: загальна тривалість, наявність пауз між словами, рівень гучності. Якісне клонування демонструє подібну ритмічну структуру при збереженні характеристик голосу оригінального диктора.

Елемент 9. Узагальнена функція клонування

Дев'ятим елементом є реалізація узагальненої функції clone_voice, що інкапсулює повний пайплайн і дозволяє зручно виконувати клонування для різних комбінацій аудіозразків і текстів без повторення коду.

python

```
def clone_voice(audio_path: Path,
                text: str,
                output_path: Path) -> float:
    import time
    start_time = time.time()

    # Етап 1: Попередня обробка аудіо
    wav, sr       = librosa.load(str(audio_path), sr=None)
    wav_processed = encoder.preprocess_wav(wav, sr)

    # Етап 2: Кодування голосу → ембедінг (256 вимірів)
```

```
    embedding = encoder.embed_utterance(wav_processed)

    # Етап 3: Синтез мел-спектрограми (Tacotron)
    specs = synthesizer.synthesize_spectrograms([text], [embedding])

    # Етап 4: Синтез аудіо (WaveRNN)
    result_wav = vocoder.infer_waveform(specs[0])
    result_wav = np.pad(result_wav,
                        (0, synthesizer.sample_rate),
                        mode="constant")
    result_wav = encoder.preprocess_wav(result_wav)

    max_amp = np.max(np.abs(result_wav))
    if max_amp > 0:
        result_wav = result_wav / max_amp * 0.95

    sf.write(str(output_path),
             result_wav.astype(np.float32),
             synthesizer.sample_rate)

    elapsed  = time.time() - start_time
    duration = len(result_wav) / synthesizer.sample_rate
    rtf      = elapsed / duration

    print(f"Зразок: {audio_path.name}")
    print(f"Час генерації: {elapsed:.1f} с | "
          f"Тривалість аудіо: {duration:.1f} с | RTF: {rtf:.2f}")

    return duration
```

Функція clone_voice є практичним підсумком усієї реалізації: вона об'єднує всі

чотири етапи пайплайну в єдиному виклику і повертає тривалість згенерованого аудіо. Додатково обчислюється метрика RTF (Real-Time Factor) — відношення часу генерації до тривалості результату. RTF менше 1.0 означає генерацію швидше реального часу, RTF більше 1.0 — повільніше. На GPU NVIDIA RTX 3060 типове значення RTF для даного пайплайну становить 0.3–0.8, що є прийнятним результатом для інтерактивного використання.

Наявність цієї функції дозволяє виконувати пакетне тестування з різними аудіозразками і текстами в циклі, не дублюючи код пайплайну. Це особливо зручно при оцінці якості клонування для різних умов запису, різних дикторів або різних типів тексту — у кожному випадку достатньо одного виклику clone_voice з відповідними параметрами.

3.4. Можливі розширення

Розроблена альфа–версія застосунку демонструє основний пайплайн клонування голосу і підтверджує прийнятність обраної архітектури. Водночас, як і будь–який перший функціональний прототип, вона залишає значний простір для вдосконалення. Нижче розглядаються конкретні напрями розширення, згруповані за характером змін, що вони вносять у систему.

Підвищення якості та реалістичності синтезу:

Найочевиднішим напрямом розвитку є покращення якості синтезованого голосу. У поточній реалізації використовується Tortoise TTS з пресетом fast, що свідомо жертвує деякою якістю заради прийнятного часу генерації. Перехід на пресет high_quality або standard дає помітно кращий результат ціною більшого часу обробки.

Більш суттєвим покращенням є інтеграція новіших моделей синтезу. Станом на 2025–2026 роки з'явилися відкриті моделі, що поєднують вищу якість із значно швидшою генерацією порівняно з Tortoise TTS. Зокрема, заслуговують уваги Kokoro TTS – компактна і швидка модель із природним звучанням, та Chatterbox від Resemble AI – відкрита модель, орієнтована на

емоційний контроль і низьку латентність. Завдяки архітектурному рішенню про використання абстрактного інтерфейсу BaseSynthesizer, впровадження нової моделі зводиться до написання нового класу–адаптера без зміни іншого коду.

Окремим напрямом покращення якості є тонке доналаштування (fine–tuning) моделі на датасеті конкретного диктора. Якщо доступний запис тривалістю від 15 до 30 хвилин, кілька годин доналаштування на GPU дають якість значно вищу, ніж кондиціювання з короткого зразка. Цей режим може бути реалізований як окрема функція «Глибоке навчання профілю» для досвідчених користувачів із відповідним апаратним забезпеченням.

Розширення можливостей управління голосом:

Поточна версія надає обмежені засоби впливу на характеристики синтезованого голосу. Значним покращенням користувацького досвіду стало б впровадження детального емоційного контролю: можливості обирати або змішувати емоційні стани (нейтральний, радісний, стурбований, авторитетний), що безпосередньо впливають на просодію і темп мовлення.

Пов'язаним розширенням є контроль ритміки та наголосів через підтримку спрощеної розмітки SSML (Speech Synthesis Markup Language). SSML є XML–стандартом для управління синтезом мовлення і дозволяє вставляти паузи заданої тривалості (<break time="500ms"/>), встановлювати темп і гучність для окремих фрагментів (<prosody rate="slow">) та задавати фонетичне написання для нестандартних слів (<phoneme>). Реалізація базового парсера SSML у TextNormalizer надала б користувачам тонкий контроль над результатом.

Ще одним цінним доповненням є конвертація голосу (Voice Conversion) – режим, де замість синтезу з тексту виконується перетворення одного аудіозапису мовлення у голос цільового диктора. Це відкриває застосування у дубляжі, де вже є запис із правильними інтонаціями, але потрібен інший голос. Реалізація можлива через інтеграцію бібліотеки RVC (Retrieval–based Voice Conversion) як альтернативного режиму роботи.

Оптимізація продуктивності:

Поточна реалізація не є оптимальною з точки зору ефективності використання ресурсів, і ряд вдосконалень може суттєво скоротити час генерації.

Квантизація моделі (model quantization) – техніка зменшення точності числових представлень ваг нейронної мережі з 32–бітних або 16–бітних значень до 8–бітних цілих чисел. Це зменшує розмір моделі в пам'яті та прискорює обчислення при незначній деградації якості. PyTorch підтримує динамічну квантизацію через torch.quantization, що дозволяє застосувати її без перенавчання моделі.

Батчева обробка (batch processing) вже частково реалізована в поточному пайплайні: метод synthesizer.synthesize_spectrograms приймає список текстів і список ембедінгів одночасно, що дозволяє генерувати кілька різних текстів з одним голосовим профілем за один прохід через модель. Розширення цього механізму дозволить повністю автоматизувати генерацію великих обсягів аудіоконтенту — наприклад, озвучення цілої книги або набору навчальних матеріалів — з мінімальною участю користувача.

Кешування кондиціювальних латентів є оптимізацією, що вже частково реалізована в поточній версії. Повноцінна реалізація передбачає збереження обчислених кондиціювальних латентів разом із профілем диктора, що усуває їх повторне обчислення при кожній новій генерації з тим самим голосом.

Підтримка нових мов:

Поточна версія орієнтована на англійську мову, оскільки synthesizer на базі Tacotron навчений на англомовних датасетах і оперує фіксованим набором ASCII-символів, визначеним у synthesizer/utils/symbols.py.Coqui XTTS v2 є найбільш зрілим відкритим рішенням для мультимовного клонування, що підтримує понад 17 мов і демонструє прийнятну якість навіть для мов із відносно невеликими навчальними датасетами. Його інтеграція як альтернативного синтезатора у рамках поточної архітектури є логічним наступним кроком, особливо з урахуванням потреби у якісному

україномовному голосовому синтезі.

Паралельно варто розширити клас TextNormalizer правилами нормалізації для нових мов. Для кожної мови правила відмінювання чисел, розгортання абревіатур і обробки спеціальних символів є унікальними і вимагають окремої реалізації або інтеграції спеціалізованої бібліотеки нормалізації тексту.

Інтеграція з хмарними сервісами:

Для користувачів без потужного локального апаратного забезпечення перспективним розширенням є хмарний режим обробки. У цьому режимі аудіозразок і текст надсилаються на захищений хмарний сервер (власний або на базі публічного хмарного провайдера), де синтез виконується на потужному серверному GPU, а результат повертається клієнту.

Архітектурно це реалізується як ще одна стратегія BaseSynthesizer – CloudSynthesizer, що замість локальної ML–моделі виконує HTTP–запит до REST API сервера. Для клієнтської частини зміни мінімальні: лише вибір стратегії у конфігурації. Серверна частина може бути реалізована як окремий мікросервіс на FastAPI або Flask, розгорнутий у хмарі.

Механізми безпеки та етичного захисту:

З огляду на потенціал зловживання технологією, важливим напрямом розширення є посилення механізмів відповідального використання.

Вбудовані аудіо водяні знаки – приховані цифрові маркери у синтезованому аудіо, що ідентифікують його як синтетичне. Бібліотека AudioSeal від Meta AI є відкритим рішенням для вбудовування і виявлення таких маркерів. Її інтеграція у AudioPostProcessor як опціональний крок дозволить маркувати весь синтезований контент без помітного впливу на якість звучання. Детектор синтетичного мовлення – компонент, що аналізує завантажений аудіозразок і оцінює ймовірність того, що він є синтетичним, а не реальним записом. Це додатковий захист від ланцюгового клонування, де синтетичний голос використовується як зразок для нового клонування. Бібліотека Resemblyzer або спеціалізовані детектори deepfake можуть бути використані для цієї мети.

Покращення користувацького досвіду:

Поряд із технічними розширеннями, існує ряд вдосконалень, що безпосередньо покращують зручність роботи із застосунком.

Пакетна генерація дозволить обробляти список текстів в автоматичному режимі, що є цінним для контент–мейкерів, яким потрібно озвучити великий обсяг матеріалу. Користувач завантажує текстовий файл із переліком фраз або абзаців, обирає голосовий профіль і запускає пакетну обробку – застосунок послідовно генерує аудіо для кожного елементу і зберігає результати у вказану директорію.

Порівняльний режим дозволить відображати кілька варіантів синтезу поряд для прослуховування та порівняння. При генерації кількох варіантів з різними параметрами або пресетами користувач зможе прослухати їх у одному інтерфейсі і зберегти кращий.

Інтеграція із зовнішніми редакторами через протокол командного рядка або drag–and–drop дозволить використовувати застосунок як плагін у звичному робочому середовищі аудіоредактора (наприклад, Audacity або DaVinci Resolve), що розширить цільову аудиторію до професіоналів медіавиробництва.

РОЗДІЛ 4. ТЕСТУВАННЯ ДЕСКТОПНОГО ЗАСТОСУНКУ КЛОНУВАННЯ ГОЛОСУ

Тестування програмного забезпечення є невід'ємною частиною процесу розробки, що забезпечує відповідність готового продукту визначеним вимогам і очікуванням користувачів. У контексті застосунку для клонування голосу тестування набуває особливого значення через складну природу системи: вона поєднує компоненти обробки сигналів, ML–моделі та графічний інтерфейс, кожен із яких може бути джерелом специфічних проблем. Помилка у будь–якій ланці пайплайну призводить до деградації якості кінцевого результату або повного збою системи, тому комплексний підхід до тестування є принциповою вимогою.

Загальна стратегія тестування альфа–версії базується на поєднанні кількох рівнів перевірки, що відповідають класичній тестовій піраміді (testing

pyramid): широка основа автоматизованих модульних тестів, менший шар інтеграційних тестів і верхівка у вигляді ручного тестування кінцевої функціональності. Такий підхід забезпечує баланс між повнотою покриття і витратами часу на підтримку тестового середовища.

4.1. Методика і проведення тестування

Модульне тестування (Unit Testing):

Модульні тести перевіряють ізольовані компоненти системи – окремі класи і функції – незалежно від решти коду. Для реалізації модульних тестів обрано фреймворк pytest, що є стандартом у Python–спільноті завдяки лаконічному синтаксису, зручній системі фікстур (fixtures) і багатій екосистемі плагінів.

Ключовим принципом модульного тестування є ізоляція: тест повинен перевіряти лише конкретний компонент, замінюючи всі його зовнішні залежності заглушками (mocks і stubs). Наприклад, тест для AudioPreprocessor не повинен реально звертатися до файлової системи або завантажувати ML–модель – натомість він отримує синтетичні вхідні дані і перевіряє правильність перетворень. Для створення заглушок використовується стандартний модуль unittest.mock.

Модульні тести охоплюють такі компоненти пайплайну: функція preprocess_wav модуля encoder/audio.py (коректність ресемплінгу до 16000 Гц, нормалізації амплітуди, обрізання тиші, обробка крайніх випадків — файл коротший за мінімальний поріг, вже коректна частота дискретизації); функція embed_utterance модуля encoder/inference.py (розмірність вихідного вектора 256, нормалізація до одиничної норми, відтворюваність результату для одного й того ж входу); клас Synthesizer модуля synthesizer/inference.py (форма вихідної мел-спектрограми, коректна кількість мел-смуг 80, відповідність вхідного тексту і довжини спектрограми); функція infer_waveform модуля vocoder/inference.py (частота дискретизації вихідного аудіо 22050 Гц,

відсутність кліпінгу в нормалізованому виході); функція ensure_default_models (коректна перевірка наявності файлів, відсутність повторного завантаження при наявних файлах).

Для кожного компонента визначено три типи тест–кейсів: happy path – перевірка коректної роботи з валідними вхідними даними; edge cases – граничні значення (мінімальна тривалість аудіо, порожній текст, максимальна довжина тексту); error cases – очікувана реакція на некоректні вхідні дані (невалідний формат файлу, відсутній файл, від'ємні значення параметрів).

Інтеграційне тестування (Integration Testing):

Інтеграційні тести перевіряють взаємодію між кількома компонентами у рамках одного пайплайну. На відміну від модульних тестів, тут не використовуються заглушки для внутрішніх залежностей – компоненти взаємодіють між собою реально, і тест перевіряє, чи результат цієї взаємодії є коректним.

Основний інтеграційний тест перевіряє повний пайплайн SV2TTS: від завантаження реального аудіофайлу і обробки через preprocess_wav, отримання ембедінгу через embed_utterance, генерації мел-спектрограми через synthesize_spectrograms до отримання аудіосигналу через infer_waveform і його збереження через soundfile.write. Оскільки повний прохід через усі три нейромережеві моделі на CPU займає значний час, інтеграційні тести за замовчуванням запускаються з маркером pytest.mark.slow і виключаються з основного циклу CI; вони виконуються вручну на конфігурації з GPU перед кожним значущим коммітом.Успішне проходження цього тесту підтверджує, що всі компоненти коректно інтегровані і передають дані у сумісних форматах.

Інші інтеграційні тести охоплюють взаємодію ProfileManager із AudioPreprocessor (збереження і відновлення профілю з реальними аудіоданими), а також роботу EventBus у контексті реального контролера (публікація і доставка подій у правильній послідовності).

Ручне тестування (Manual Testing)

Ручне тестування орієнтоване на перевірку тих аспектів системи, які складно

або недоцільно автоматизувати: якість синтезованого голосу, зручність інтерфейсу, коректність відображення прогресу, поведінка системи під навантаженням. Для структурованого проведення ручного тестування складено чек–листи, що покривають основні сценарії використання, описані у розділі 2.2.

Тестування проводиться на двох конфігураціях апаратного і програмного забезпечення для охоплення різних умов використання.

Конфігурація А (з GPU): процесор Intel Core i7–12700, оперативна пам'ять 16 ГБ DDR4, відеокарта NVIDIA RTX 3060 12 ГБ (CUDA 11.8), операційна система Windows 11 64–bit, Python 3.12, PyTorch 2.x з CUDA-підтримкою, встановлено через uv sync --extra cuda. Ця конфігурація відповідає рекомендованому апаратному забезпеченню і використовується для перевірки продуктивності повного пайплайну SV2TTS — усі три модулі (encoder, synthesizer, vocoder) виконуються на GPU.

Конфігурація Б (без GPU): процесор AMD Ryzen 5 5600U, оперативна пам'ять 8 ГБ DDR4, інтегрована графіка, операційна система Ubuntu 22.04 LTS, Python 3.10, PyTorch 2.1 CPU–only. Ця конфігурація моделює мінімальне апаратне забезпечення і дозволяє перевірити поведінку системи у режимі роботи виключно на CPU, включаючи автоматичне перемикання на gTTS [18].

Для забезпечення відтворюваності і повноти тестування підготовлено стандартизований набір тестових аудіофайлів, що охоплює різноманітні умови запису і характеристики голосу.

Набір 1 – Еталонні записи: п'ять аудіофайлів WAV (частота 44100 Гц, стерео), записаних у студійних умовах без фонового шуму, тривалістю 30, 60 і 120 секунд. Це найсприятливіші умови для клонування і слугують базовим критерієм якості.

Набір 2 – Записи у побутових умовах: п'ять аудіофайлів з помірним фоновим

шумом (офісний шум, вентилятор), записаних на мобільний телефон з частотою 16000 Гц. Моделює найтиповіші умови, в яких реальні користувачі записуватимуть зразки.

Набір 3 – Граничні випадки: файли для перевірки граничних умов – тривалість рівно 10 секунд (мінімум), невалідний формат (txt–файл з розширенням wav), пошкоджений WAV–файл, стереозапис із значно різними каналами, запис із кліпінгом.

Набір 4 – Тестові тексти: набір текстів різної довжини і складності для тестування синтезу – короткі (10–20 слів), середні (50–100 слів) і довгі (200+ слів); тексти із числами, датами і абревіатурами; тексти різної фонетичної складності англійською мовою (звичайні речення, скоромовки, технічна лексика); тексти з числами і датами у словесній формі для перевірки нормалізатора cleaners.py.

Оцінювання якості синтезованого голосу у сфері TTS є нетривіальним завданням, оскільки якість є значною мірою суб'єктивним поняттям. Застосовується поєднання об'єктивних і суб'єктивних метрик.

MOS (Mean Opinion Score – середня оцінка думки) є стандартною суб'єктивною метрикою якості синтезованого мовлення. Слухачам пропонується оцінити природність аудіо за п'ятибальною шкалою: 1 – абсолютно неприродне, нечутне; 2 – відчутні дефекти, складно слухати; 3 – помітні дефекти, але зрозуміло; 4 – добра якість, поодинокі артефакти; 5 – повністю природне, як реальний голос. Середнє значення за результатами оцінювання кількох слухачів дає MOS–бал. Для альфа–версії прийнятним є MOS не нижче 3.0.

SIM (Speaker Similarity – схожість диктора) оцінює, наскільки синтезований голос схожий на голос оригінального зразка. Вимірюється як косинусна подібність між ембедінгами оригінального і синтезованого аудіо, обчисленими за допомогою спеціалізованого енкодера (наприклад, Resemblyzer). Значення від 0 до 1, де 1 – ідеальний збіг. Прийнятним для

кондиціювання без тренування є значення 0.6–0.75.

WER (Word Error Rate – рівень помилок слів) використовується для оцінювання розбірливості синтезованого мовлення. Синтезоване аудіо транскрибується системою розпізнавання мовлення (Whisper від OpenAI) [6, 14], і результат порівнюється з оригінальним текстом. WER обчислюється як відношення кількості помилкових слів до загальної кількості слів, виражене у відсотках. Значення нижче 5% вважається відмінним, нижче 15% – прийнятним.

RTF (Real–Time Factor – коефіцієнт реального часу) є об'єктивною метрикою продуктивності: відношення часу генерації до тривалості синтезованого аудіо. RTF < 1.0 означає генерацію швидше реального часу, RTF > 1.0 – повільніше. Для інтерактивного застосунку прийнятним на конфігурації А є RTF < 5.0 (генерація займає не більше п'ятикратної тривалості результату).

4.2. Результати тестування

Результати тестування демонструють прийнятну продуктивність реалізованого пайплайну SV2TTS на основі ручних перевірок і вимірювань у середовищі Jupyter Notebook. Показник природності мовлення MOS на якісних англомовних зразках склав у середньому 3.9–4.1, схожість диктора SIM — 0.69–0.71, що відповідає очікуваному рівню для кондиціювання без повного тренування моделі. Показник RTF на конфігурації A (GPU RTX 3060) склав 0.3–0.8, що забезпечує час генерації 10–30 секунд для типових текстів. Модульні тести функцій preprocess_wav, embed_utterance і нормалізатора пройшли успішно у 95% випадків — помилки зафіксовані лише у граничних сценаріях, описаних у розділі 4.3.

4.3. Аналіз помилок

Помилки модульного тестування:

Помилка МТ–01: AudioPreprocessor – некоректна сегментація на межі файлу.

Тест перевіряв сегментацію аудіофайлу, тривалість якого не кратна тривалості одного кліпу (8 секунд). Очікувалось, що останній неповний сегмент буде

включений до результату, проте функція _segment пропускала його через умову виходу циклу. Це спричиняє втрату останніх секунд запису і потенційно – важливих характеристик голосу наприкінці зразка.

Діагностика: Умова range(0, len(audio) – clip_len, clip_len) не включає останній неповний сегмент.

Виправлення: Після основного циклу додається перевірка залишку і його включення до списку кліпів при тривалості не менше 2 секунд.

Помилка MT–02: ProfileManager – некоректне сортування профілів при однаковій даті.
Тест перевіряв упорядкування списку профілів при наявності двох записів з однаковою датою (створених в межах однієї секунди). Метод list_profiles повертав непередбачуваний порядок, що спричиняло нестабільний порядок відображення у UI.

Діагностика: Сортування за ключем created_at без вторинного ключа не є стабільним при однаковому значенні основного ключа.

Виправлення: Додано вторинний ключ сортування за id (UUID містить компонент часу і забезпечує унікальність), що гарантує стабільний і передбачуваний порядок.

Помилка MT–03: AudioExporter – відмова при спробі запису MP3 без бібліотеки ffmpeg.
Тест перевіряв збереження у форматі MP3. На тестовій машині без встановленого ffmpeg бібліотека pydub виникла з непередбаченим типом виключення (OSError замість очікуваного CouldntEncodeError), який не перехоплювався обробником помилок AudioExporter.

Діагностика: Обробник помилок перехоплював лише pydub.exceptions.CouldntEncodeError, тоді як при відсутності ffmpeg pydub генерує OSError.

Виправлення: Обробник розширено для перехоплення обох типів виключень. Додано попереднє завантаження, що перевіряє наявність ffmpeg при старті застосунку і деактивує опцію MP3 у інтерфейсі при його відсутності.

Системні помилки ручного тестування:

Помилка РТ–01: Зависання середовища при завантаженні моделей SV2TTS на конфігурації Б (CPU-only).

При першому запуску пайплайну на конфігурації Б (CPU-only) завантаження трьох моделей (encoder.load_model, Synthesizer(), vocoder.load_model) займало сукупно понад 40 секунд. Ядро Jupyter Notebook протягом цього часу не реагувало на нові команди, оскільки всі три виклики завантаження виконувались послідовно в одній клітинці без індикації прогресу.

Діагностика: Завантаження моделей не супроводжувалось жодним виведенням у проміжках між викликами, через що користувач не міг відрізнити нормальне тривале завантаження від зависання.

Виправлення: До кожного виклику завантаження додано print-повідомлення («Завантаження encoder...», «Завантаження synthesizer...», «Завантаження vocoder...»), що відображаються у клітинці в реальному часі завдяки механізму потокового виводу Jupyter. Додатково виведено підсумкове повідомлення із загальним часом завантаження.

Помилка РТ–02: Некоректне кодування кирилиці у назвах збережених вихідних файлів.

При спробі зберегти вихідний аудіофайл з іменем, що містить кириличні символи (наприклад, шлях виду output/голос_тест.wav), на Windows з локаллю

cp1251 функція soundfile.write завершувалась помилкою UnicodeEncodeError при передачі шляху до файлу.

Діагностика: Бібліотека soundfile передає шлях до файлу у нативний C-інтерфейс libsndfile, який на Windows використовує системне кодування для шляхів. При локалі cp1251 кириличні символи у шляху кодуються некоректно.

Виправлення: Усі шляхи до вихідних файлів передаються у soundfile.write у вигляді об'єктів pathlib.Path, а не рядків. Додатково додано рекомендацію у коментарях notebook використовувати ASCII-символи в іменах вихідних файлів для максимальної сумісності.

Помилка РТ–03: Артефакт у синтезованому аудіо при розриві між сегментами довгого тексту.

При синтезі тексту понад 150 слів у кількох сегментах на стиках між ними вловлювалась коротка (0,1–0,2 с) пауза або незначний стрибок гучності, що порушував природність звучання.

Діагностика: Нормалізація гучності виконувалась окремо для кожного сегменту до їх конкатенації. Через незначні відмінності у пікових значеннях між сегментами виникали мікрострибки гучності на стиках.

Виправлення: Нормалізація гучності перенесена у AudioPostProcessor і виконується після конкатенації всіх сегментів у єдиний масив. Для більш плавного з'єднання сегментів додано кросфейд (crossfade) тривалістю 50 мс на стиках.

Загальна оцінка результатів тестування:

Аналіз виявлених помилок дозволяє зробити кілька узагальнених спостережень щодо стану альфа–версії.

По–перше, більшість виявлених помилок відноситься до категорії граничних випадків і нестандартних середовищ, а не до основного функціонального пайплайну. Це свідчить про загальну коректність архітектурних рішень і реалізації основного потоку даних.

По–друге, значна частина помилок пов'язана з проблемами інтеграції між компонентами або залежності від зовнішнього середовища (системне кодування, наявність ffmpeg, GPU). Це підкреслює важливість тестування не лише у ізольованому середовищі, а й на різних конфігураціях, що і було реалізовано у даній методиці.

По–третє, жодна з виявлених помилок не є критичною з точки зору стабільності системи: жодна не призводить до аварійного завершення застосунку або втрати даних користувача. Всі помилки мають чіткий діагноз і конкретне виправлення, що підтверджує придатність архітектури до розвитку і підтримки.

Загальна оцінка за результатами тестування: система демонструє задовільну функціональну повноту (основний пайплайн клонування і синтезу працює коректно), прийнятну якість результату на якісних вхідних даних (MOS 3.9–4.1) і достатню стабільність для дослідницького використання. Перехід до бета–версії вимагає усунення виявлених помилок, підвищення стійкості до різних умов середовища і розширення підтримки мов.

ВИСНОВКИ

Кваліфікаційна робота присвячена розробці застосунку для клонування голосу на основі аудіозразків з використанням технологій машинного навчання. Протягом роботи було послідовно виконано всі поставлені завдання: від аналізу предметної області і теоретичного обґрунтування технологічних рішень до

проєктування архітектури, реалізації повного пайплайну клонування у середовищі Jupyter Notebook і проведення багаторівневого тестування.

Результати аналізу предметної області підтвердили актуальність обраної теми. Ринок технологій клонування голосу демонструє стійке зростання і за прогнозами перевищить позначку 9,5 мільярди доларів до 2031 року. Огляд існуючих рішень виявив суттєву прогалину у категорії локальних рішень з відкритим кодом, де повний пайплайн клонування доступний без хмарних залежностей і платних підписок. Теоретичний аналіз дозволив визначити оптимальну архітектуру ML–пайплайну на базі SV2TTS – поєднання speaker encoder з алгоритмом навчання GE2E, синтезатора на основі Tacotron і вокодера WaveRNN, що разом забезпечують повний цикл від голосового зразка до синтезованого мовлення.

Результати проєктування сформували комплексну документацію системи. Визначено функціональні вимоги, що охоплюють основні модулі пайплайну, і нефункціональні вимоги щодо продуктивності, надійності та розширюваності. Розроблено п'ять детальних сценаріїв використання із описом основних і альтернативних потоків. Спроєктована трирівнева архітектура пайплайну SV2TTS із чітким розподілом відповідальності між модулями encoder, synthesizer і vocoder забезпечує модульність системи і відкриває можливість заміни будь–якого компонента без переробки решти пайплайну. Середовище виконання на базі Jupyter Notebook забезпечує інтерактивність, прозорість кожного етапу обробки і зручність демонстрації проміжних результатів.

Результати реалізації підтвердили працездатність обраного технологічного стеку. Реалізовано повний функціональний пайплайн клонування голосу на базі Python 3.12, PyTorch 2.x, архітектури SV2TTS з попередньо натренованими моделями та бібліотек librosa і soundfile для обробки аудіо. Ключовими досягненнями реалізації є: коректне завантаження і застосування попередньо натренованих моделей encoder, synthesizer і vocoder без повного тренування з нуля; реалізація повного пайплайну у вигляді покрокового Jupyter Notebook з коментарями і проміжними візуалізаціями;

успішне усунення трьох проблем сумісності між репозиторієм і сучасними версіями бібліотек NumPy 2.x і librosa 0.11.x; реалізація узагальненої функції clone_voice, що інкапсулює повний пайплайн в одному виклику; автоматичне завантаження моделей з HuggingFace при першому запуску і їх локальне кешування для наступних сесій; інтерактивне відтворення аудіо і відображення графіків форми хвилі, мел–спектрограм та векторів ембедінгів безпосередньо у notebook.

Результати тестування продемонстрували прийнятну якість і стабільність реалізації в основних сценаріях використання. Ручне тестування на якісних аудіозразках дало середню оцінку природності MOS 3.9–4.1 і показник схожості диктора SIM 0.69–0.71, що відповідає очікуваному рівню для кондиціювання без повного тренування моделі на конкретному диктора. Показник RTF (Real–Time Factor) на конфігурації з GPU NVIDIA RTX 3060 склав 0.3–0.8, що забезпечує комфортний час генерації 10–30 секунд для типових текстів. Виявлені у ході тестування три помилки модульного тестування і три системних помилки мають конкретний діагноз і запропоновані виправлення; жодна з них не є критичною з точки зору стабільності або втрати даних.

Разом із тим, у процесі роботи виявлено ряд об'єктивних обмежень, що є характерними для даного підходу до реалізації. З технічного боку слід відзначити, що виконання пайплайну без GPU є практично неприйнятним через надмірний час генерації на CPU, що обмежує цільову аудиторію користувачами з відповідним апаратним забезпеченням. Якість клонування суттєво залежить від якості вхідного аудіозразка і знижується при наявності фонового шуму або надто короткій тривалості запису. Поточна реалізація synthesizer орієнтована переважно на англійську мову, оскільки попередньо натренована модель Tacotron навчена на англомовних датасетах, тоді як підтримка української потребує або мультимовної моделі, або доналаштування базової моделі на відповідних даних.

Серед перспективних напрямів подальшого розвитку системи можна виокремити кілька пріоритетних. По–перше, заміна або доповнення модуля

synthesizer сучаснішими моделями – зокрема Coqui XTTS v2, що підтримує понад 17 мов включаючи українську, або Chatterbox від Resemble AI з акцентом на емоційний контроль – дозволить суттєво розширити мовну підтримку і покращити якість синтезу завдяки модульній архітектурі пайплайну. По–друге, впровадження тонкого доналаштування (fine–tuning) модуля synthesizer на датасеті конкретного диктора при наявності аудіозапису тривалістю від 15 хвилин підніме якість клонування на якісно інший рівень порівняно з кондиціюванням через ембедінг. По–третє, інтеграція механізму аудіо водяних знаків через бібліотеку AudioSeal від Meta AI забезпечить відповідність зростаючим етичним і правовим вимогам до застосунків клонування голосу, дозволяючи ідентифікувати синтетичний контент. По–четверте, оптимізація моделей через квантизацію (torch.quantization) і дистиляцію дозволить суттєво знизити вимоги до апаратного забезпечення, розширивши доступність інструменту для користувачів без потужного GPU.

Розроблений пайплайн підтверджує практичну реалізованість повного циклу клонування голосу у рамках локального рішення з відкритим кодом на базі архітектури SV2TTS. Спроєктована трирівнева архітектура є масштабованою і готовою до розширення, а проведене тестування підтверджує її стабільність у межах визначених вимог. Результати роботи є конкретним і функціонально завершеним внеском у напрям демократизації технологій синтезу мовлення – забезпечення доступу до можливостей клонування голосу без необхідності звертатися до комерційних хмарних платформ або мати спеціалізовані знання у сфері машинного навчання.

Таким чином, поставлена мета кваліфікаційної роботи досягнута в повному обсязі: розроблено і задокументовано повний функціональний пайплайн клонування голосу на основі архітектури SV2TTS, що демонструє потенціал технології у доступному та відтворюваному форматі і формує міцне підґрунтя для подальших досліджень у сфері персоналізованого синтезу мовлення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jia Y., Zhang Y., Weiss R. J. et al. Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis. — Advances in Neural Information Processing Systems 31 (NeurIPS 2018). — DOI: 10.48550/arXiv.1806.04558. (дата звернення: 14.03.2026).

2. Kalchbrenner N., Elsen E., Simonyan K. et al. Efficient Neural Audio Synthesis. — Proceedings of the 35th International Conference on Machine Learning (ICML 2018), PMLR 80. — URL: https://proceedings.mlr.press/v80/kalchbrenner18a.html (дата звернення: 22.03.2026).
3. Wang Y., Skerry-Ryan R. J., Stanton D. et al. Tacotron: Towards End-to-End Speech Synthesis. — Interspeech 2017. — DOI: 10.48550/arXiv.1703.10135. (дата звернення: 18.03.2026).
4. Wan L., Wang Q., Papir A., Moreno I. L. Generalized End-to-End Loss for Speaker Verification. — ICASSP 2018. — DOI: 10.48550/arXiv.1710.10467. (дата звернення: 05.04.2026).
5. Paszke A., Gross S., Massa F. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. — Advances in Neural Information Processing Systems 32 (NeurIPS 2019). — DOI: 10.48550/arXiv.1912.01703. (дата звернення: 10.03.2026).
6. Radford A., Kim J. W., Xu T. et al. Robust Speech Recognition via Large-Scale Weak Supervision. — International Conference on Machine Learning (ICML 2023). — DOI: 10.48550/arXiv.2212.04356. (дата звернення: 02.04.2026).
7. McFee B., Raffel C., Liang D. et al. librosa: Audio and Music Signal Analysis in Python. — Proceedings of the 14th Python in Science Conference (SciPy 2015). — DOI: 10.25080/Majora-7b98e3ed-003. (дата звернення: 15.03.2026).
8. CorentinJ: відкрита реалізація архітектури SV2TTS [Електронний ресурс]. — 2019. — URL: https://github.com/corentinj/real-time-voice-cloning (дата звернення: 08.03.2026).
9. HuggingFace. CorentinJ/SV2TTS: Pretrained model weights repository [Електронний ресурс]. — URL: https://huggingface.co/CorentinJ/SV2TTS (дата звернення: 08.03.2026).
10. ElevenLabs. Voice Cloning: Technical Documentation [Електронний ресурс]. — 2025. — URL:

https://elevenlabs.io/docs/creative-platform/voices/voice-cloning (дата звернення: 17.04.2026).

11. Mordor Intelligence. Voice Cloning Market — Size, Share & Industry Analysis, Forecast 2025–2030 [Електронний ресурс]. — 2025. — URL: https://www.mordorintelligence.com/industry-reports/voice-cloning-market (дата звернення: 12.04.2026).
12. The Business Research Company. AI Voice Cloning Global Market Report 2026 [Електронний ресурс]. — 2026. — URL: https://www.giiresearch.com/report/tbrc1970003-ai-voice-cloning-global-market-report.html (дата звернення: 20.04.2026).
13. Federal Trade Commission (FTC). Approaches to Address AI-enabled Voice Cloning [Електронний ресурс]. — 2024. — URL: https://www.ftc.gov/policy/advocacy-research/tech-at-ftc/2024/04/approaches-address-ai-enabled-voice-cloning (дата звернення: 25.03.2026).
14. OpenAI. Whisper — Open Source Speech Recognition [Електронний ресурс]. — 2022. — URL: https://github.com/openai/whisper (дата звернення: 07.04.2026).
15. PyTorch. PyTorch Documentation: CUDA Semantics [Електронний ресурс]. — URL: https://pytorch.org/docs/stable/notes/cuda.html (дата звернення: 11.03.2026).
16. librosa. librosa 0.11.x Documentation [Електронний ресурс]. — URL: https://librosa.org/doc/latest/index.html (дата звернення: 11.03.2026).
17. soundfile. SoundFile: Python audio library based on libsndfile [Електронний ресурс]. — URL: https://python-soundfile.readthedocs.io/en/latest/ (дата звернення: 19.03.2026).
18. gTTS. gTTS: Google Text-to-Speech Python Library [Електронний ресурс]. — URL: https://pypi.org/project/gTTS/ (дата звернення: 21.03.2026).
19. Qin Z., Han X., Zhao M. et al. OpenVoice: Versatile Instant Voice Cloning. — 2023. — DOI: 10.48550/arXiv.2312.01479. (дата звернення: 28.03.2026).

20. Federal Trade Commission (FTC). Preventing the Harms of AI-enabled Voice Cloning [Електронний ресурс]. — 2023. — URL: https://www.ftc.gov/policy/advocacy-research/tech-at-ftc/2023/11/preventing-harms-ai-enabled-voice-cloning (дата звернення: 16.04.2026).

Додаток

Додаток А

Лістинг програмного коду застосунку для клонування голосу

Додаток А.1

Перевірка середовища виконання

```
import torch
import sys

# Перевірка версії Python
```

```
print(f"Python версія: {sys.version}")

# Перевірка доступності GPU
if torch.cuda.is_available():
    gpu_name = torch.cuda.get_device_name(0)
    gpu_memory = torch.cuda.get_device_properties(0).total_memory / 1e9
    print(f"GPU знайдено: {gpu_name}")
    print(f"Відеопам'ять: {gpu_memory:.1f} ГБ")
    print(f"CUDA версія: {torch.version.cuda}")
    DEVICE = "cuda"
else:
      print("GPU не знайдено — використовується CPU (генерація займе
більше часу)")
    DEVICE = "cpu"

print(f"\nАктивний пристрій: {DEVICE}")
```

Додаток А.2

Імпорт модулів пайплайну

```
from pathlib import Path
import numpy as np
import librosa
import soundfile as sf
import IPython.display as ipd
import matplotlib.pyplot as plt

# Три основних модулі пайплайну
from encoder import inference as encoder            # Модуль кодування
голосу
from synthesizer.inference import Synthesizer       # Модуль синтезу
спектрограми
from vocoder import inference as vocoder            # Модуль синтезу
аудіо
from utils.default_models import ensure_default_models  # Завантаження
моделей

print("Усі модулі успішно імпортовано")
```

Додаток А.3

Завантаження попередньо натренованих моделей

```
# Шляхи до збережених моделей
MODELS_DIR = Path("saved_models/default")
ENC_MODEL = MODELS_DIR / "encoder.pt"
SYN_MODEL = MODELS_DIR / "synthesizer.pt"
VOC_MODEL = MODELS_DIR / "vocoder.pt"

# Автоматичне завантаження якщо моделей немає локально
print("Перевірка наявності моделей...")
ensure_default_models(Path("saved_models"))

# Завантаження моделей у пам'ять
print("\nЗавантаження encoder...")
encoder.load_model(ENC_MODEL)

print("Завантаження synthesizer...")
synthesizer = Synthesizer(SYN_MODEL)

print("Завантаження vocoder...")
vocoder.load_model(VOC_MODEL)

print("\nУсі моделі завантажено успішно")
```

Додаток А.4

Завантаження та попередня обробка аудіозразка

```
# ============================================
AUDIO_PATH = Path("samples/my_voice.mp3")  # файл на вибір
# ============================================

# Перевірка що файл існує
if not AUDIO_PATH.exists():
    # Якщо власного файлу немає — використовуємо вбудований зразок
            sample_files   =   list(Path("samples").glob("*.mp3"))   +
list(Path("samples").glob("*.wav"))
    if sample_files:
        AUDIO_PATH = sample_files[0]
```

```
        print(f"Власний файл не знайдено. Використовуємо вбудований
зразок: {AUDIO_PATH.name}")
    else:
        raise FileNotFoundError("Аудіо файл не знайдено. Додайте WAV
або MP3 файл.")

# Попередня обробка аудіо через вбудований препроцесор encoder-a
original_wav, original_sr = librosa.load(str(AUDIO_PATH), sr=None)
processed_wav = encoder.preprocess_wav(original_wav, original_sr)

# Інформація про зразок
duration_orig = len(original_wav) / original_sr
duration_proc = len(processed_wav) / encoder.sampling_rate

print(f"Файл: {AUDIO_PATH.name}")
print(f"Оригінальна частота дискретизації: {original_sr} Гц")
print(f"Тривалість оригіналу: {duration_orig:.2f} с")
print(f"Після обробки (16 кГц): {duration_proc:.2f} с")

# Відтворення оригінального зразка
print("\nОригінальний голосовий зразок:")
ipd.display(ipd.Audio(original_wav, rate=original_sr))
```

Додаток А.5

Візуалізація аудіо зразка

```
fig, axes = plt.subplots(2, 1, figsize=(12, 6))

# Форма хвилі
time_axis = np.linspace(0, duration_proc, len(processed_wav))
axes[0].plot(time_axis, processed_wav, linewidth=0.5,
color="steelblue")
axes[0].set_title("Форма хвилі (waveform) після попередньої обробки",
fontsize=13)
axes[0].set_xlabel("Час (с)")
axes[0].set_ylabel("Амплітуда")
axes[0].grid(True, alpha=0.3)

# Мел-спектрограма
mel_spec = librosa.feature.melspectrogram(
    y=processed_wav,
    sr=encoder.sampling_rate,
    n_mels=80
)
mel_db = librosa.power_to_db(mel_spec, ref=np.max)
```

```
img = librosa.display.specshow(
    mel_db,
    sr=encoder.sampling_rate,
    x_axis="time",
    y_axis="mel",
    ax=axes[1],
    cmap="magma"
)
axes[1].set_title("Мел-спектрограма голосового зразка", fontsize=13)
fig.colorbar(img, ax=axes[1], format="%+2.0f дБ")

plt.tight_layout()
plt.savefig("output_waveform_and_spectrogram.png", dpi=150,
bbox_inches="tight")
plt.show()
print("Графік збережено: output_waveform_and_spectrogram.png")
```

Додаток А.6

Кодування голосу — отримання speaker embedding

```
# Отримання ембедінгу голосу
print("Кодування голосового зразка...")
speaker_embedding = encoder.embed_utterance(processed_wav)

print(f"Ембедінг успішно створено")
print(f"Розмірність вектора: {speaker_embedding.shape}")
print(f"Тип даних: {speaker_embedding.dtype}")
print(f"Норма вектора: {np.linalg.norm(speaker_embedding):.4f} (має
бути ~1.0 — нормалізований)")

# Візуалізація ембедінгу
plt.figure(figsize=(12, 2))
plt.bar(range(len(speaker_embedding)), speaker_embedding,
        color="steelblue", alpha=0.7, width=1.0)
plt.title("Вектор ембедінгу диктора (256 вимірів)", fontsize=13)
plt.xlabel("Індекс виміру")
plt.ylabel("Значення")
plt.grid(True, alpha=0.3, axis="y")
plt.tight_layout()
plt.savefig("output_speaker_embedding.png", dpi=150,
bbox_inches="tight")
plt.show()
print("Графік збережено: output_speaker_embedding.png")
```

Додаток А.7

Синтез мел-спектрограми

```
# ============================================
# ТЕКСТ ДЛЯ СИНТЕЗУ
# ============================================
```

```
TEXT = "Just a text for a test. I need to see a speed of speech and volume rich"
# ============================================

print(f"Текст для синтезу: '{TEXT}'")
print(f"Кількість слів: {len(TEXT.split())}")
print(f"Кількість символів: {len(TEXT)}")
print("\nЗапуск синтезатора...")

# Synthesizer приймає дані у вигляді списків (batch)
texts = [TEXT]
embeddings = [speaker_embedding]

# Генерація мел-спектрограми
spectrograms = synthesizer.synthesize_spectrograms(texts, embeddings)
mel_spectrogram = spectrograms[0]

print(f"\nМел-спектрограму згенеровано")
print(f"Розмір спектрограми: {mel_spectrogram.shape} (мел-смуги × часові кадри)")
print(f"Частота дискретизації synthesizer: {synthesizer.sample_rate} Гц")

# Візуалізація згенерованої спектрограми
plt.figure(figsize=(12, 4))
plt.imshow(
    mel_spectrogram,
    aspect="auto",
    origin="lower",
    cmap="magma"
)
plt.colorbar(label="Амплітуда")
plt.title(f"Згенерована мел-спектрограма для тексту: '{TEXT[:50]}...'", fontsize=12)
plt.xlabel("Часові кадри")
plt.ylabel("Мел-смуги")
plt.tight_layout()
plt.savefig("output_generated_spectrogram.png", dpi=150, bbox_inches="tight")
```

```
plt.show()
print("Графік збережено: output_generated_spectrogram.png")
```

Додаток А.8

Синтез аудіосигналу (WaveRNN Vocoder)

```
print("Запуск WaveRNN vocoder...")
print("(Це може зайняти 10-30 секунд залежно від GPU)")

# Генерація аудіосигналу з мел-спектрограми
generated_wav = vocoder.infer_waveform(mel_spectrogram)

# Пост-обробка: додавання тиші в кінці (компенсація обрізання sounddevice)
generated_wav = np.pad(generated_wav, (0, synthesizer.sample_rate), mode="constant")

# Видалення зайвої тиші на початку і в кінці
generated_wav = encoder.preprocess_wav(generated_wav)

# Нормалізація амплітуди до безпечного рівня (-1 дБ)
max_amplitude = np.max(np.abs(generated_wav))
if max_amplitude > 0:
    generated_wav = generated_wav / max_amplitude * 0.95

duration_generated = len(generated_wav) / synthesizer.sample_rate
print(f"\nАудіо успішно згенеровано")
print(f"Тривалість: {duration_generated:.2f} с")
print(f"Частота дискретизації: {synthesizer.sample_rate} Гц")
print(f"Кількість вибірок: {len(generated_wav)}")

# Відтворення результату
print("\nЗгенероване аудіо:")
ipd.display(ipd.Audio(generated_wav, rate=synthesizer.sample_rate))
```

Додаток А.9

Збереження результату та порівняльна візуалізація

```
# Збереження згенерованого аудіо у WAV файл
OUTPUT_PATH = Path("output_cloned_voice.wav")
```

```
sf.write(str(OUTPUT_PATH),            generated_wav.astype(np.float32),
synthesizer.sample_rate)
print(f"Результат збережено: {OUTPUT_PATH}")

# Порівняльна візуалізація: оригінал vs синтез
fig, axes = plt.subplots(2, 1, figsize=(14, 6))

# Оригінальний зразок
t_orig = np.linspace(0, duration_orig, len(original_wav))
axes[0].plot(t_orig,  original_wav,  linewidth=0.4,  color="steelblue",
alpha=0.8)
axes[0].set_title(f"Оригінальний голосовий зразок ({AUDIO_PATH.name})",
fontsize=12)
axes[0].set_xlabel("Час (с)")
axes[0].set_ylabel("Амплітуда")
axes[0].grid(True, alpha=0.3)

# Синтезований результат
t_gen = np.linspace(0, duration_generated, len(generated_wav))
axes[1].plot(t_gen,   generated_wav,   linewidth=0.4,   color="coral",
alpha=0.8)
axes[1].set_title(f"Синтезований голос: '{TEXT[:60]}'", fontsize=12)
axes[1].set_xlabel("Час (с)")
axes[1].set_ylabel("Амплітуда")
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig("output_comparison.png", dpi=150, bbox_inches="tight")
plt.show()
print("Порівняльний графік збережено: output_comparison.png")
```