Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка програмного забезпечення для автоматичної розмітки звукових файлів»

Виконав студент IV курсу
групи ЗПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(число й назва напряму підготовки, спеціальності)
Дажура О.Р.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Ткаченко О.М.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. ОТ Кадук О. В.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ Романюк О. Н.

«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк

«21» березня 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Дажурі Олегу Романовичу

1. Тема роботи – Розробка програмного забезпечення для автоматичної розмітки звукових файлів.

Керівник роботи: Ткаченко Олександр Миколайович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: методи обробки аудіо алгоритм Silero VAD для виявлення мовлення, алгоритм DeepFilterNet2 для шумозаглушення, модель Whisper для транскрибування мовлення; графічний режим matplotlib для візуалізації аудіо сигналу; вихідні дані для обробки звукові файли формату .mp3 .wav .flac .ogg .m4a .aac .wma .aiff, .aif .opus, які конвертуються у WAV з частотою дискретизації 16000 Гц; дані про параметри обробки настройки для шумозаглушення, покращення якості звуку, транскрибування мовлення; координати для аналізу звукових файлів сегменти мовлення, музики, тиші; вихідні дані очищені звукові файли, текстові транскрипції у форматі SRT або TXT, графічні зображення аудіо сигналу.

4. Зміст текстової частини: аналіз вхідних даних формат звукових файлів, частота дискретизації, алгоритми класифікації для мовлення, музики, шуму; опис алгоритмів обробки Silero VAD, DeepFilterNet2, Whisper; оптимізація якості звуку шумозаглушення, покращення звуку, виділення мовлення; розробка інтерфейсу користувача зручність вибору функцій, налаштування параметрів, адаптація до різних запитів; виведення результатів транскрипція мовлення, очищення звукових файлів, графічне відображення звукових доріжок; тестування та ефективність роботи – перевірка точності алгоритмів на реальних даних, підвищення ефективності взаємодії з користувачем.

5. Перелік ілюстративної частини: галузі застосування розмітки; алгор
обробки мовлення та звуку, принципи шумозаглушення та покращення я
звукових файлів; основні етапи обробки звукових файлів детекція мовл
очищення, покращення якості, транскрибування; графічне відображ
аудіосигналу спектрограми, амплітуди звукових доріжок; інтерфейс прог
зручність для користувача, адаптація до різних задач, настройка параме
тестування програми перевірка на реальних даних, оцінка ефективності резуль
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдан прийн
1-4	Ткаченко О. М., к.т.н., доцент кафедри ПЗ	24.03.2025	01.06.2

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Прим
1	Аналіз сучасних методів виявлення мовлення, шумозаглушення та транскрипції звукових файлів	25.03.25 – 06.04.25	вик
2	Розробка комбінованої системи виявлення мовлення	07.04.25 – 21.04.25	вик
3	Розробка модуля шумозаглушення	22.04.25 – 06.05.25	вик
4	Розробка модуля транскрипції мовлення	07.05.25 – 19.05.25	вик
5	Оформлення матеріалів до захисту БКР	20.05.25 – 30.05.25	вик

Студент _____ **Дажура**
(підпис) (ініціали та п

Керівник бакалаврської кваліфікаційної роботи _____ **Ткаченко О**
(підпис) (ініціали та п

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 55 сторінок формату А4, на яких є 27 рисунків, 3 таблиці, список використаних джерел містить 30 найменувань.

У бакалаврській кваліфікаційній роботі розроблено програмне забезпечення для автоматизованої розмітки звукових файлів із використанням сучасних моделей штучного інтелекту. Проведено аналіз методів класифікації та транскрипції аудіо, а також визначено основні виклики при обробці мовлення в шумових умовах.

Запропоновано архітектуру десктопного застосунку, що реалізує поетапну обробку аудіо: виявлення мовлення (Silero VAD, YAMNet), шумозаглушення (DeepFilterNet2) та транскрипцію (Whisper). У роботі вперше комбіновано два підходи до виявлення мовлення – Silero VAD та YAMNet – що дозволяє точніше визначати межі мовних сегментів.

Застосунок реалізовано мовою програмування Python у середовищі Visual Studio Code. Для реалізації основних функцій використано open-source рішення: Silero VAD, YamNet, DeepFilterNet2, Whisper. Результати візуалізуються з можливістю фільтрації та прослуховування.

Експериментальне тестування показало зниження середнього показника Word Error Rate на 15-25% у порівнянні з транскрипцією без попередньої обробки. Застосунок має потенціал для подальшого розвитку, зокрема в напрямках контекстної автокорекції мовлення, семантичного поділу тексту та адаптації моделей до користувача.

Ключові слова: розмітка аудіо, звукова обробка, транскрипція, шумозаглушення, штучний інтелект, Python, мовлення, Whisper, VAD, програмне забезпечення.

ANNOTATION

The bachelor's thesis consists of 55 pages in A4 format, containing 27 figures, 3 tables, and a list of 30 references.

In his bachelor's thesis, he developed software for automated markup of audio files using modern artificial intelligence models. An analysis of audio classification and transcription methods was conducted, and the main challenges in processing speech in noisy conditions were identified.

The architecture of a desktop application that implements step-by-step audio processing is proposed: speech detection (Silero VAD, YAMNet), noise reduction (DeepFilterNet2), and transcription (Whisper). For the first time, we combined two approaches to speech detection - Silero VAD and YAMNet - which allows us to more accurately determine the boundaries of speech segments.

The application is implemented in the Python programming language in the Visual Studio Code environment. Open-source solutions were used to implement the main functions: Silero VAD, YamNet, DeepFilterNet2, Whisper. The results are visualized with the ability to filter and listen.

Experimental testing has shown a 15-25% reduction in the average Word Error Rate compared to transcription without pre-processing. The application has the potential for further development, in particular in the areas of contextual speech autocorrection, semantic text separation, and user-specific model adaptation.

Keywords: audio markup, audio processing, transcription, noise reduction, artificial intelligence, Python, speech, Whisper, VAD, softwar

ЗМІСТ

ВСТУП....4
1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ ОБРОБКИ ЗВУКОВИХ ФАЙЛІВ І ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ....8
1.1 Аналіз сучасних технологій обробки звукових файлів....8
1.2 Порівняльний аналіз аналогів....9
1.3 Аналіз методів розв'язання задачі....15
1.4 Постановка задач....16
1.5 Висновки....17
2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ....18
2.1 Формалізація вхідних даних системи....18
2.2 Розробка моделі системи....19
2.3 Розробка алгоритмів роботи системи....22
2.5. Розробка архітектури системи....26
2.6. Розробка комбінованого методу сегментації звукових даних....28
2.7 Висновки....29
3. РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ СИСТЕМИ АВТОМАТИЗОВАНОЇ ОБРОБКИ ЗВУКОВИХ ФАЙЛІВ....30
3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення....30
3.2 Розробка модуля виявлення типів звуків....31
3.3 Розробка модуля для видалення шумів та тиші зі звукових файлів....35
3.4 Розробка модуля для транскрипції мовлення в текстовий формат....37
3.5 Розробка інтерфейсу програмного застосунку....39
3.6 Висновки....43
4 ТЕСТУВАННЯ ПРОГРАМИ....44
4.1 Тестування системи....44
4.2 Результати тестування програмного забезпечення....45
4.2 Розробка інструкції користувача....52
4.3 Застосування результатів аудіорозмітки....54
4.4 Висновки....56
ВИСНОВКИ....57

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 59
ДОДАТКИ 62
Додаток А – Технічне завдання 63
Додаток Б – Протокол перевірки роботи на плагіат 64
Додаток В – Лістинг програми 65
Додаток Г – Ілюстративна частина 91

ВСТУП

У сучасному світі, де штучний інтелект і машинне навчання розвиваються надзвичайно швидко, аудіолейблінг (розмітка аудіо) набуває особливого значення. Ця технологія активно впроваджується в найрізноманітніші сфери – від медицини та автомобільної промисловості до фінансів, агросектора й індустрії розваг. Основна ідея аудіолейблінгу полягає в детальному маркуванні аудіозаписів для підвищення ефективності навчання AI-моделей. Це може включати розшифрування мовлення, розпізнавання емоцій, визначення особистості мовця, виділення фонових шумів або класифікацію звукових фрагментів [1].

Однією з основних проблем при роботі з великими обсягами звукових даних є ефективне та точне виявлення значущих подій у звуковому сигналі, таких як зміна тембру, початок і завершення мовних фрагментів, паузи чи інші акустичні ознаки. Ця задача є критично важливою для автоматичної розмітки звукових файлів і побудови систем розпізнавання подій у звуках. Сучасні підходи базуються на методах глибокого навчання та цифрової обробки сигналів, що дозволяє покращити точність і масштабованість обробки аудіоданих [2].

Ручна розмітка звукових файлів є трудомістким процесом, особливо при обробці великих аудіоархівів. Застосування автоматизованих методів дозволяє значно зменшити час на виконання розмітки та підвищити точність результатів [3].

Багато сучасних інструментів для розмітки звукових файлів мають низку обмежень, що ускладнює їх ефективне використання. Однією з головних проблем є обмежена функціональність, яка не завжди дозволяє одночасно виконувати транскрипцію та розмітку. Деякі інструменти також мають недостатню точність автоматичного розпізнавання, що змушує користувачів витрачати додатковий час на коригування. Важливим аспектом є відсутність комплексної класифікації аудіо, що обмежує можливості аналізу даних [4].

Для вирішення цих проблем необхідна розробка нових програмних інструментів для автоматизації процесу розмітки звукових файлів, які забезпечують високу точність завдяки інтеграції сучасних моделей машинного навчання та

пропонують зручний і адаптивний графічний інтерфейс для подальшої обробки результатів. Таким чином, тема бакалаврської кваліфікаційної роботи є актуальною.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалась згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення

Мета та завдання дослідження. Метою бакалаврської кваліфікаційної роботи є підвищення точності сегментації та транскрипції звукових файлів за рахунок розробки десктопного застосунку, який інтегрує каскадне виявлення мовлення, шумозаглушення та транскрипцію з використанням сучасних моделей штучного інтелекту. Це дозволить підвищити точність звукової розмітки та зменшити кількість помилок розпізнавання мовлення (Word Error Rate) на 15-25% у порівнянні з традиційними методами розмітки та транскрипції.

Відповідно до поставленої мети роботи необхідно виконати наступні завдання:

- провести аналіз сучасних методів виявлення мовлення, шумозаглушення та транскрипції звукових файлів із використанням моделей штучного інтелекту (Silero VAD, YAMNet, DeepFilterNet2, Whisper);
- спроектувати десктопний застосунок для автоматизованої розмітки звукових файлів із модулями аналізу, обробки, шумозаглушення та транскрипції;
- реалізувати комбіновану систему виявлення мовлення на основі моделей Silero VAD і YAMNet для точного визначення меж мовних сегментів;
- інтегрувати модель DeepFilterNet2 для шумозаглушення звукових файлів перед транскрипцією для підвищення якості мовних сегментів;
- розробити модуль транскрипції мовлення на основі моделі Whisper;
- провести експериментальну перевірку ефективності попередньої обробки (VAD + YAMNet + шумозаглушення) шляхом порівняння показників WER і CER із базовими методами транскрипції без попередньої обробки;
- розробити графічний інтерфейс застосунку для автоматизованої розмітки та візуалізації звукових сегментів;

• виконати тестування системи на реальних звукових файлах різної якості та тривалості для оцінки працездатності, точності сегментації та транскрипції.

Об’єкт дослідження. Об'єктом дослідження є процес автоматичної розмітки звукових файлів та їх транскрипції за допомогою методів обробки сигналів і машинного навчання.

Предмет дослідження. Предметом дослідження є методи та засоби автоматизованої розмітки звукових файлів, зокрема використання моделей глибокого навчання та детекторів активності голосу.

Методи дослідження. У процесі дослідження застосовувались: методи цифрової обробки сигналів для попередньої обробки аудіоданих, зокрема для фільтрації шумів та покращення якості звуку, а також методи машинного навчання для сегментації звукових файлів та автоматичної транскрипції мовлення, методи комп’ютерного моделювання для перевірки ефективності запропонованих рішень.

Новизна отриманих результатів

1. Отримала подальшого розвитку архітектура десктопного програмного забезпечення для локальної обробки звукових файлів, в якій, на відміну від існуючих, реалізовано поєднання каскадного виявлення мовлення (Silero VAD і YAMNet), шумозаглушення (DeepFilterNet2) та транскрипції (Whisper), що доволило зменшити кількість помилок розпізнавання мовлення (Word Error Rate) на 15-25% порівняно з транскрипцією без попередньої обробки.

2. Отримав подальшого розвитку метод сегментації звукових даних, в якому, на відміну від існуючих, використано комбінацію легкої моделі Silero VAD і глибокої моделі YAMNet, що дозволило підвищити точність визначення міжмовних сегментів у змішаних звукових файлах (мовлення, шум, музика).

Практичне значення результатів. Розроблене програмне забезпечення дозволить підвищити точність розмітки звукових файлів та транскрипції мовлення, що буде корисним для аудіоінженерів, дослідників, мовознавців та інших фахівців, що працюють з великими обсягами аудіоданих. Зокрема, застосування інструменту для автоматизованої розмітки допоможе знизити витрати часу на розмітку великих

аудіоархівів, підвищити точність аналізу та полегшити інтеграцію аудіообробки в різноманітні сфери діяльності.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать результати аналізу сучасного стану справ та технологій, архітектура програмної системи.

Апробація матеріалів кваліфікаційної роботи. Основні положення роботи доповідалися та обговорювалися на Всеукраїнській науково- технічній конференції підрозділів Вінницького національного технічного університету (Вінниця, 2025).

Публікації. Основні результати досліджень було опубліковано у матеріалах Всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету [5].

1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ ОБРОБКИ ЗВУКОВИХ ФАЙЛІВ І ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз сучасних технологій обробки звукових файлів

Обробка звукових файлів є важливим напрямком в інформаційних технологіях, який охоплює широкий спектр завдань, від підвищення якості звуку до автоматизованого аналізу аудіоданих, розмітку звукових даних з метою навчання моделей штучного інтелекту. За останні роки ця галузь зазнала значного прогресу, еволюціонувавши від базових методів обробки сигналів до використання складних систем штучного інтелекту та машинного навчання. Основні етапи обробки звуку включають захоплення сигналу, його фільтрацію, аналіз, модифікацію та відтворення, що дозволяє вирішувати різноманітні прикладні задачі.

Сучасні методи обробки аудіо знаходять застосування в численних галузях, таких як медицина, автомобільна промисловість, фінансовий сектор, агротехнології, розважальна індустрія та багато інших. Наприклад, у медицині аудіолейблінг використовується для аналізу звуків дихальних звуків чи серцевих тонів [6]. У сфері розваг технології обробки звуку застосовуються для аналізу музичних аудіосигналів: виявлення тональних, темпоральних, тембральних та інтенсивних характеристик [7]. Завдяки розвитку апаратного забезпечення та програмних засобів, обробка звукових файлів стала доступною не лише для спеціалізованих лабораторій, але й для широкого кола користувачів.

Одним із фундаментальних напрямів у сфері звукових технологій є цифрова обробка сигналів (Digital Signal Processing, DSP). Вона включає широкий спектр алгоритмів та методик, які дозволяють покращувати якість звуку, змінювати його характеристики, а також витягувати з нього корисну інформацію. До базових завдань DSP належать фільтрація шумів, регулювання гучності, нормалізація амплітуди, еквалізація, а також аналіз спектрального складу сигналу. Такі підходи 7 активно застосовуються у професійному та побутовому програмному забезпеченні для редагування аудіо – від простих редакторів до складних систем машинного навчання [8].

Особливу роль у сучасному аудіоаналізі відіграють методи глибинного навчання. Завдяки розвитку нейронних мереж, обробка звукових сигналів досягла нового рівня: стало можливим автоматичне розпізнавання мовлення, класифікація типів звуків, виявлення інструментів або емоцій, а також інтерпретація складних аудіосцен. Серед найпоширеніших підходів – аналіз спектрограм, які слугують графічною інтерпретацією частотно-часових характеристик сигналу та є основою для багатьох алгоритмів розпізнавання [9].

Попри ці досягнення, значна частина програм для автоматичної обробки та розмітки звукових файлів все ще має низку недоліків. Основна проблема полягає в обмеженій універсальності: багато інструментів можуть здійснювати або лише транскрипцію мовлення, або тільки розмітку за звуковими категоріями. Також спостерігається недостатня точність автоматичного аналізу, особливо в умовах фонового шуму чи при обробці багатокомпонентних аудіосигналів, що потребує ручної перевірки та редагування результатів. Крім того, розробники рідко пропонують комплексні рішення, здатні обробляти різні типи звуків – мовлення, шум, музику чи вокал – в одному середовищі [4].

Таким чином, впровадження інноваційних методів штучного інтелекту в галузь обробки аудіосигналів відкриває нові можливості для ефективного та гнучкого аналізу аудіоданих. Проте існуючі рішення поки що не повністю задовольняють потреби користувачів у точності, масштабованості та функціональності.

Отже, розвиток технологій обробки звукових файлів сприяє вдосконаленню якості обробки аудіоінформації, підвищенню точності виконання завдань та автоматизації різних процесів, пов'язаних з звукових файлами. Це є важливим етапом для розвитку таких галузей, як розпізнавання мовлення, аудіоаналіз, музичне програмування та обробка мультимедійного контенту.

1.2 Порівняльний аналіз аналогів

У сфері обробки звукових файлів існує низка програмних рішень, які дозволяють виконувати ті чи інші функції, пов’язані з аналізом, редагуванням і фільтрацією аудіо. Проте більшість з них спеціалізується на окремих аспектах

обробки звуку – таких як шумозаглушення, редагування або транскрипція – і не забезпечують комплексної інтеграції всіх функцій в одному інтерфейсі. Для порівняльного аналізу було обрано сучасні платформи: Sonix, Descript, Descript, Speechmatics та Encord. Нижче наведено короткий огляд кожного з них та порівняння з розробленим застосунком SoundSifter.

Sonix – це автоматизована платформа для транскрипції аудіо- та відеофайлів, що використовує штучний інтелект для перетворення мовлення на текст. Вона підтримує понад 40 мов і надає користувачам можливість швидко отримати транскрипти з часовими мітками та ідентифікацією спікерів [10]. Інтерфейс програми дозволяє редагувати текст безпосередньо в браузері, що спрощує процес коригування та уточнення транскрипцій (див рисунок 1.1).

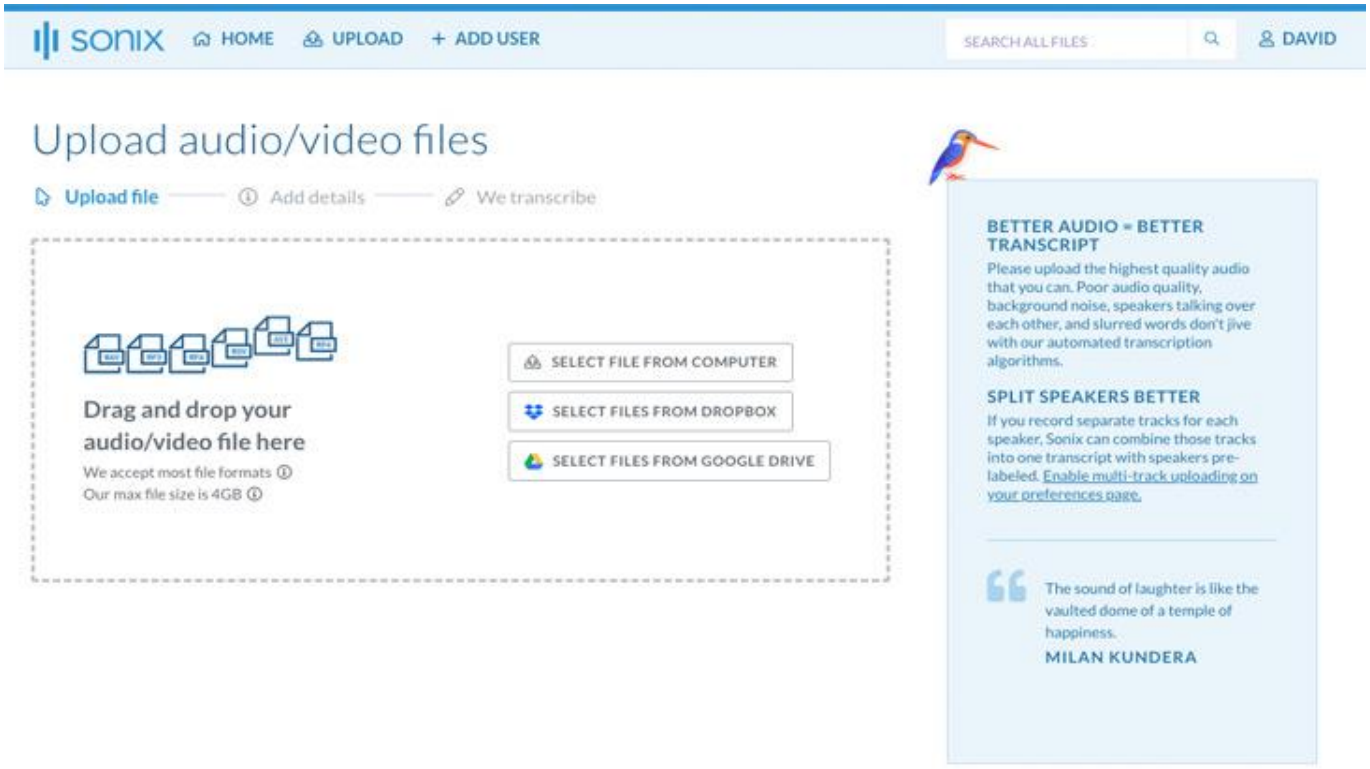

Рисунок 1.1 – Вікно платформи Sonix

Хоча платформа демонструє високу точність у багатьох випадках, деякі користувачі відзначають труднощі з розпізнаванням в умовах фонових шумів або при

наявності акцентів. Крім того, відсутність мобільних додатків та відносно висока вартість можуть бути обмеженнями для деяких користувачів [11].

Descript – це багатофункціональна платформа для редагування аудіо та відео, яка поєднує автоматичну транскрипцію з інтуїтивним текстовим інтерфейсом редагування. Користувачі можуть не лише отримувати транскрипти, а й безпосередньо редагувати їх, додаючи анотації та синхронізуючи текст з аудіо (див. рисунок 1.2). Такий підхід може бути корисним для швидкої обробки контенту, особливо в умовах обмеженого часу [12].

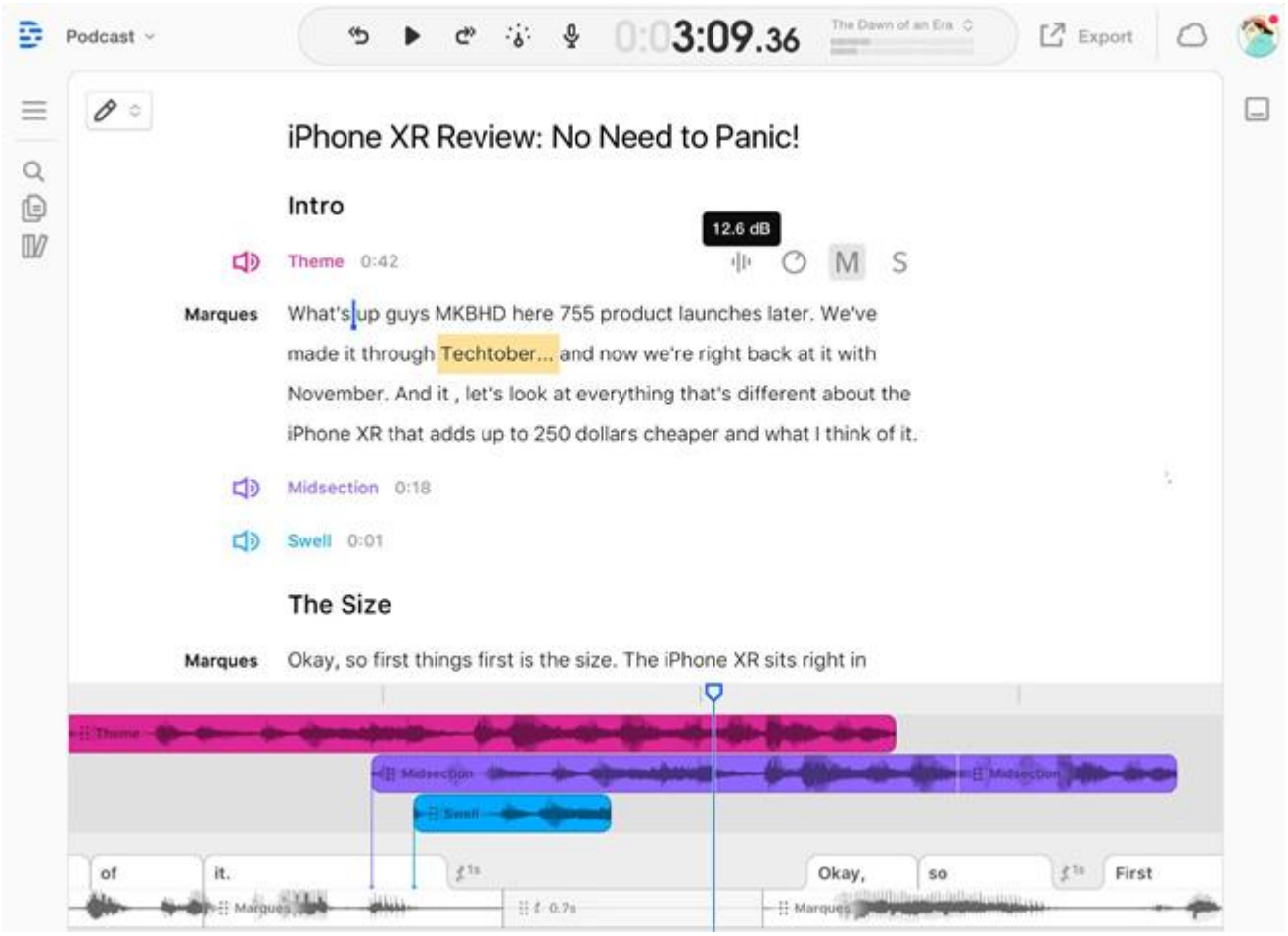

Рисунок 1.2 – Вікно застосунку Descript

Однією з особливостей Descript є можливість командної роботи: кілька користувачів можуть одночасно редагувати, коментувати та маркувати фрагменти, що може бути зручним для спільних проєктів. Однак варто зазначити, що деякі

користувачі відзначають труднощі з продуктивністю та стабільністю програми, особливо при роботі з великими проєктами або в умовах обмежених ресурсів.

Speechmatics – це платформа для автоматичного розпізнавання мовлення, яка використовує сучасні моделі машинного навчання для перетворення аудіо в текст. Вона підтримує понад 50 мов і спеціалізується на транскрипції в складних звукових умовах, таких як фоновий шум або різноманітні акценти. Серед особливостей Speechmatics підтримка ідентифікації мовлення кількох спікерів, автоматичне форматування чисел, дат і валют, а також можливість інтеграції через API для використання в реальному часі або з попередньо записаним аудіо [13]. На рисунку 1.3 наведено зображення інтерфейсу платформи.

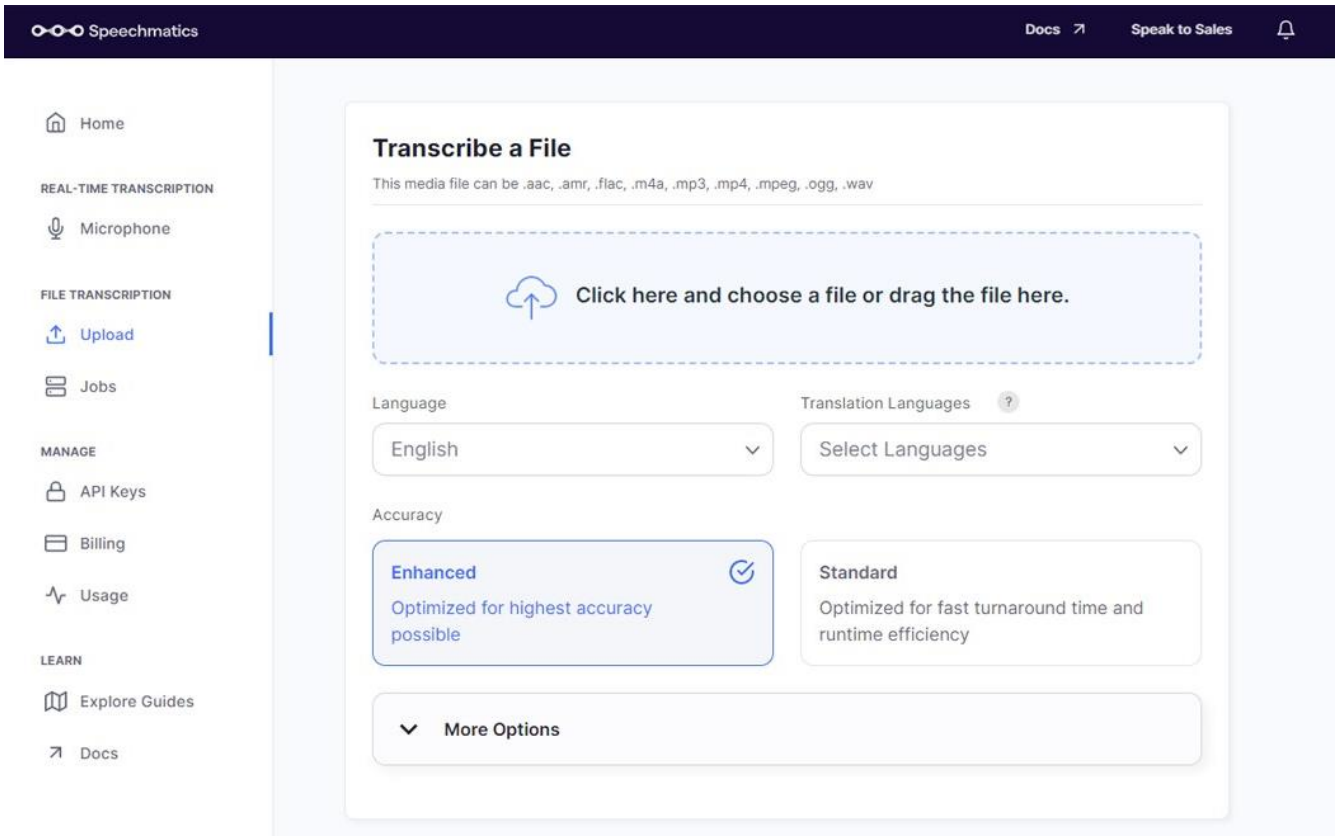

Рисунок 1.3 – Вікно платформи Speechmatics

Encord – це платформа для анотації мультимодальних даних, яка підтримує ефективне управління, маркування та підготовку великих неструктурованих наборів даних, включаючи аудіо, відео, зображення, тексти та документи. Вона надає

командам можливість виконувати різноманітні завдання з аудіоанотації, такі як розпізнавання мовлення, виявлення емоцій, класифікація звукових подій та аналіз звукових файлів.

Платформа підтримує мультимодальну анотацію, що дозволяє одночасно працювати з текстом, зображеннями та аудіо, що є корисним для складних проєктів у сфері штучного інтелекту. Encord також пропонує гнучкі інструменти для класифікації та точної часової прив'язки, включаючи підтримку багаторівневих анотацій та можливість відміток одночасних звукових подій або голосів [14]. На рисунку 1.4 зображено інтерфейс Encord, який дозволяє зручно редагувати анотації, відстежувати прогрес у реальному часі, фіксувати зміни та організовувати перевірку результатів.

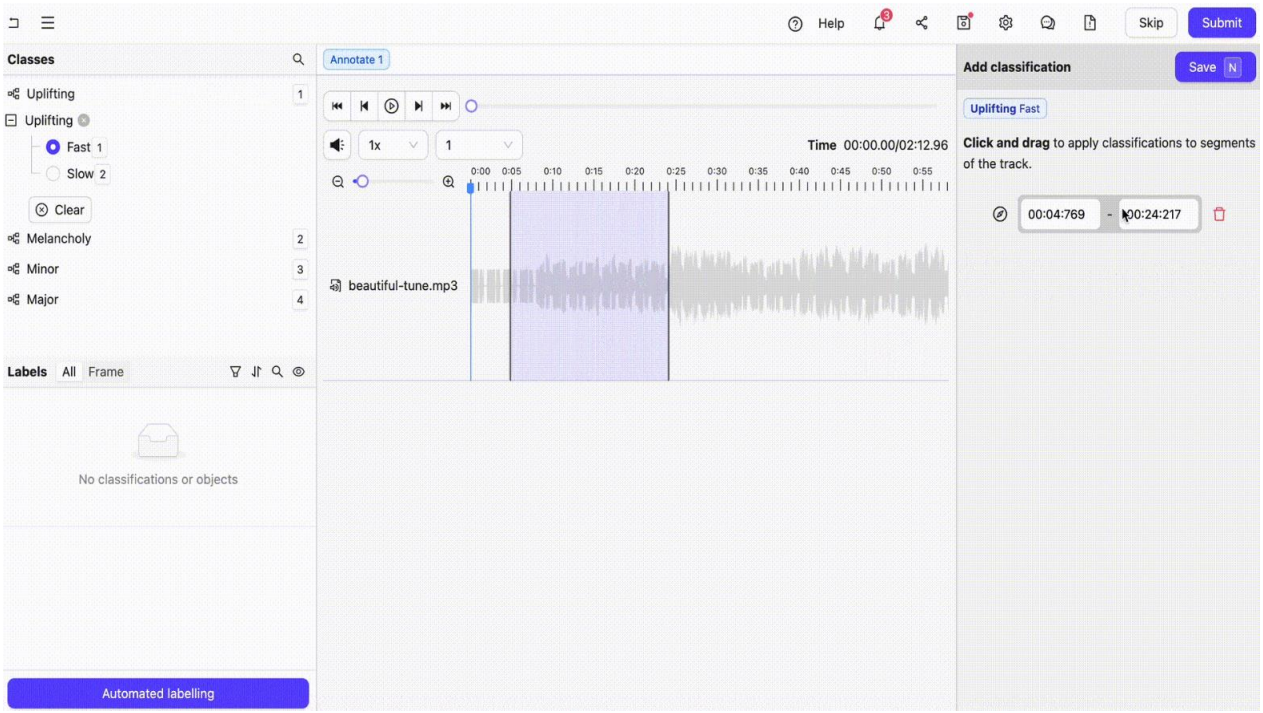

Рисунок 1.4 – Вікно інтерфейсу Encord

Завдяки інтеграції інструментів з елементами штучного інтелекту, таких як попереднє маркування чи контроль якості, процес створення анотацій стає значно швидшим і точнішим.

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	Sonix	Descript	Speechmatics	Encord	SoundSifter
Автоматична розмітка	+-	+-	+-	+	+
Транскрипція	+	+	+	+	+
Робота офлайн	-	+-	-	-	+
Точність транскрипції >85%	+	+	+	+	+
Точність класифікації типів звуків >90%	-	-	-	+	+
Високоякісне шумозаглушення	+-	+	+-	-	+
Сума балів	3	4	3	4	6
Загальна оцінка	50%	66%	50%	66%	100%

Відповідно до таблиці порівняльних характеристик, розробка власного програмного засобу для обробки звукових файлів є доцільною. Застосунок SoundSifter поєднує ключові функції, які раніше були розділені між кількома інструментами, та усуває їхні обмеження. Завдяки підтримці інтелектуального аналізу звуку, ефективного шумозаглушення, автоматичного видалення тиші, покращення якості аудіо та розпізнавання мовлення, користувач отримує універсальний інструмент для повної обробки мовних записів. Інтегрований графічний інтерфейс забезпечує зручність і наочність при взаємодії з файлами.

Отже, розробка програмного засобу SoundSifter відкриває нові можливості в обробці аудіо – як для подкастерів, журналістів, дослідників, так і для побутових користувачів. Поєднання сучасних алгоритмів штучного інтелекту та інтуїтивно зрозумілого GUI дозволяє ефективно перетворювати сирі звукові записи у якісний контент з мінімальними зусиллями. Це робить SoundSifter актуальним і конкурентоспроможним рішенням на тлі існуючих аналогів.

1.3 Аналіз методів розв’язання задачі

Розробка програмних модулів системи автоматизованої розмітки звукових файлів вимагає застосування сучасних методів та інструментів для ефективної обробки аудіоінформації. Для досягнення поставлених функціональних вимог необхідно визначити найбільш підходящі технології для аналізу, обробки та класифікації звуків. У цьому підрозділі буде проаналізовано основні підходи до розв'язання завдання автоматизованої розмітки звукових файлів та обґрунтовано вибір технологій і методів для реалізації кожної з ключових функцій.

Аналіз звукових файлів є першочерговою задачею, що полягає в розпізнаванні типів звуків (мовлення, тиша, музика тощо). Для ефективного аналізу та класифікації звуків використовуються проста модель Silero VAD та модель глибокого навчання YAMNet. Silero VAD – це попередньо навчена модель детектора голосової активності (VAD – Voice Activity Detection) корпоративного рівня, оптимізовану для роботи на одному потоці процесора [15]. YAMNet глибока модель, яка класифікує аудіо на понад 500 категорій, включаючи мовлення й музику, що робить її цінним інструментом для аудіоаналізу [16]. Використання цих інструментів дає більш точне визначення мовленнєвих фрагментів, навіть за наявності фонового шуму.

Шумозаглушення є важливою частиною обробки звукових файлів, особливо коли мають місце фонові шуми. У цій ситуації важливим є використання моделей, що використовують глибокі нейронні мережі для усунення небажаних шумів. Модель DeepFilterNet2, заснована на нейронних мережах, дозволяє ефективно приглушувати фонові шуми та покращувати якість мовлення. Її стабільність і результати роблять її оптимальним вибором для цього етапу обробки [17].

Подальше покращення якості аудіо здійснюється через обробку сигналів: нормалізацію, фільтрацію, еквалізацію. Для цього використовуються бібліотеки, що дозволяють працювати на рівні частотних характеристик (pydub, pyloudnorm), забезпечуючи чітке й природне звучання мовлення.

Перетворення мовлення в текст виконується за допомогою моделі Whisper від OpenAI. Вона показує відмінні результати навіть у складних умовах – при багатомовному мовленні або значному шумовому фоні [18].

Зважаючи на поставлені функціональні вимоги, GUI (графічний інтерфейс користувача) для програми буде розроблено з використанням бібліотеки customtkinter, яка дозволяє створювати сучасні, інтуїтивно зрозумілі інтерфейси для користувачів. Вибір цієї технології обумовлений її зручністю, простотою використання та широкими можливостями для налаштування.

Отже, завдяки поєднанню цих технологій програма забезпечить надійне розпізнавання звукових файлів, ефективне видалення шумів та мовлення, а також зручний інтерфейс для користувачів. Вибір зазначених технологій обґрунтований їх високою точністю, стабільністю та широким застосуванням у сучасних системах обробки аудіо.

1.4 Постановка задач

На основі аналізу існуючих підходів до обробки звукових сигналів та вимог до розробки програмного забезпечення для автоматизованої розмітки звукових файлів було визначено наступні основні задачі:

1. Провести аналіз сучасних методів виявлення мовлення, шумозаглушення та транскрипції звукових файлів із використанням моделей штучного інтелекту (Silero VAD, YAMNet, DeepFilterNet2, Whisper);
2. Спроектувати десктопний застосунок для автоматизованої розмітки звукових файлів із модулями аналізу, обробки, шумозаглушення та транскрипції;
3. Реалізувати комбіновану систему виявлення мовлення на основі моделей Silero VAD і YAMNet для точного визначення меж мовних сегментів;
4. Інтегрувати модель DeepFilterNet2 для шумозаглушення звукових файлів перед транскрипцією для підвищення якості мовних сегментів;
5. Розробити модуль транскрипції мовлення на основі моделі Whisper;
6. Провести експериментальну перевірку ефективності попередньої обробки (VAD + YAMNet + шумозаглушення) шляхом порівняння показників WER і CER із базовими методами транскрипції без попередньої обробки;
7. Розробити графічний інтерфейс застосунку для автоматизованої розмітки та візуалізації звукових сегментів;

8. Виконати тестування системи на реальних звукових файлах різної якості та тривалості для оцінки працездатності, точності сегментації та транскрипції.

Ці задачі спрямовані на створення ефективного та зручного програмного забезпечення для обробки та аналізу звукових файлів з можливістю автоматичної розмітки звуків.

1.5 Висновки

У першому розділі бакалаврської роботи було проведено аналіз існуючих програмних засобів для обробки звукових файлів, зокрема для автоматизованої розмітки та обробки аудіосигналів. Розглянуті були різні методи обробки звукових файлів, включаючи виявлення мовлення, класифікацію типів аудіо та очищення від шумів. Було проаналізовано такі популярні програмні рішення для аналізу звуку та його обробки.

Здійснивши порівняння цих програмних рішень, було встановлено, що існує потреба в розробці нового програмного забезпечення, яке буде поєднувати зручний графічний інтерфейс із розширеною функціональністю, орієнтованою на автоматизовану розмітку звукових файлів.

З огляду на це, було визначено завдання для подальшої розробки програмного забезпечення, що включає реалізацію алгоритмів для виявлення мовлення, покращення якості звуку, а також створення модуля для транскрипції мовлення. Також поставлене завдання інтеграції методів очищення звукових файлів від шумів та покращення їх якості, що дозволить підвищити точність обробки даних.

Таким чином, результати проведеного аналізу підтверджують актуальність теми бакалаврської роботи та обґрунтовують необхідність розробки програмного забезпечення для автоматизованої розмітки звукових файлів. Це рішення дозволить користувачам здійснювати точну обробку звукових файлів, автоматизуючи процеси виявлення мовлення та покращення якості звуку, що є важливим для багатьох сфер, зокрема для наукових досліджень і медичних застосунків.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Формалізація вхідних даних системи

Програмний продукт SoundSifter для обробки звукових файлів працює з різними типами вхідних даних: звуковий файл, параметри обробки та користувацькі запити. Важливим етапом є аналіз та обробка цих даних для забезпечення точності та ефективності роботи системи.

Вхідні звукові файли є першою категорією даних, з якими працює система. Застосунок призначений для обробки звукових файлів формату .mp3 .wav .flac .ogg .m4a .aac .wma .aiff, .aif .opus, які у подальшому конвертуються до формату WAV з частотою дискретизації 16000 Гц, що є необхідним для моделей що використовуються. Звукові файли можуть включати різноманітні звукові елементи, такі як мовлення, музика, тиша або фонові шуми. Для їх аналізу застосовуються передові алгоритми, зокрема Silero VAD (Voice Activity Detection), який дозволяє точно виявляти сегменти мовлення серед інших звуків. Цей інструмент є важливим для подальшого процесу обробки звукових файлів, адже дозволяє чітко відокремити мовлення від інших елементів запису.

Далі, важливим компонентом є визначення параметрів обробки аудіо, які залежать від користувацьких налаштувань. Зокрема, застосунок надає можливість виконання попереднього шумозаглушення, покращення якості звуку через фільтрацію та нормалізацію. Для реалізації шумозаглушення використовується алгоритм DeepFilterNet2, який ефективно знижує вплив шумів, дозволяючи отримати чистий звук навіть в умовах високого фону. Цей алгоритм є важливим елементом при роботі з аудіо, що містить великий обсяг шумів.

Одним із головних завдань програми є автоматичне перетворення мовлення в текст. Для цього використовується Whisper, модель для розпізнавання мовлення від OpenAI, яка забезпечує високу точність та швидкість перетворення. Модель Whisper розпізнає мовлення навіть в умовах складних акустичних ситуацій, таких як фоновий шум або погана якість запису.

Залежно від запитів користувача, система повинна динамічно реагувати та виконувати потрібні операції: відтворення очищеного звукового файлу, трансформація мовлення у текст або візуалізація звукової доріжки за допомогою бібліотеки matplotlib. Ця бібліотека використовується для створення графічного відображення аудіо сигналу, що дозволяє користувачам переглядати структуру звукової доріжки та перевіряти ефективність обробки аудіо [19].

Таким чином, вхідні дані системи можуть бути класифіковані на кілька основних категорій, і правильне їх оброблення є критичним для успішного виконання поставлених завдань. Використання таких інструментів, як Silero VAD, DeepFilterNet2, Whisper та matplotlib, дозволяє забезпечити високий рівень точності та зручності під час взаємодії з користувачем, що в свою чергу сприяє досягненню необхідного рівня ефективності роботи програми.

2.2 Розробка моделі системи

Для кращого розуміння логіки роботи програмного забезпечення SoundSifter, створеного для аналізу та обробки звукових файлів, було побудовано UML-діаграму діяльності. Ця діаграма дає змогу простим і наочним способом зобразити послідовність дій, що відбуваються у межах системи, а також взаємодію її складових з користувачами на кожному з етапів. Такий підхід дозволяє краще проаналізувати логіку роботи програмних компонентів, виявити вузькі місця або потенційні проблеми у бізнес-процесах [20].

Діаграма діяльності – це інструмент, що фокусується на відображенні динамічного аспекту системи, тобто на тому, як саме виконуються дії у часі. Кожна така діаграма будується на основі активностей (дій), які зображуються у вигляді прямокутників, та з’єднуються стрілками, що показують порядок виконання. Крім цього, вона може містити умови переходу, гілки прийняття рішень, а також початкові та кінцеві точки процесу. Подібні діаграми є надзвичайно корисними не лише для розробників, а й для аналітиків, менеджерів і замовників, адже вони допомагають уніфіковано інтерпретувати логіку роботи системи.

Застосування UML (Unified Modeling Language) у проєктуванні є сучасною практикою, яка забезпечує формалізацію процесу створення програмного забезпечення. UML дозволяє описувати як структурні, так і поведінкові характеристики системи. У випадку моделювання діяльності особливо важливо враховувати, як саме користувачі взаємодіють із системою, як відбувається обробка даних, і які сценарії можуть реалізовуватись у реальному середовищі [21].

Використання діаграм діяльності стає фундаментальним кроком при формуванні технічної документації та забезпечує єдине розуміння між усіма учасниками розробки – від бізнес-аналітиків до програмістів. Це також дозволяє ефективніше керувати змінами, тестуванням та подальшою підтримкою продукту.

Процес використання програми SoundSifter починається з того, що користувач, через інтерфейс програми, обирає один звуковий файл у підтримуваному форматі. Якщо обрано декілька файлів, система повідомляє користувача про те що потрібно обрати лише один звуковий файл. Якщо обраний файл має невідповідний формат, система видає повідомлення про відповідну помилку.

Після успішного завантаження звукового файлу система динамічно змінює інтерфейс змінюючи стан, тож далі користувач має обрати дії які він хоче виконати з звуковим файлом: нічого не робити, виконати аналіз – розмітку звуків, транскрибування і тд. Виконання дії транскрибування доступне лише разом з дією аналіз – розмітки звукового файлу, так як ці дії безпосередньо пов’язані. Також якщо обрано дію транскрибування то користувачу відкриється додаткове вікно з налаштуванням параметрів цієї дії. Після обраних дій й натискання на кнопку «Старт» відкриється вікно з динамічним текстом, який вказує на виконання поточного процесу, й також прогрес баром. Коли всі процеси виконаються вікно процесів закриється й система динамічно змінить інтерфейс головного вікна у фінальну форму, змінюючи стан.

У фінальному інтерфейсі на основі результатів аналізу звуків звукового файлу користувач може переглядати виділені сегменти звуків на звуковій доріжці. Також переглядати результати у вигляді діаграм та текстової опису розмітки звукового файлу. У цьому вікні користувач може також виконати деякі функції такі як

видалення усього, окрім мовлення, видалення тиші та інші, налаштувати їх параметри та виконати.

Користувач також має можливість експортувати розмітку звуків звукового файлу у зручному форматі для подальшого аналізу. Усі етапи розробки моделі були зосереджені на забезпеченні інтуїтивно зрозумілого інтерфейсу та ефективної взаємодії з користувачем.

UML-діаграму діяльності роботи системи зображено на рисунку 2.1.

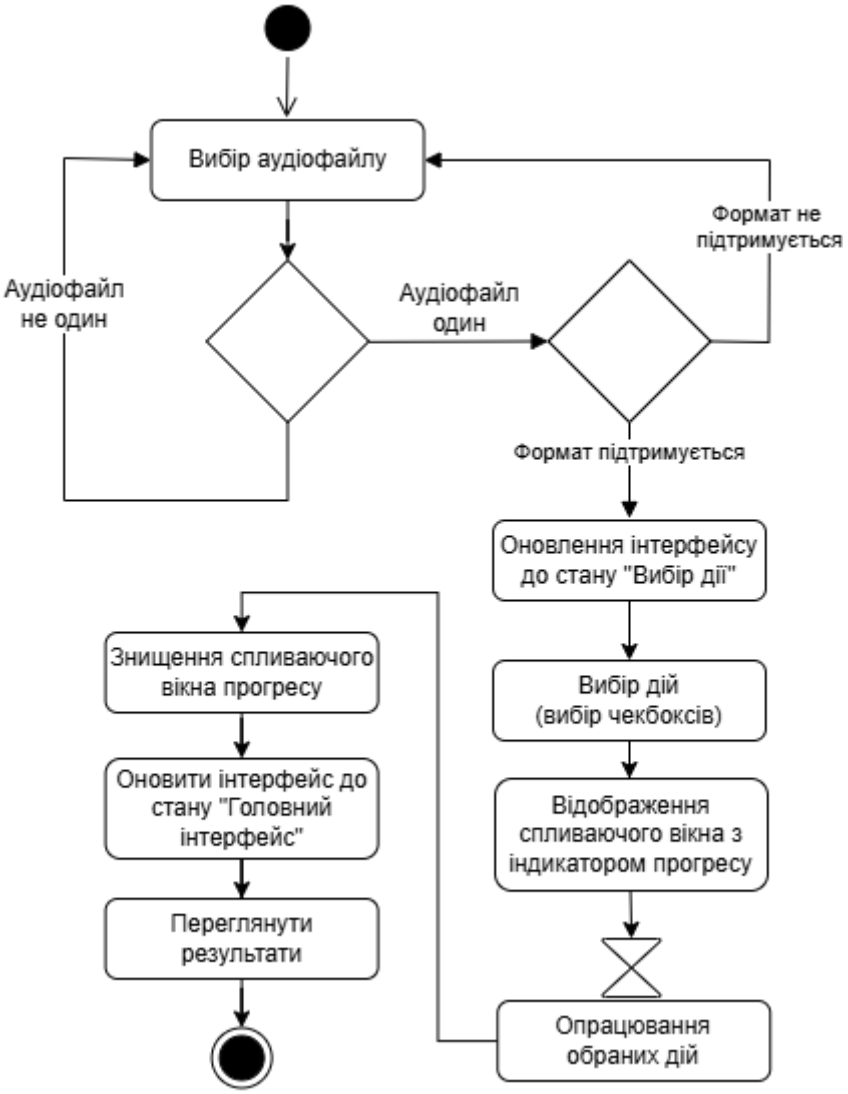

Рисунок 2.1 – Модель роботи системи у вигляді діаграми діяльності

Розробка діаграми діяльності проводилася в онлайн-середовищі app.diagrams.net.

2.3 Розробка алгоритмів роботи системи

У процесі створення програмного забезпечення для автоматичної розмітки звукових файлів важливим етапом є побудова логічно послідовних та структурованих алгоритмів, які визначають основну функціональність системи. Алгоритми мають охоплювати етапи обробки даних – від початкового імпорту звукового файлу, до генерації результату у вигляді структурованої розмітки, субтитрів, знешумленого або покращеного звукового файлу. Вони також повинні враховувати виняткові ситуації та варіанти нестандартної поведінки користувача, щоб забезпечити надійну роботу навіть у непередбачуваних умовах.

Одним з найбільш ефективних засобів для візуального представлення логіки таких алгоритмів є блок-схеми. Цей підхід дозволяє будувати послідовну структуру програмних дій за допомогою графічних елементів: прямокутників, ромбів та стрілок. Такі схеми суттєво полегшують процес розуміння і верифікації алгоритму ще до його програмної реалізації. Саме завдяки блок-схемам можливо на ранніх етапах розробки виявити неузгодженості або помилки в логіці, що сприяє зменшенню витрат часу на налагодження системи в подальшому [22].

Ключовою функцією, реалізованих у програмному забезпеченні SoundSifter, є аналіз – розмітка звуків звукового файлу, метод classify_audio(). Цей метод виконує комплексну обробку аудіоданих, забезпечуючи автоматичну класифікацію звукових сегментів на чотири основні категорії: мовлення, спів, музика та тиша. Його робота побудована на гармонійному поєднанні двох потужних моделей – Silero VAD для виявлення мовлення та YAMNet для класифікації звукових подій, що дозволяє досягти високої точності та гнучкості в обробці звукових файлів різного типу. Нижче детально описано алгоритм роботи цього методу, який є серцем системи SoundSifter, і його логіку, що забезпечує надійну та структуровану розмітку. На рисунку 2.2 зображено блок-схему алгоритму методу classify_audio().

Метод classify_audio() приймає шлях до звукового файлу та повертає чотири списки часових інтервалів, що відповідають мовленню, співу, музиці та тиші. Його робота складається з послідовних етапів, які забезпечують точну класифікацію звукових даних.

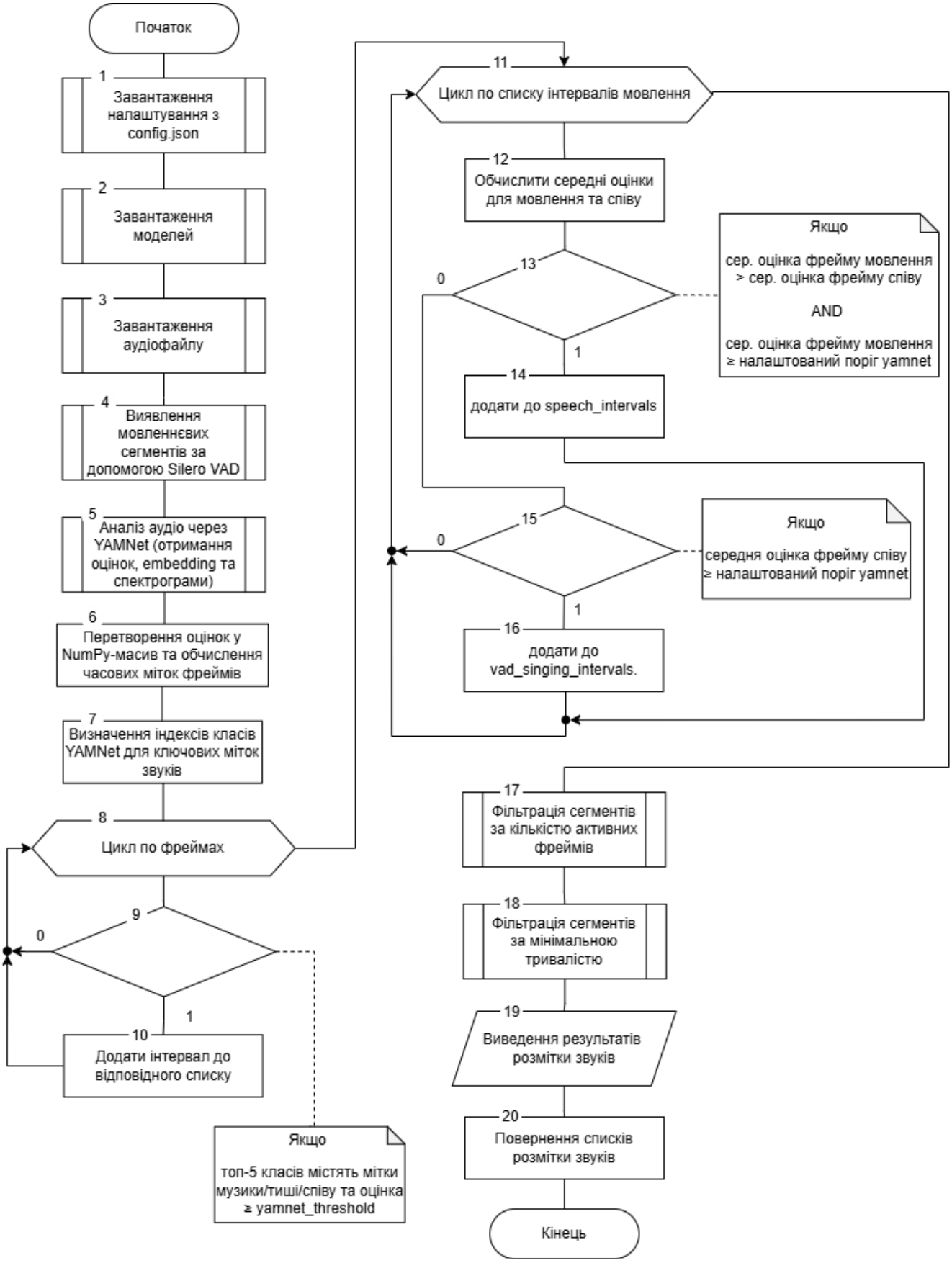

Рисунок 2.2 – Блок схема алгоритму методу classify_audio()

Спочатку звуковий файл завантажується, конвертується до моноформату з частотою 16 кГц і нормалізується для стабільної обробки. Далі спеціалізована модель виявляє сегменти мовлення, об’єднуючи їх за параметром максимальної паузи між звуковими фреймами (за замовчуванням 6 секунд). Паралельно інша модель аналізує аудіо, генеруючи оцінки для різних класів звуків кожні 0.48 секунди, та ідентифікує сегменти музики, співу й тиші, якщо ймовірність перевищує встановлений поріг (0.01).

Сегменти мовлення додатково класифікуються як мовлення або спів шляхом порівняння середніх оцінок ймовірності. Усі сегменти об’єднуються, якщо пауза між ними менша за заданий максимум, і фільтруються: за мінімальною кількістю активних фреймів (за замовчуванням 4) та мінімальною тривалістю (1.5 секунди), щоб усунути шуми. Для налагодження передбачено аналіз оцінок для окремих ділянок аудіо, а результати виводяться у зрозумілому форматі.

На завершення метод повертає списки часових інтервалів для мовлення, співу, музики та тиші, які можуть використовуватися для експорту, подальшого транскрибування чи іншої обробки аудіо.

Однією з важливих функцій, реалізованих у програмному забезпеченні SoundSifter, є метод reduce_audio_noise_deepfilternet(). Ця функція відповідає за очищення звукових файлів від шумів за допомогою моделі DeepFilterNet2, аналіз рівня шуму до та після обробки, а також збереження результатів у зручному форматі. Таким чином, вона відіграє центральну роль у модулі шумозаглушення, забезпечуючи якісну обробку аудіо та інформування користувача про зміни.

На рисунку 2.3 зображено блок-схему алгоритму даної функції.

Алгоритм роботи reduce_audio_noise_deepfilternet() розпочинається з аналізу рівня шуму вхідного звукового файлу за допомогою функції analyze_noise_level(), що дозволяє оцінити початковий стан аудіо. Далі виконується завантаження звукового файлу за допомогою бібліотеки torchaudio, після чого перевіряється частота дискретизації. Якщо вона відрізняється від цільової (48 кГц), аудіо автоматично конвертується до відповідного значення для сумісності з моделлю. У разі

невідповідності формату чи інших проблем система виводить повідомлення про помилку та завершує роботу.

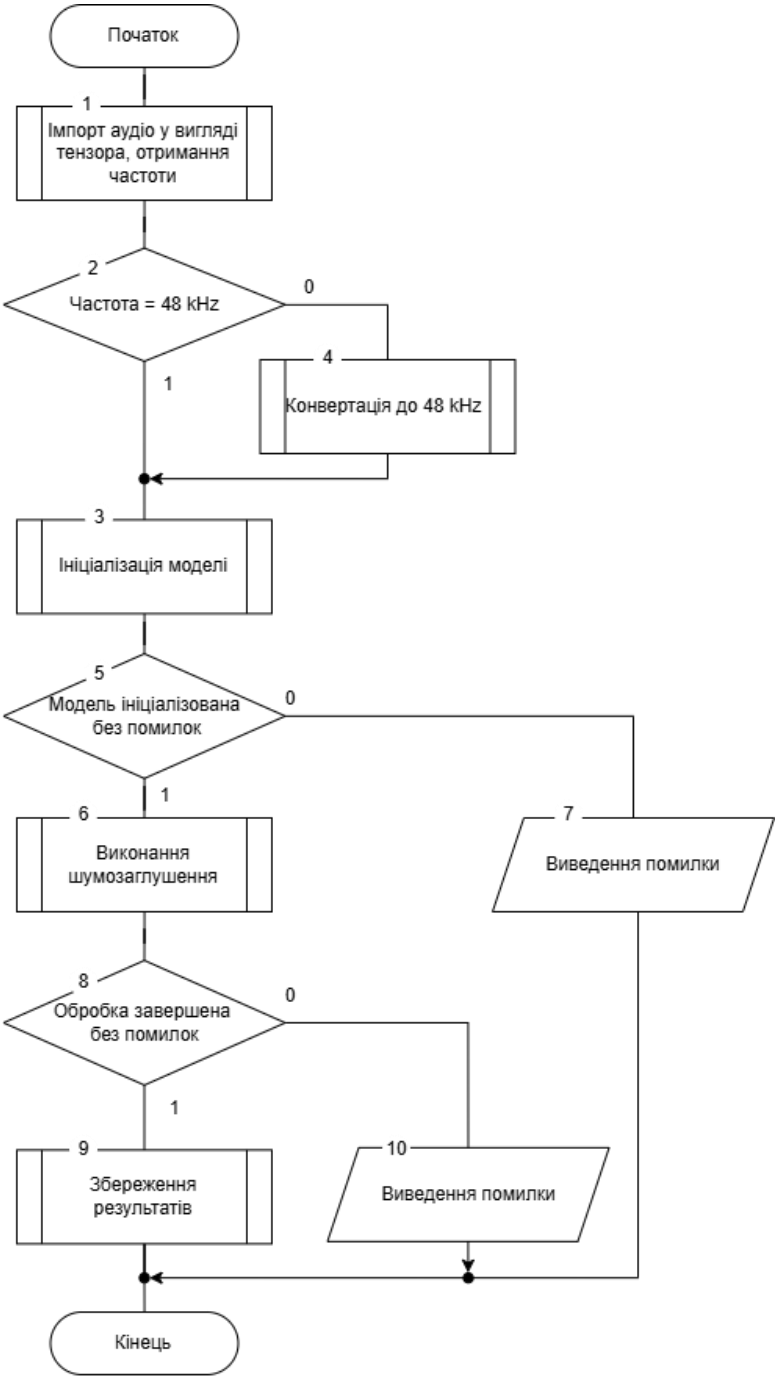

Рисунок 2.3 – Алгоритм роботи методу метод reduce_audio_noise_deepfilternet()

Якщо підготовка даних успішна, метод ініціалізує модель DeepFilterNet2 із включеним постфільтром через функцію init_df(). У разі виникнення помилок під час

ініціалізації (наприклад, через відсутність ресурсів або неправильну конфігурацію) функція повертає None і повідомляє про проблему. Після успішної ініціалізації модель застосовується для шумозаглушення вхідного звукового файлу, результатом чого є оновлений звуковий сигнал. Якщо обробка завершується з помилкою, система знову фіксує це та повертає None.

Потім за вказаним шляхом зберігається результат. Функція створює необхідні директорії, якщо їх ще немає, і зберігає оброблений звуковий файл за допомогою torchaudio.save().

Отже, для повнішого розуміння логіки роботи методу reduce_audio_noise_deepfilternet() була розроблена блок-схема, що відображає послідовність операцій, умови обробки помилок та етапи збереження результатів. Візуальне подання дозволяє легко простежити логіку переходів між етапами та зрозуміти структуру реалізації модуля шумозаглушення

2.5. Розробка архітектури системи

У межах цієї роботи запропоновано архітектуру десктопного програмного забезпечення для локальної обробки звукових файлів, яка поєднує каскадне виявлення мовлення, шумозаглушення та транскрипцію мовлення. Особливість системи полягає в інтеграції трьох ключових компонентів: каскадного виявлення мовлення за допомогою моделей Silero VAD і YAMNet, шумозаглушення з використанням DeepFilterNet2 та транскрипції мовлення через модель Whisper. Такий підхід дозволив зменшити кількість помилок розпізнавання мовлення (Word Error Rate) на 15–25% порівняно з транскрипцією необроблених звукових файлів, що є значним кроком вперед у порівнянні з традиційними методами.

Архітектура системи побудована за принципом каскадної обробки, де кожен модуль виконує свою функцію та передає результат наступному. Спочатку звуковий файл надходить до модуля аналізу – розмітки звуків, де Silero VAD визначає потенційні мовні сегменти, а YAMNet уточнює їх, відсіваючи спів, музику чи шум. Далі звуковий файл надходить до модуля шумозаглушення, де DeepFilterNet2 видаляє шум, покращуючи якість мовлення. На завершальному етапі у модуль

транскрибування надходить очищений звуковий файл та мітки мовленнєвих сегментів, й за допомогою Whisper відбувається аналіз й перетворення звуку мовлення, у текст, що забезпечує високу точність розпізнавання тексту.

Один із видів структурних діаграм в уніфікованій мові моделювання (UML), що показує, як компоненти системи розташовані та пов'язані між собою, називається діаграмою компонентів. Вони широко використовуються в проектуванні систем для сприяння модульності та полегшення для розробників процесу проектування, розуміння та передачі інформації про архітектуру [23].

На рисунку 2.4 зображено діаграму компонентів модулів аналізу та обробки системи.

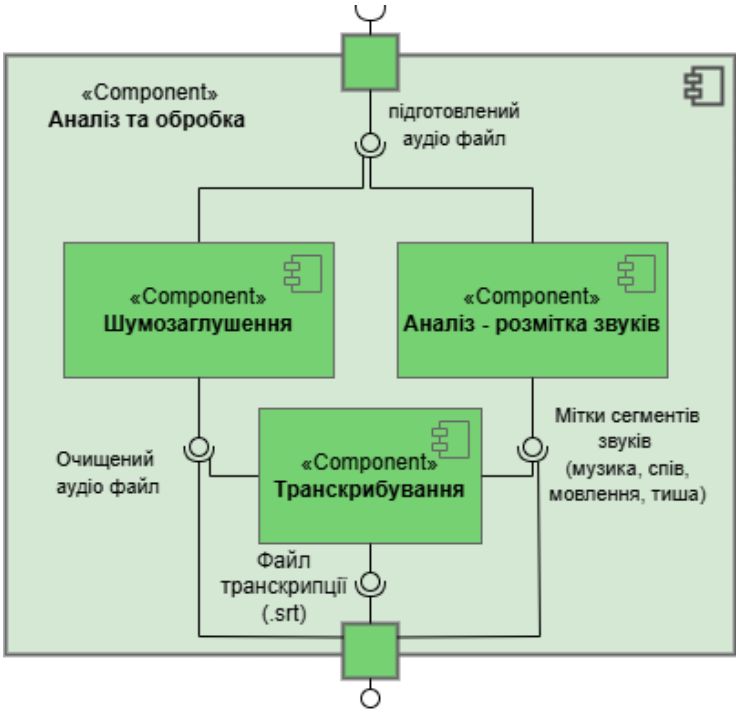

Рисунок 2.4 – Діаграма компонентів модулів аналізу та обробки системи

Новизна запропонованої архітектури полягає в її здатності ефективно обробляти змішані звукові файли (мовлення, шум, музика) у локальному середовищі, без необхідності підключення до хмарних сервісів. На відміну від аналогів, таких як Sonix чи Descript, які покладаються на онлайн-обробку, система гарантує конфіденційність даних і можливість роботи офлайн. Крім того, каскадна структура

зменшує обчислювальне навантаження, оскільки транскрипція застосовується лише до верифікованих і очищених мовних сегментів, що суттєво знижує кількість помилок.

Завдяки розробленій архітектурі вдалося створити повноцінну систему, яка є не лише надає локальне користування, ніж хмарні аналоги, але й демонструє вищу ефективність в обробці складних аудіо, зокрема змішаного типу (мовлення + фонові завади). Такий підхід особливо актуальний для роботи з різними записами, інтерв'ю, подкастами, відео блогами та іншим контентом, де чистота звуку може бути неідеальною.

2.6. Розробка комбінованого методу сегментації звукових даних

Другим ключовим компонентом системи є вдосконалений метод сегментації звукових даних, заснований на каскадному поєднанні двох нейронних моделей Silero VAD та YAMNet. На відміну від однокомпонентних рішень, які використовують лише один інструмент для виявлення мовлення, запропонований підхід дозволяє точніше виокремлювати мовні сегменти в звукових файлах зі змішаним вмістом – де мовлення чергується із співом, музикою, шумами чи фоновими звуками.

Комбінований метод сегментації звукових даних складається з таких етапів:

1. Попереднє виявлення активних звукових сегментів. Легка модель Silero VAD аналізує звуковий файл і визначає часові інтервали, де присутній звук, який може бути мовленням. Цей етап дозволяє швидко відсіяти великі ділянки тиші або шуму, зменшуючи обсяг даних для подальшої обробки.

2. Класифікація звукових фреймів. YAMNet аналізує весь звуковий файл, розділяючи його на фрейми тривалістю 0.48 секунди, і присвоює кожному фрейму ймовірності належності до класів, таких як мовлення, спів, музика чи тиша. Цей етап забезпечує детальну оцінку звукового вмісту, включаючи розрізнення співу без музичного супроводу.

3. Уточнення мовних сегментів. Для сегментів, виділених Silero VAD, проводиться порівняння оцінок YAMNet для класів мовлення та співу. Сегмент класифікується як мовлення, якщо середня оцінка для мовлення перевищує оцінку

для співу та заданий поріг. Це дозволяє відрізнити мелодійне мовлення від вокальних фрагментів.

4. Об’єднання та фільтрація сегментів. Сусідні сегменти об’єднуються, якщо пауза між ними не перевищує 6 секунд. Далі сегменти фільтруються за мінімальною кількістю активних фреймів (не менше 4) і тривалістю (не менше 1.5 секунди), що усуває короткочасні шуми чи помилкові детекції.

Тож Silero VAD забезпечує швидке попереднє виділення мовленнєвих ділянок, а YAMNet уточнює класифікацію, розрізняючи складні звукові текстури, такі як спів чи музика. Завдяки цьому досягається висока точність визначення міжмовних сегментів у змішаних звукових файлах.

У результаті, комбінований підхід дозволяє досягти точності сегментації понад 90% у реальних умовах – на змішаних звукових файлах з мовленням, шумом, музикою та співом. У порівнянні з базовими підходами, що використовують лише Silero або лише YAMNet, запропонована система демонструє суттєво вищу стійкість до помилок класифікації та дає чіткіші часові межі мовлення. Це безпосередньо впливає на ефективність транскрипції, оскільки дозволяє фокусуватися тільки на релевантних фрагментах аудіо, зменшуючи навантаження на ASR-модель і знижуючи загальну кількість помилок у результаті.

2.7 Висновки

У другому розділі було проведено аналіз вхідних даних та функціональних вимог програмного застосунку SoundSifter. Розглянуто основні типи даних, з якими працює система, а також дії користувача, які ініціюють різні етапи обробки. Було розроблену модель роботи системи за допомогою UML діаграми. Також було розроблено алгоритм методів аналізу – розмітки звуків та заглушення шуму. Крім того, розібрано архітектуру десктопного програмного забезпечення, що поєднує каскадне виявлення мовлення, локальне шумозаглушення та транскрипцію. Детально описано новий комбінований метод сегментації звукових даних, який дозволяє більш точно визначати межі мовлення навіть у складних звукових файлах зі змішаним вмістом.

3. РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ СИСТЕМИ АВТОМАТИЗОВАНОЇ ОБРОБКИ ЗВУКОВИХ ФАЙЛІВ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Для реалізації застосунку SoundSifter було обрано мову програмування Python, що обумовлено специфікою завдань, які вирішує система. Однією з ключових переваг цієї мови є її потужна екосистема бібліотек, орієнтованих на машинне навчання, обробку звуку та побудову графічного інтерфейсу. Ще одним практичним аспектом стала сумісність бібліотек та моделей – ключових компонентів SoundSifter, зокрема Silero VAD, Whisper, YAMNet та DeepFilterNet2, мають повноцінну підтримку саме в Python-середовищі. Таким чином, вибір мови був не лише доцільним, а й стратегічно обґрунтованим.

Для розробки середовищем було обрано Visual Studio Code (VS Code) – сучасний, легкий і розширюваний редактор, який забезпечує зручну роботу з Python-кодом, підтримку віртуальних середовищ, швидке налагодження, а також безліч корисних розширень для автодоповнення, інтеграції з Git, форматування коду тощо. Простота налаштування та кросплатформенність зробили VS Code оптимальним вибором як для розробки застосунку.

Для виявлення мовлення було використано бібліотеку Silero VAD. Її головна перевага підтримка багатьох мов, включно з українською, та стабільна робота навіть за наявності фонових шумів. Модель легка у використанні та не потребує складної конфігурації. Для класифікації звуків (музика, мовлення, шум тощо) застосовано модель YAMNet, яка навчана на базі TensorFlow. Її інтеграція допомагає попередньо аналізувати типи аудіосигналів й підвищує точність виявлення мовлення та виявлення інших типів звуків.

Для транскрипції мовлення у текст обрано Whisper відкриту нейромережеву модель, розроблену OpenAI, яка вважається однією з найкращих на ринку. Вона підтримує українську мову та ще 50 інших без потреби у додатковому донавчанні, забезпечуючи високу точність навіть у складних умовах запису.

Шумозаглушення реалізовано за допомогою DeepFilterNet2 сучасної нейромережі, що працює офлайн. Це рішення дозволяє зберігати приватність даних, уникати залежності від хмарних сервісів і підвищити стабільність роботи із великими файлами.

Для забезпечення максимальної зручності користувача застосунок SoundSifter має мати інтуїтивно зрозумілий інтерфейс, який дозволяє легко завантажувати звукові файли, налаштовувати параметри обробки та отримувати результати у зрозумілій формі. Усі ключові функції, мають бути доступні через кілька кліків, щоб програма була зрозуміла як для професійних, так і для початкових користувачів. Тому графічний інтерфейс створено з використанням customtkinter – розширення до класичного tkinter, яке дозволяє створювати сучасні, привабливі й адаптивні візуальні компоненти. Цей інструмент добре працює на всіх основних операційних системах (Windows, macOS, Linux), що, без додаткових залежностей, робить застосунок універсальним і доступним.

Для побудови візуальних елементів аудіоаналізу, зокрема спектрограм та графіків сигналу, використано бібліотеку matplotlib, яка надає користувачу візуальне уявлення про звук оброблюваного звукового файлу, що корисно для візуальної оцінки звукової хвилі, на якій розміщено виділення сегментів звуків.

Архітектура застосунку побудовано за принципом модульності: кожна функціональна частина – окремий блок, що полегшує тестування, оновлення або заміну окремих компонентів без шкоди для стабільності всього застосунку. Такий підхід також відкриває можливості для подальшого масштабування проєкту [24].

У підсумку, поєднання Python, зручного середовища розробки та перевірених бібліотек дозволило створити сучасний, точний, гнучкий та доступний інструмент для глибокої обробки мовлення в звукових файлах із фокусом на якість, швидкість і зручність для кінцевого користувача.

3.2 Розробка модуля виявлення типів звуків

Модуль виявлення типів звуків було розроблено для класифікації та сегментації звукових файлів на основі їхнього вмісту, зокрема розпізнавання

мовлення, співу, музики та тиші. Основна мета цього модуля – надавати точну ідентифікацію різних типів звукових сегментів у вхідному звуковому файлі, що є фундаментом для подальшої обробки в системі SoundSifter. Модуль дозволяє користувачу отримувати детальну інформацію про структуру аудіо, включаючи часові межі кожного типу звуку, що значно полегшує аналіз та обробку звукових даних.

Модуль реалізовано у файлі silero_vad_yamnet_classifier.py з використанням допоміжних функцій із utils.py. Головною функцією модуля є classify_audio(), яка координує весь процес класифікації. Вона виконує наступні кроки:

1. Викликає load_wav() для завантаження аудіо.
2. Використовує Silero VAD для виявлення сегментів, що потенційно містять мовлення, з параметром vad_max_gap=6.0, який визначає максимальну тривалість паузи між сегментами для їх об’єднання.
3. Застосовує YAMNet для аналізу всьому файлу, визначаючи ймовірності для різних класів звуків (мовлення, спів, музика, тиша) із заданим порогом yamnet_threshold=0.2.
4. Фільтрує та об’єднує сегменти з урахуванням мінімальної тривалості (min_duration=1.5) та мінімальної кількості активних фреймів (min_active_frames=4).

На рисунку 3.1 наведено фрагмент коду, що ілюструє початок роботи classify_audio().

```
63  def classify_audio(filepath):
64      wav, sr = load_wav(filepath)
65      waveform = wav.astype(np.float32) / np.abs(wav).max()
66      # 1. виявлення мовлення через Silero VAD
67      vad_segments = get_speech_timestamps(
68          torch.from_numpy(wav),
69          model,
70          sampling_rate=sr,
71          min_silence_duration_ms=int(vad_max_gap * 1000)
72      )
73      vad_intervals = [(seg['start'] / sr, seg['end'] / sr) for seg in vad_segments]
```

Рисунок 3.1 – Початок класифікації аудіо з використанням Silero VAD

Метод класифікації звуків у модулі базується на комбінованому підході, який поєднує сильні сторони моделей Silero VAD та YAMNet для точного визначення типів звуків. Спочатку Silero VAD аналізує звуковий файл і визначає часові межі сегментів, що потенційно містять мовлення, шляхом виявлення активності голосу. Ці межі зберігаються у вигляді списку інтервалів для подальшого аналізу. Потім YAMNet обробляє весь звуковий файл, розбиваючи його на фрейми тривалістю 0.48 секунди, і для кожного фрейму визначає ймовірності належності до різних класів звуків, таких як мовлення, спів, музика та тиша. YAMNet добре справляється з ідентифікацією музики, тому її результати для музичних сегментів вважаються пріоритетними і не піддаються додатковій корекції.

Для сегментів, які Silero VAD визначив як мовлення, проводиться додатковий аналіз за допомогою YAMNet. У кожному такому сегменті обчислюються середні оцінки для категорій «Speech» (з використанням ярликів speech_labels, таких як «speech», «conversation», «narration») та «Singing» (з ярликами singing_labels, наприклад, «singing», «lullaby», «a capella»). Якщо середня оцінка для «Speech» перевищує оцінку для «Singing» і досягає порогу yamnet_threshold=0.2, сегмент класифікується як мовлення. Наприклад, якщо у фреймах сегмента оцінки для «Speech» становлять 0.894, 0.975, 0.994, а для «Singing» — 0.000, 0.001, 0.000, то середня оцінка для «Speech» значно вища, і сегмент однозначно визначається як мовлення. У протилежному випадку, якщо оцінка «Singing» перевищує «Speech» і також досягає порогу, сегмент класифікується як спів.

Паралельно YAMNet аналізує весь файл для виявлення співу незалежно від результатів Silero VAD, оскільки спів може бути присутнім і без музичного супроводу. Для цього перевіряється, чи ярлики з singing_labels входять до топ-5 класів для кожного фрейму, і якщо оцінка для «Singing» перевищує поріг, ці фрейми додаються до сегментів співу. Наприклад, у фреймах із оцінками «singing: 0.237», «lullaby: 0.105», «a capella: 0.157» видно, що спів присутній, і ці фрейми додаються до відповідних меж. Для уникнення фрагментації сегментів використовується функція merge_segments() із utils.py, яка об’єднує сегменти, якщо між ними пауза менша за 6 секунд (див. рисунок 3.2)

```python
def merge_segments(segments, max_gap=6.0):
    """Об'єднує сегменти, які йдуть підряд
    або розділені паузою до max_gap сек."""
    if not segments:
        return []

    segments = sorted(segments, key=lambda x: x[0])
    merged = [segments[0]]
    for start, end in segments[1:]:
        last_start, last_end = merged[-1]
        if start - last_end <= max_gap:
            merged[-1] = (last_start, max(end, last_end))
        else:
            merged.append((start, end))
    return merged
```

Рисунок 3.2 – Функція об'єднання сегментів

Музичні сегменти, визначені YAMNet, також аналізуються окремо. Якщо у фреймі присутні ярлики з music_labels, такі як «music», «guitar», «piano», і оцінка перевищує поріг, цей фрейм додається до музичних сегментів. Наприклад, фрейми з оцінками «music: 0.995», «new-age music: 0.770», «ambient music: 0.129» однозначно класифікуються як музика. Якщо у фреймах одночасно присутні ярлики з singing_labels, наприклад, «singing: 0.052», «lullaby: 0.220», то сегмент додається як до музичних, так і до співочих меж, що дозволяє коректно обробляти музику зі співом.

Для підвищення якості класифікації застосовується фільтрація. Функція filter_by_min_duration() із utils.py видаляє сегменти, коротші за задану мінімальну тривалість (1.5 секунди), що допомагає уникнути надмірної фрагментації (див. рисунок 3.3)

```python
def filter_by_min_duration(segments, min_duration=0.5):
    """Фільтрує сегменти, коротші за min_duration."""
    return [(start, end) for start, end in segments if (end - start) >= min_duration]
```

Рисунок 3.3 – Функція фільтрації за мінімальною тривалістю

Додатково модуль використовує функцію filter_by_min_active_frames(), яка перевіряє, чи достатня кількість фреймів у сегменті відповідає потрібному типу звуку, щоб уникнути хибних спрацьовувань. Наприклад, сегмент вважається валідним, якщо щонайменше 4 фрейми мають оцінки, що перевищують поріг для відповідного типу звуку.

Отже, метод класифікації звуків у модулі виявлення типів звуків дозволяє точно сегментувати звукові файли, враховуючи специфіку різних типів звуків. Поєднання Silero VAD для первинного виявлення мовлення та YAMNet для детальної класифікації, разом із механізмами фільтрації та об'єднання сегментів, забезпечує надійну основу для подальшої обробки в SoundSifter. Модуль підтримує гнучкість завдяки налаштуванням порогів і параметрів фільтрації, що дозволяє адаптувати його до різних аудіосценаріїв.

3.3 Розробка модуля для видалення шумів та тиші зі звукових файлів

Модуль для видалення шумів та тиші зі звукових файлів було розроблено для забезпечення якісної обробки аудіоданих у системі SoundSifter, дозволяючи користувачу видаляти непотрібні сегменти тиші та зберігати лише мовлення з можливістю додавання контрольованих пауз. Основна мета цього модуля – надати гнучкий інструмент для очищення звукових файлів від тиші з урахуванням різних критеріїв, а також ізолювати сегменти мовлення для подальшого використання. Модуль забезпечує інтуїтивно зрозумілу обробку аудіо, дозволяючи користувачу налаштовувати тривалість пауз і режими видалення тиші, що робить його універсальним для різних сценаріїв обробки звуку.

Модуль реалізовано у файлі remove_interval.py з використанням бібліотеки pydub для роботи зі звуковими файлами та допоміжних функцій із utils.py для форматування часу. Він включає дві основні функції: keep_only_speech() для ізоляції сегментів мовлення та remove_silence() для видалення тиші за заданими критеріями. Ці функції дозволяють точно обробляти звукові файли, зберігаючи лише потрібні сегменти, а також надають детальну інформацію про процес обробки через логи у консолі.

Модуль складається з двох ключових функцій, кожна з яких вирішує окрему задачу обробки аудіо. Перша функція, keep_only_speech(), призначена для створення нового звукового файлу, який містить лише сегменти мовлення, із можливістю додавання тиші на початку та між сегментами. Функція приймає вхідний файл, вихідну папку, список сегментів мовлення (у форматі кортежів із початковими та кінцевими часами), а також параметри initial_silence і inter_silence для контролю тривалості пауз. Вона завантажує звуковий файл за допомогою AudioSegment.from_file() і створює новий звуковий файл, додаючи спочатку тишу заданної тривалості (initial_silence), а потім послідовно сегменти мовлення, між якими вставляється тиша (inter_silence).

Функція завершує роботу, зберігаючи результат у вихідний файл, який визначається залежно від параметра overwrite. Якщо overwrite=True, оригінальний файл замінюється; інакше створюється новий файл із суфіксом _speech_only.wav у вказаній папці. Процес супроводжується логуванням, яке включає часові межі доданих сегментів і загальну тривалість результату, що полегшує відстеження обробки.

Друга функція, remove_silence(), призначена для видалення сегментів тиші зі звукового файлу за заданими критеріями. Вона підтримує чотири режими видалення: «all» (видалити всю тишу), «longer_than» (видалити проміжки, довші за поріг), «shorter_than» (видалити проміжки, коротші за поріг), і «selected» (видалити сегменти за вказаними індексами). Функція завантажує звуковий файл, сортує сегменти тиші за початковим часом і фільтрує їх відповідно до обраного режиму. На рисунку 3.4 наведено приклад коду, що показує вибір сегментів для видалення.

```
76    intervals_to_remove = []
77    if mode == "all":
78        intervals_to_remove = silence_intervals
79    elif mode == "longer_than" and threshold is not None:
80        intervals_to_remove = [(start, end) for start, end in silence_intervals if (end - start) >= threshold]
81    elif mode == "shorter_than" and threshold is not None:
82        intervals_to_remove = [(start, end) for start, end in silence_intervals if (end - start) <= threshold]
83    elif mode == "selected" and selected_indices is not None:
84        intervals_to_remove = [silence_intervals[i] for i in selected_indices if i < len(silence_intervals)]
```

Рисунок 3.4 – Логіка вибору сегментів тиші для видалення

Якщо жоден сегмент не відповідає критеріям, функція повертає оригінальний файл без змін, про що користувач отримує відповідне повідомлення. У протилежному випадку створюється новий звуковий файл, який складається з сегментів між проміжками тиші. Процес видалення реалізується шляхом ітерації по відфільтрованих сегментах тиші, додавання проміжків між ними до результуючого аудіо та збереження залишку після останнього сегмента тиші.

Розробка модуля для видалення шумів та тиші була спрямована на створення гнучкого та зручного інструменту для обробки звукових файлів. Функції keep_only_speech() і remove_silence() забезпечують точне видалення тиші та ізоляцію мовлення з можливістю тонкого налаштування параметрів, таких як тривалість пауз і режими видалення. Інтеграція з бібліотекою pydub та детальне логування роблять модуль ефективним і зручним для користувача, а його структура дозволяє легко інтегрувати його з іншими компонентами SoundSifter.

3.4 Розробка модуля для транскрипції мовлення в текстовий формат

Модуль для транскрипції мовлення в текстовий формат було розроблено для автоматичного розпізнавання мовлення з звукових файлів та перетворення його в текст із можливістю збереження результатів у форматі SRT для подальшого використання, наприклад, у створенні субтитрів. Основна мета цього модуля – забезпечити точну транскрипцію мовлення в заданих сегментах звукового файлу, з підтримкою додаткових функцій, таких як шумозаглушення та покращення якості звуку, що підвищує якість розпізнавання. Модуль надає користувачу повний текст розпізнаного мовлення у консолі та зберігає його у структурованому форматі SRT, що робить його зручним для використання у різних сценаріях, таких як створення субтитрів для відео чи документування аудіозаписів.

Модуль реалізовано у файлі transcribe.py, який інтегрує бібліотеку whisper для розпізнавання мовлення, pydub для обробки звукових файлів, а також функції з інших модулів, таких як reduce_audio_noise_deepfilternet() для шумозаглушення та enhance_speech() для покращення звуку. Він складається з трьох основних функцій: transcribe_audio() для координації процесу транскрипції, create_srt_file() для

створення SRT-файлу та format_srt_time() для форматування таймкодів. Модуль забезпечує гнучкість завдяки параметрам, які дозволяють налаштувати мову розпізнавання, застосовувати попередню обробку аудіо та додавати контекстний промпт для підвищення точності.

Модуль включає кілька ключових функцій, кожна з яких виконує певну роль у процесі транскрипції. Функція format_srt_time() відповідає за форматування часу у форматі SRT (HH:MM:SS,mmm), який використовується для створення субтитрів. Вона перетворює час у секундах на рядок із годинами, хвилинами, секундами та мілісекундами, що забезпечує коректне відображення таймкодів у SRT-файлі.

Функція create_srt_file() створює SRT-файл на основі списку сегментів тексту з відповідними таймкодами. Вона записує кожен сегмент у форматі SRT, який включає номер сегмента, часові межі у форматі «початок --> кінець» та сам текст. Функція забезпечує коректну кодування UTF-8 для підтримки різних мов, включаючи українську, і додає порожній рядок між сегментами для відповідності стандарту SRT.

Головною функцією модуля є transcribe_audio(), яка координує весь процес транскрипції. Вона приймає вхідний звуковий файл, вихідну папку, список сегментів мовлення, а також параметри для активації шумозаглушення, покращення звуку, вибору мови та контекстного промпту. Функція спочатку перевіряє, чи потрібно застосовувати шумозаглушення (за допомогою reduce_audio_noise_deepfilternet()) або покращення звуку (за допомогою enhance_speech()), і створює тимчасові файли для обробленого аудіо. Далі завантажується модель Whisper (модель «medium» для оптимальної якості), після чого кожен сегмент мовлення обрізається з оригінального файлу за допомогою pydub і обробляється окремо.

Отже, розробка модуля для транскрипції мовлення в текстовий формат була спрямована на створення потужного інструменту для автоматичного розпізнавання мовлення з звукових файлів. Інтеграція моделі Whisper із функціями попередньої обробки, такими як шумозаглушення та покращення звуку, забезпечує високу точність транскрипції навіть у складних умовах. Збереження результатів у форматі SRT дозволяє використовувати текст для створення субтитрів або подальшого аналізу, а детальне логування робить процес прозорим і зручним для користувача.

Модуль гармонійно інтегрується з іншими компонентами SoundSifter, зокрема з модулем виявлення мовлення, що забезпечує цілісність системи.

3.5 Розробка інтерфейсу програмного застосунку

Розробка інтерфейсу програмного застосунку SoundSifter була зосереджена на створенні зручного, функціонального та візуально приємного середовища для користувача, яке забезпечує легкий доступ до всіх можливостей програми. Головною метою інтерфейсу є спрощення взаємодії користувача з функціоналом обробки звукових файлів, включаючи аналіз звуку, видалення тиші, шумозаглушення, покращення звуку та транскрипцію мовлення. Інтерфейс розроблено з використанням бібліотеки customtkinter, яка забезпечує сучасний вигляд і адаптивність елементів, а також темну тему для комфортної роботи в різних умовах освітлення.

На рисунку 3.5 зображено початковий екран програми відображається після запуску застосунку.

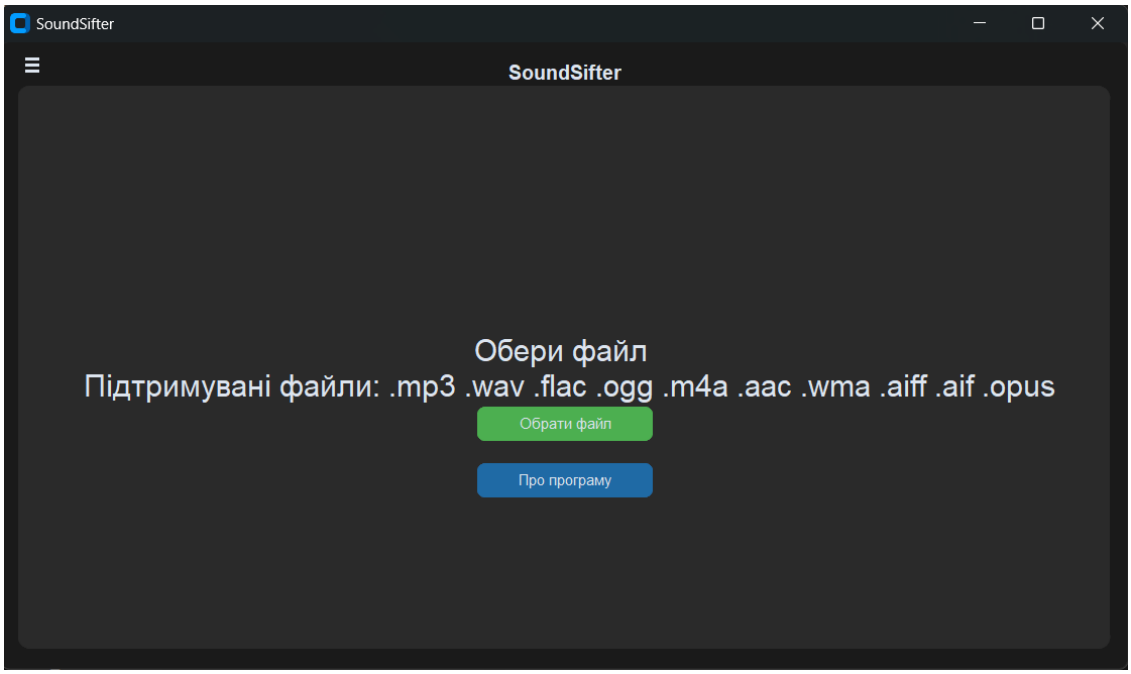

Рисунок 3.5 – Початковий екран застосунку SoundSifter

У верхній лівій частині вікна розташована кнопка бургер меню, яка відкриває шторку, у якій містяться основні опції, такі як вибір файлу, налаштування розмітки, вибір теми і тд (див рисунок 3.6).

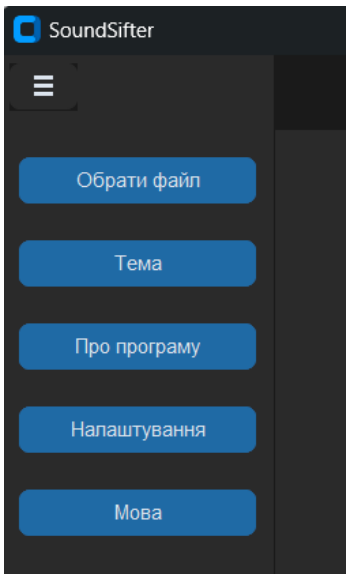

Рисунок 3.6 – Меню

Після обрання файлу відбудиться динамічна зміна вмісту інтерфейсу: у ньому з’являться чек бокси з обранням виконання дій й кнопка запуску (див. рисунок 3.7).

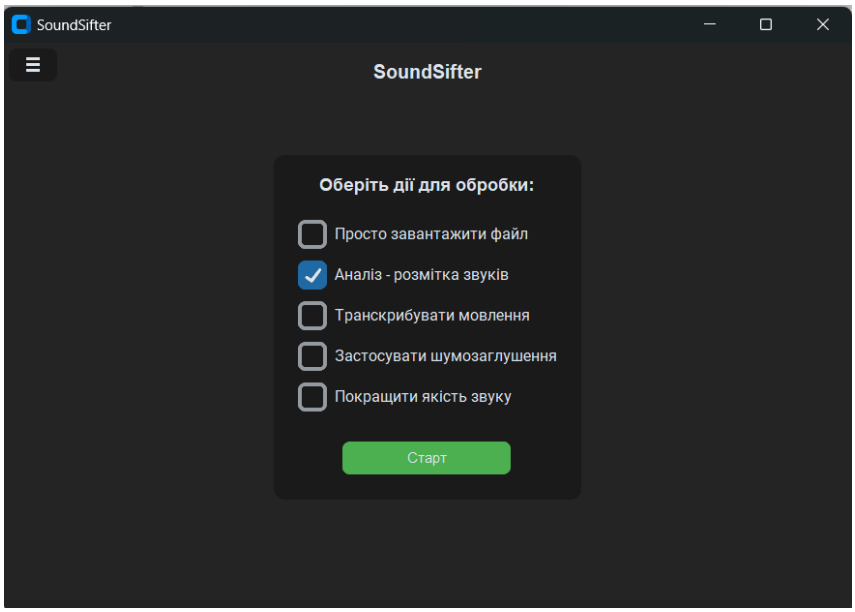

Рисунок 3.7 – Вміст вікна з вибором дій

Після обрання дій й натискання на кнопку «Старт» почнеться процес обробки дій, який буде відображено у вікні «Прогрес обробки» (див. рисунок 3.8).

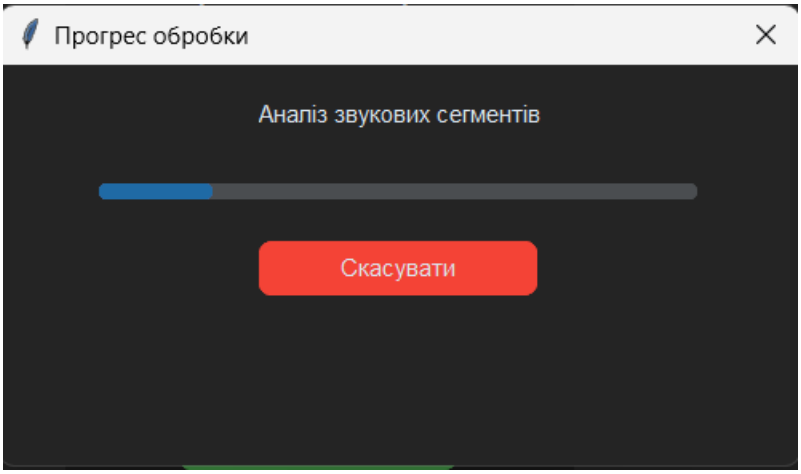

Рисунок 3.8 – Вікно «Прогрес обробки»

Коли процеси завершаться закриється вікно «Прогрес обробки», а у головному вікні відбудиться динамічна зміна вмісту інтерфейсу, інтерфейс набуде фінального вигляду (див. рисунок 3.9).

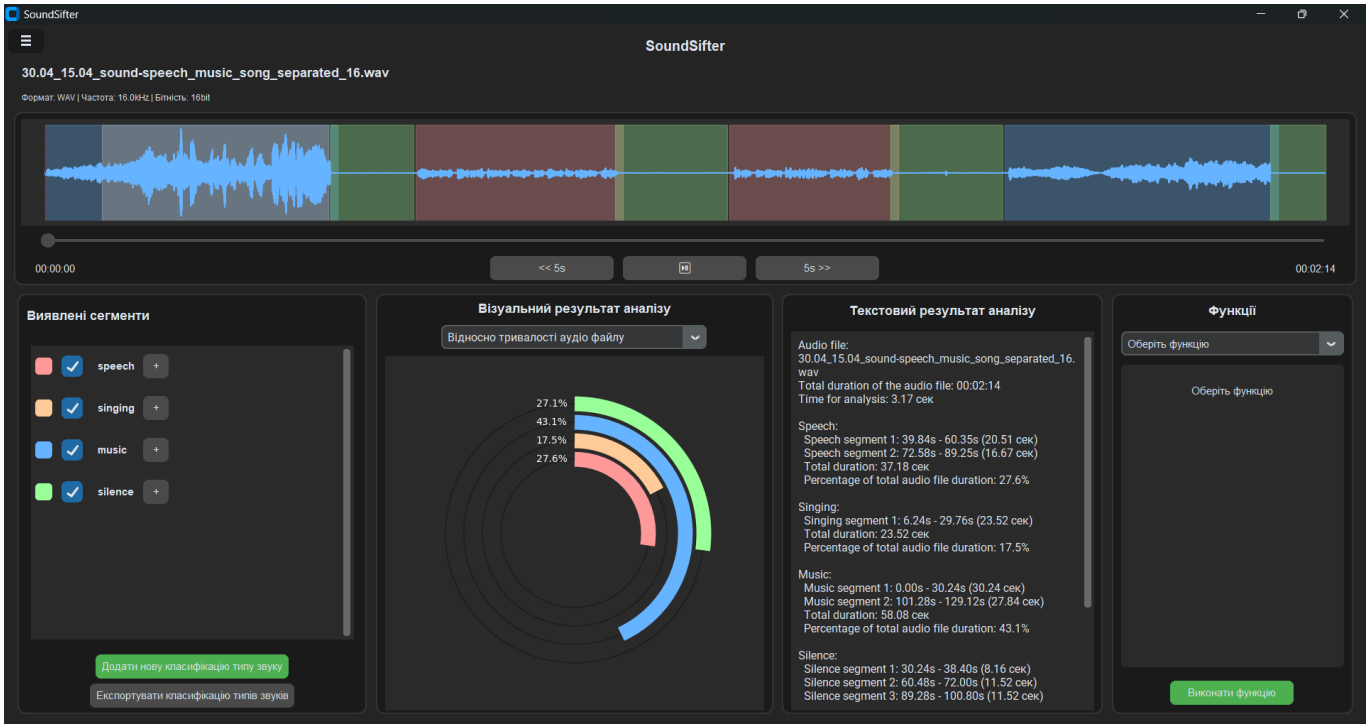

Рисунок 3.9 – Фінальне вікно

Верхній блок інтерфейсу включає область для відображення звукової доріжки, яка показує амплітуду аудіосигналу у вигляді графіка, а також дозволяє візуально виділяти сегменти різних типів звуку (мовлення, спів, музика, тиша) за допомогою кольорових зон. Під доріжкою розташовані елементи керування відтворенням: кнопки для відтворення/паузи, перемотування вперед і назад на 5 секунд, смужка прогресу з повзунком для індикації та перемотки відтворення. а також текстові поля, що показують поточну позицію та загальну тривалість аудіо.

Частина інтерфейсу під блоком 1 поділена на чотири блоки, що забезпечують чітку організацію елементів. Блок 2, який знаходиться у лівому краї вікна, містить розкладні меню, які користувач може розгорнути щоб побачити інформацію про сегменти відповідного типу звуку (див рисунок 3.10). Також наявні кнопки додавання нової класифікації типу звуку та кнопка експорту розмітки.

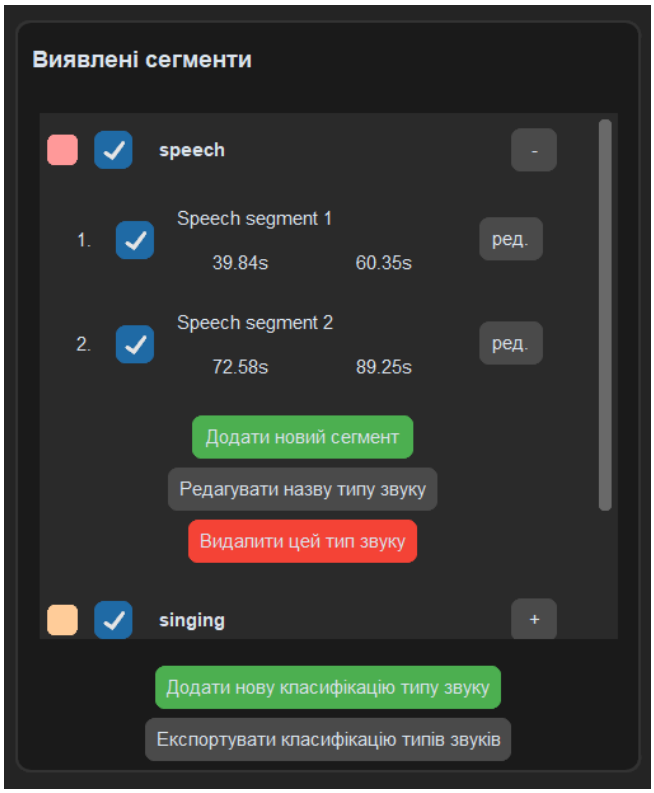

Рисунок 3.10 – Блок 2 з розкладеним меню

Блок 3 який знаходиться по праву сторону від блоку 2, містить діаграму з відсотковими залежностями типів звуків до хронометражу звукового файлу, є

можливість поглянути на іншу діаграму яка показує залежності типів звуків по відношенню один до одного. З права від блоку 3 йде блок 4 з текстовим результатом аналізу, у ньому у текстовому вигляді описана розмітка звуків звукового файлу. У останньому блоці 5 знаходиться випадаючий список функцій у якому можна обрати функції, та динамічний елемент який змінюється в залежності від обраних функцій й який містить у собі параметри обраної функції.

3.6 Висновки

У третьому розділі було обґрунтовано вибір технологій та інструментів, які використовувалися для розробки програмного застосунку SoundSifter, призначеного для аналізу та обробки звукових файлів.

У межах цього розділу детально описано розробку основних модулів програми, Кожен із модулів було реалізовано з акцентом на модульність і можливість інтеграції з іншими компонентами системи, що забезпечує її розширюваність та адаптивність до різних сценаріїв використання.

Крім цього, було розроблено та описано інтерфейс користувача, побудований на основі бібліотеки customtkinter, який забезпечує інтуїтивно зрозуміле середовище для роботи зі звуковими файлами.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Для забезпечення надійності та високої якості роботи програмного застосунку SoundSifter було впроваджено комплексний підхід до тестування, який поєднує модульне, функціональне та ручне тестування користувацького інтерфейсу. Такий підхід дозволяє виявляти помилки як на рівні окремих модулів, так і при інтеграції всіх компонентів системи, що є важливим для обробки звукових файлів із різними характеристиками.

Основним методом тестування стала функціональна перевірка, яка дозволила оцінити коректність виконання ключових завдань програми. Тести проводилися на різних звукових файлах із різною тривалістю та типами звуків, щоб переконатися, що система виконує свої функції відповідно до вимог користувача. Функціональні тести реалізовувалися вручну через графічний інтерфейс, що дозволило перевірити зручність використання та відповідність результатів очікуванням. Такий підхід відповідає сучасним практикам функціонального тестування аудіо- та відеозастосунків, де важливо забезпечити точність та надійність обробки медіа-даних [25].

Додатково проводилося модульне тестування окремих функцій, таких як classify_audio() для аналізу звуку, keep_only_speech() і remove_silence() для обробки сегментів, а також transcribe_audio() для транскрипції. Це тестування здійснювалося на ізольованих блоках коду з доданим логуванням та використанням тестових звукових файлів, що допомогло виявити потенційні помилки в логіці до інтеграції модулів у загальну систему [26]. Такий підхід до модульного тестування є ефективним для виявлення помилок на ранніх етапах розробки та забезпечення стабільності роботи окремих компонентів [27].

Також було проведено ручне тестування інтерфейсу користувача, спрямоване на оцінку зручності роботи з програмою та правильності візуалізації даних, таких як звукова доріжка, візуальний та текстовий результат аналізу. У процесі тестування перевірялися всі елементи, включаючи кнопки керування відтворенням, вибір файлів,

налаштування параметрів функцій та відображення сегментів на графіку. Ручне тестування інтерфейсу є важливим етапом у забезпеченні якості програмного забезпечення, особливо коли йдеться про взаємодію користувача з аудіо застосунками [28].

Обрані методи тестування забезпечили всебічну перевірку SoundSifter, підтвердивши його стабільність, ефективність і готовність до використання кінцевими користувачами. Цей підхід дозволив усунути виявлені недоліки та оптимізувати взаємодію між модулями, створивши надійний інструмент для обробки аудіоданих.

4.2 Результати тестування програмного забезпечення

У ході тестування програмного застосунку SoundSifter було перевірено коректність виконання всіх ключових функцій у різних умовах роботи. Ручне тестування проводилося на операційній системі Windows 11, що дозволило оцінити роботу програми в реальних умовах використання кінцевим користувачем.

Першою було протестовано функцію аналізу звукових файлів, яка визначає сегменти мовлення, співу, музики та тиші. Для тестування використовувалися звукові файли різної тривалості (від 30 секунд до 3 хвилин) із різними звуками. Аналіз проводився через графічний інтерфейс: після завантаження файлу та натискання кнопки «Аналіз - розмітка звуку» програма коректно визначала сегменти, які відображалися на звуковій доріжці, у блокові візуального та текстового результатів аналізу. Результати відповідали очікуваним: у тестовому файлі з чітким мовленням сегменти мовлення були визначені з точністю до 0.5 секунди.

Порівняння результатів автоматичної розмітки з ручною перевіркою показало високу точність роботи застосунку: точність автоматичної розмітки >95%.

Наступною перевірялася функція видалення всього, крім мовлення. Тестування проводилося з параметрами за замовчуванням (початкова тиша – 5 секунд, між сегментами – 10 секунд). Після виконання функції результуючий файл містив лише сегменти мовлення з доданими паузами, а музика, спів і тиша були успішно видалені. Загальна тривалість файлу відповідала сумі тривалостей мовленнєвих сегментів плюс

задані паузи. Жодних артефактів у звуці не виявлено. На рисунку 4.1 зображено файл до виконання функції видалення всього крім мовлення.

Рисунок 4.1 – Файл з мовленням, музикою та тишею

На рисунку 4.2 зображено файл який був отриманий після виконання функції.

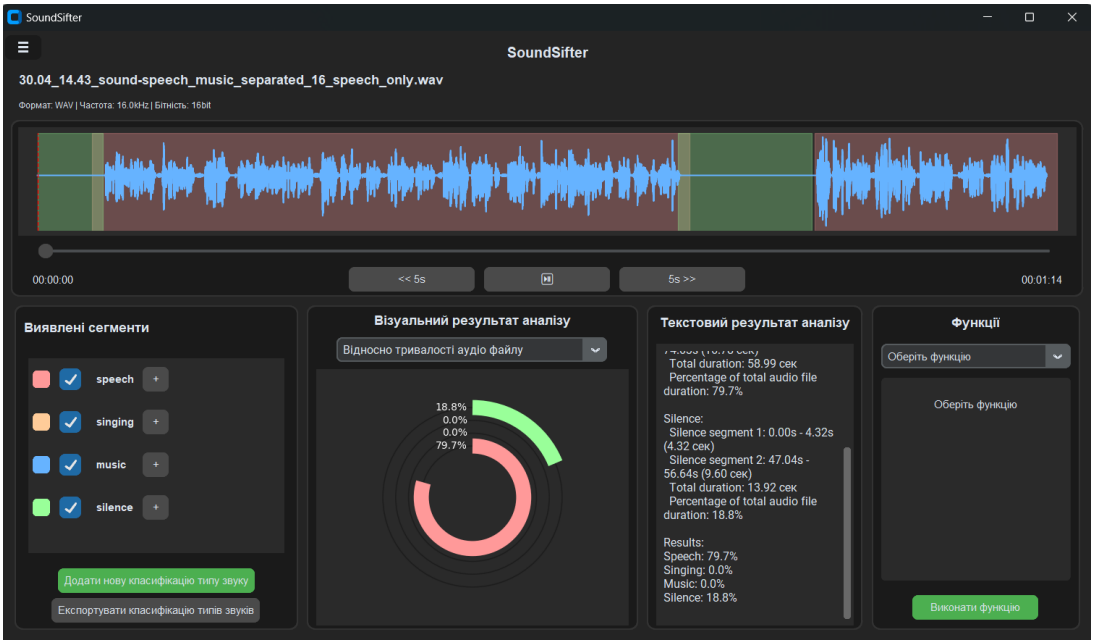

Рисунок 4.2 – Файл лише з мовленням

Функція видалення проміжків тиші також показала стабільну роботу. Використовуючи режим «all», програма видалила всі сегменти тиші, залишивши лише активні звукові фрагменти. У тестовому файлі тривалістю 01:14 з 13 секундами тиші результуючий файл мав тривалість 1 хвилина, що підтверджує коректність роботи функції. Результат видалення проміжків тиші зображено на рисунку 4.3.

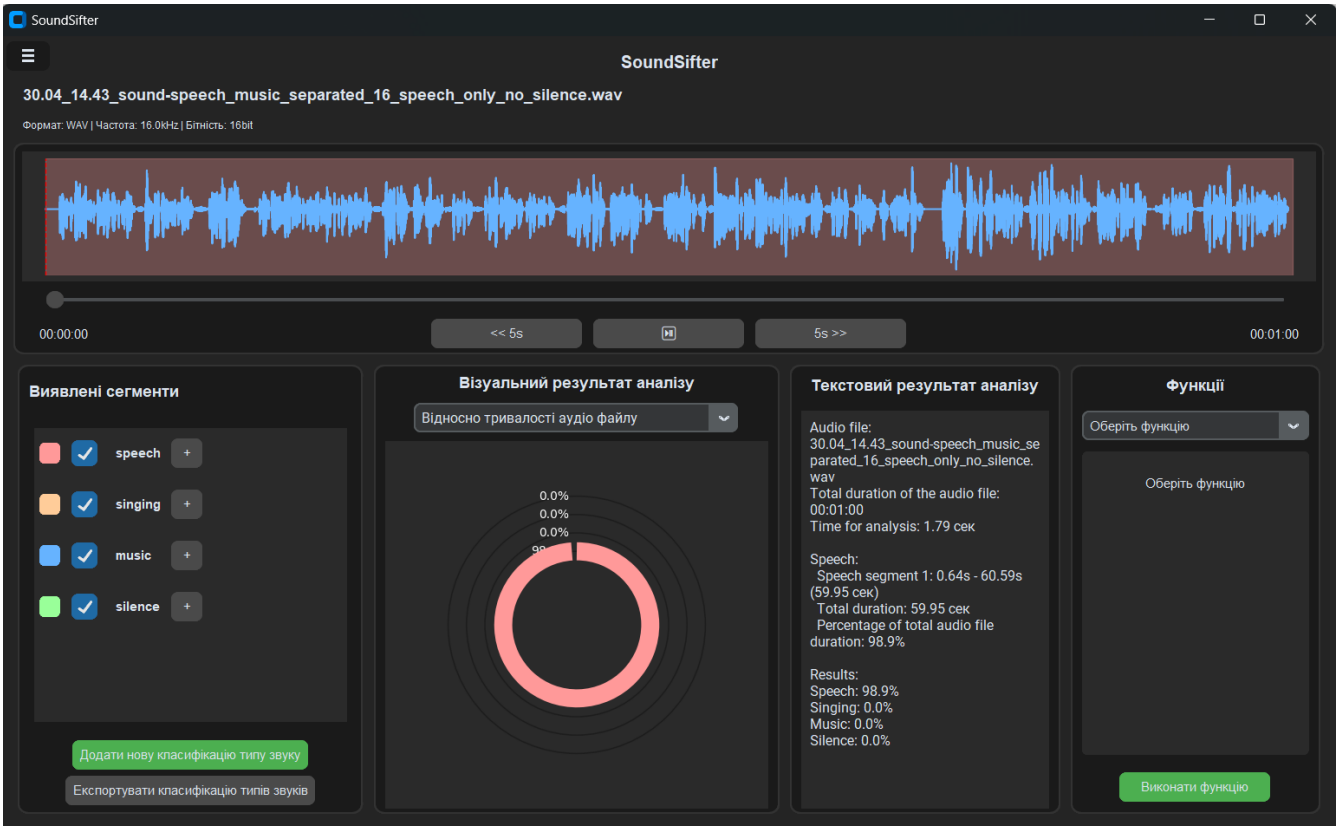

Рисунок 4.3 – Файл без проміжків тиші

Модуль шумозаглушення, що використовує DeepFilterNet2, було протестовано на звуковому файлі з фоновим шумом (шум фену). Після обробки шум майже повністю зник, а голос став чіткішим, що підтверджується суб’єктивною оцінкою якості звуку та подальшою транскрипцією, яка показала кращі результати порівняно з необробленим файлом. На рисунку 4.4 зображено звукову доріжку на якій видно стійкий сигнал шуму.

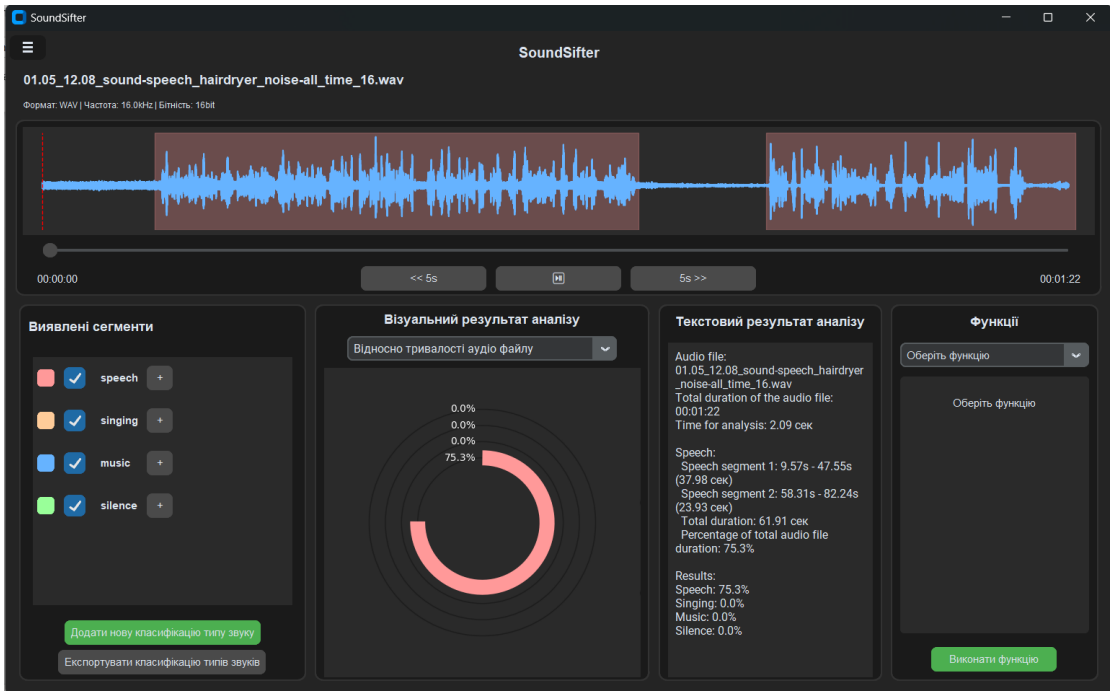

Рисунок 4.4 – Звуковий файл з шумом

Після виконання функції шумозаглушення, було отримано файл, звукову доріжку якого зображено на рисунку 4.5, й на ній видно шомумовий сигнал пропав й було класифіковано сегменти тиші

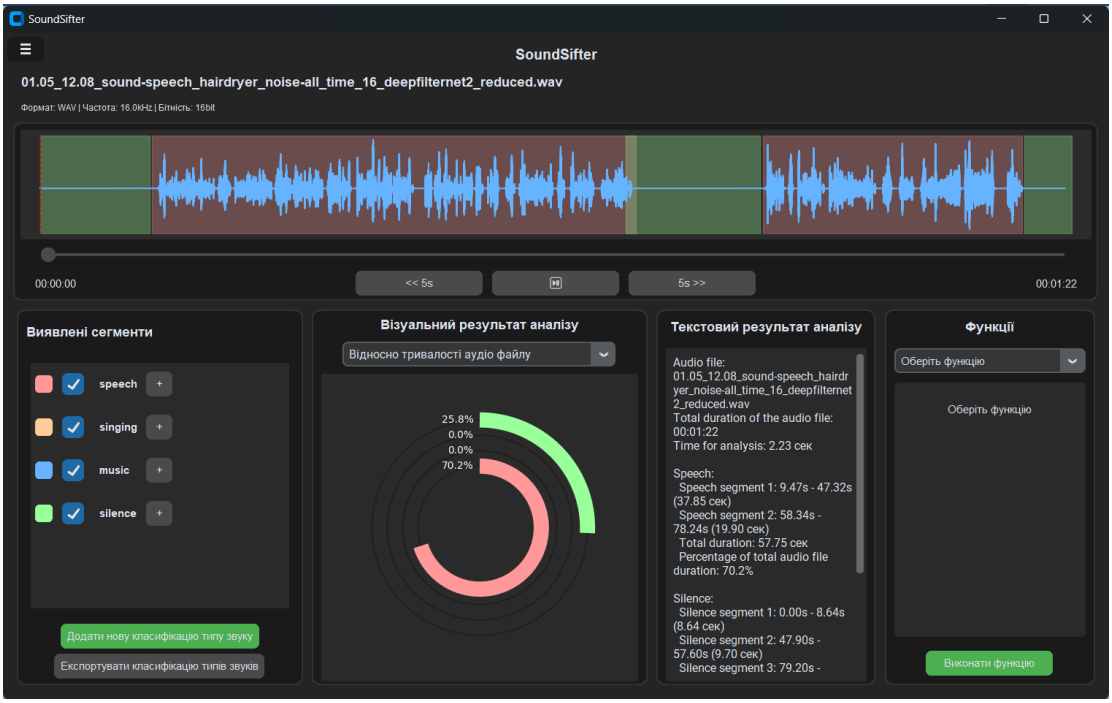

Рисунок 4.5 – Звуковий файл після шумозаглушення

За допомогою коду логування результату, було отримано ось таке порівняння з рівнем шуму (див. рисунок 4.6).

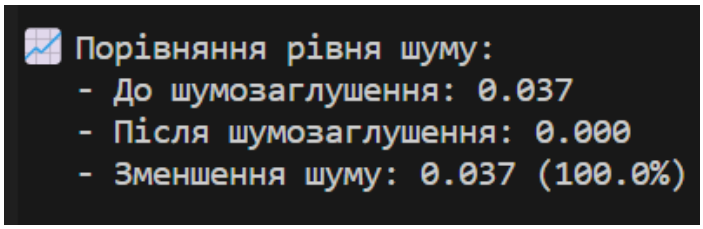

Рисунок 4.6 – Порівняння рівня шуму

Функція покращення звуку (нормалізація та фільтри) була перевірена на записі з перепадами рівня гучності (див. рисунок 4.7).

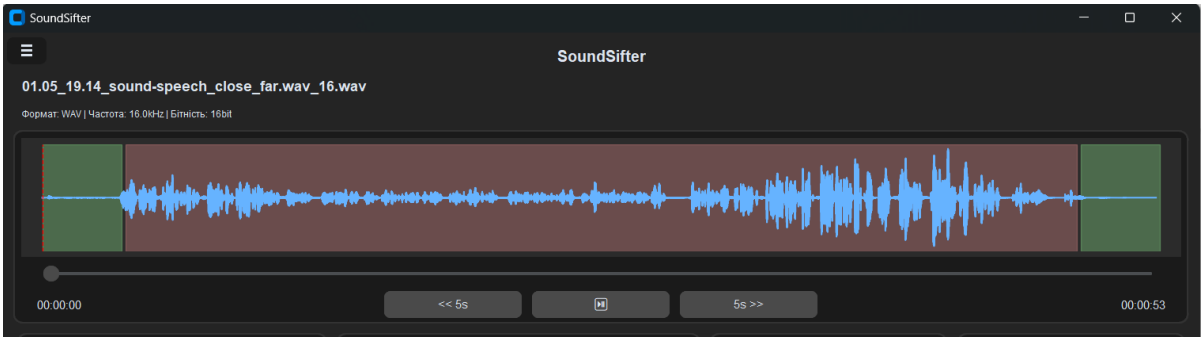

Рисунок 4.7 – Запис з перепадами гучності

Після обробки гучність звуку стала рівномірною, а голос – більш виразним, без спотворень (див. рисунок 4.8).

Рисунок 4.8 – Покращений, рівномірний звук

Функція перетворення мовлення на текст за допомогою моделі Whisper тестувалася на україномовному звуковому файлі тривалістю 1 хвилина. Тест проводився з увімкненим шумозаглушенням і покращенням звуку, а також із параметром мови «uk». Програма коректно створила SRT-файл із таймкодами та розпізнаним текстом, точність якого склала близько 90% для чіткого мовлення. У створеному .srt файлі відображався повний розпізнаний текст з таймкодами відповідних частин тексту. На рисунку 4.9 зображено вміст отриманого файлу .str.

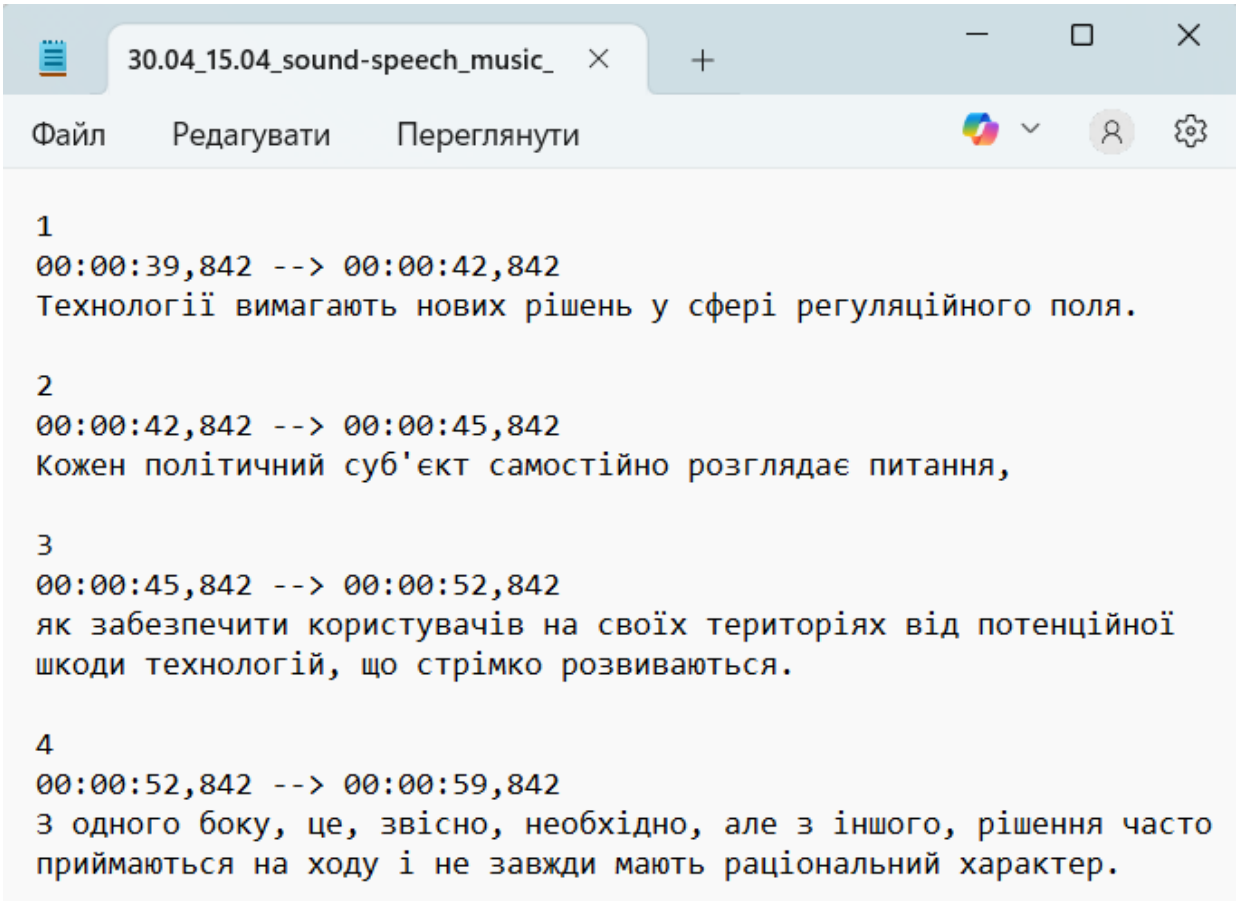

Рисунок 4.9 – Текст мовлення з файлу

Для оцінки точності транскрипції було використано набір даних LibriSpeech та метрику Word Error Rate (WER). Модель Whisper medium показала WER у межах 6-8%, що є конкурентоспроможним результатом порівняно з іншими сучасними системами розпізнавання мовлення. Це свідчить про високу ефективність застосованого підходу.

Інтерфейс користувача також був протестований: усі елементи, такі як кнопки відтворення, вибір файлів, налаштування параметрів і відображення сегментів, працювали коректно. Звукова доріжка оновлювалася в реальному часі під час відтворення, а візуальний та текстовий результати аналізу динамічно оновлювались в залежності від змін. Зависань чи візуальних помилок не виявлено.

На основі проведеного тестування функціональних можливостей застосунку було складено таблицю 4.1.

Таблиця 4.1 – Функціональні можливості застосунку

Номер	Назва	Умова тестування	Очікуваний результат	Фактичний результат	Результат
1	Аналіз звукових файлів	Завантаження файлу та запуск аналізу	Сегменти звуку визначаються коректно	Сегменти визначені з точністю до 0.5 сек	Успішно
2	Видалення всього, крім мовлення	Виконання функції з паузами 5 і 10 сек	Залишаються лише сегменти мовлення з паузами	Сегменти збережені коректно, без артефактів	Успішно
3	Видалення проміжків тиші	Виконання функції в режимі «all»	Усі сегменти тиші видаляються	Проміжки тиші видалені, тривалість відповідає очікуваній	Успішно
4	Шумозаглушення	Обробка файлу з фоновим шумом	Рівень шуму знижується, голос стає чіткішим	Шум повністю видалено	Успішно
5	Покращення звуку	Обробка файлу з низькою гучністю	Гучність нормалізується, звук стає виразнішим	Звук нормалізований, без спотворень	Успішно
6	Перетворення мовлення на текст	Транскрипція україномовного файлу	Створюється SRT-файл із коректними таймкодами	SRT створено, точність розпізнавання ~80%	Успішно
7	Експорт розмітки звуків звукового файлу	Обрано експорт розмітки .json файлу, у деяку папку	Створено .json файл з розміткою звуків у вказаній папці	У відповідні папці було сторено файл .json з розміткою звуків.	Успішно

Результати тестування свідчать про високу стабільність роботи програмного продукту SoundSifter та його готовність до використання кінцевими користувачами. Усі протестовані функції виконуються відповідно до заданих вимог і не викликають критичних помилок.

4.2 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску програмного продукту SoundSifter. Деталі щодо мінімальної та рекомендованої конфігурації розміщено в таблиці 4.2.

Таблиця 4.2 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Рекомендовано	Мінімально
Процесор	Intel Core i7 або аналогічний	Intel Core i3 або аналогічний
Оперативна пам'ять	16 ГБ RAM	8 ГБ RAM
Вільний простір на диску	6 ГБ	6 ГБ
Відеокарта	NVIDIA GeForce RTX 3060 або аналогічна	NVIDIA GeForce GTX 1050
Операційна система	Windows 11	Windows 10
Монітор	Роздільна здатність не менше 1920x1080 пікселів, діагональ 21" або більше	Роздільна здатність не менше 1280x720 пікселів, діагональ 15" або більше

Після встановлення та запуску програмного забезпечення SoundSifter користувач автоматично потрапляє на початковий екран застосунку. Першим кроком взаємодії передбачається вибір звукового файлу у підтримуваному форматі через кнопку «Обрати файл» у меню або на по центрі екрану. Після завантаження звукового файлу інтерфейс динамічно оновлюється: з’являються чекбокси для вибору необхідних операцій та кнопка запуску обробки.

Натиснувши «Старт», користувач запускає процес обробки обраних користувачем дій. Прогрес виконання відображається у спеціальному вікні, що дозволяє відслідковувати стан операцій у реальному часі. Коли обробка

завершується, вікно прогресу закривається, й інтерфейс динамічно оновлюється до головного вигляду.

У головному вікні користувач має змогу переглянути аналіз та розмітку звукового файлу у різних виглядах. Верхня частина вікна відведена для відображення звукової доріжки, на якій у вигляді графіка представлена амплітуда аудіо сигналу, з виділеними на ній сегментами звуків у вигляді кольорових зон. Під графіком розташовані зручні елементи керування відтворенням: кнопки для відтворення та паузи, перемотування вперед і назад на 5 секунд, прогрес-бар із повзунком, з якими може взаємодіяти користувач для керування відтворенням звукового файлу.

Нижня частина інтерфейсу поділена на чотири функціональні блоки. Лівий блок містить розкладні меню зі списком сегментів за типом звуку за допомогою відповідних кнопок користувач може додати розмітку нового звуку та його сегментів, як і редагувати або видалити їх. Є можливість обрати інший колір відображення виділення сегментів звуку на звуковій доріжці, а також, за допомогою чекбоксів, чи відображати виділення взагалі. За допомогою кнопки експорту користувач може обрати формат та папку збереження й зберегти розмітку звукового файлу. Поруч розміщено діаграму, яка наочно демонструє процентне співвідношення різних типів звуку відповідно до тривалості звукового файлу. Наступний блок відведений для текстового опису результатів аналізу, де розмітка звуків подана у зрозумілій формі. Останній, п’ятий блок – це меню вибору функцій, доповнене динамічною панеллю налаштувань, що змінюється відповідно до обраної операції. Користувач може обрати одну з доступних функцій у випадаючому списку функцій, таких як:

- «аналіз - розмітка звуку» для класифікації аудіо на сегменти;
- «видалити все, крім мовлення» для збереження лише мовних частин;
- «видалити проміжки тиші» для видалення пауз;
- «шумозаглушення» для зменшення фонового шуму;
- «покращення звуку» для підвищення якості аудіо;
- «перетворення на текст» для транскрибування мовлення.

У динамічній панелі користувач може налаштувати параметри кожної функції: папку для збереження, поріг тиші чи мову для транскрибування.

Таким чином, інтерфейс SoundSifter створений з метою забезпечити інтуїтивно зрозумілу та комфортну роботу із звуковими файлами, надаючи користувачу швидкий доступ до широкого набору потужних інструментів для обробки, аналізу й трансформації звукових файлів.

4.3 Застосування результатів аудіорозмітки

Розроблений застосунок SoundSifter забезпечує автоматичну розмітку аудіо з можливістю експорту результатів у формати, сумісні з системами машинного навчання (наприклад, JSON, CSV з таймкодами, TextGrid). Це відкриває широкі можливості його використання в різних галузях, де потрібна класифікація, сегментація або аналіз звукових сигналів. Завдяки підтримці сучасних моделей для виявлення мовлення (Silero VAD), класифікації типу звуку (YAMNet) та транскрипції (Whisper), система може виділяти мовлення, музику, тишу та інші акустичні події у складних аудіозаписах.

Одним із найбільш перспективних напрямів використання застосунку є медіаіндустрія зокрема, радіостанції, які працюють з великими обсягами архівного контенту. Застосунок дозволяє автоматично визначити, яка частина запису містить мовлення або музичні треки. Це значно полегшує індексацію, повторне використання або аналітику архівного контенту. Наприклад, можна швидко оцінити, яка частина ефіру припадала на музику протягом певного періоду, без залучення людських розмітників.

У сферах машинного навчання та штучного інтелекту SoundSifter може використовуватися як інструмент для підготовки навчальних даних. Якісно анотовані звукові файли ключова умова для тренування моделей розпізнавання мовлення, класифікації акустичних сцен або виявлення емоцій. Автоматична розмітка прискорює цей процес у десятки разів у порівнянні з ручним анотуванням. Крім того, система дозволяє уникнути людських похибок, забезпечуючи стабільну та однорідну розмітку великих аудіодатасетів.

Крім того, система має застосування у створенні мультимедійного контенту. Завдяки точній розмітці мовлення з таймкодами, можна автоматично створювати

субтитри для аудіо подкастів, синхронізувати аудіо з текстом, автоматично нарізати фрагменти за змістом або темою. Це значно скорочує час підготовки контенту до публікації, особливо для великих відеоплатформ або подкаст-мереж [29].

Окремо варто відзначити важливість точного визначення мовлення у транскрипційних завданнях. Whisper, що використовується у системі, зазвичай транскрибує увесь звук включаючи шум, музику, паузи, що знижує якість розпізнавання. Комбінування Whisper із Silero VAD і YAMNet дозволяє попередньо відфільтрувати лише релевантні сегменти з мовленням. Це підвищує точність транскрипції, знижує WER (Word Error Rate), а також економить обчислювальні ресурси.

Також при доопрацюванні – реалізації розмітки у реальному або майже реальному часі, застосунок зможе бути використаний у сфері безпеки та моніторингу. Наприклад, аудіорозмітка дозволяє виявляти сигнали тривоги, крики або інші звуки, що свідчать про потенційно небезпечні ситуації. Експорт сегментованого аудіо може бути інтегрований у системи автоматичного сповіщення або відеоспостереження з аудіоканалами. Таким чином, SoundSifter зможе виступити частиною більших мультисенсорних систем контролю.

При додаванні модулю виявлення емоцій, пошуку фраз та діаризації мовців, застосунок зможе бути використаний у customer service та кол-центрах, де аудіорозмітка допомагає аналізувати дзвінки клієнтів наприклад, виділяти ключові фрази, емоційні реакції або виявляти момент, коли клієнт переходить до ескалації конфлікту. Поєднання виявлення мовлення, трансформації в текст і подальшої обробки через NLP-інструменти дозволяє підвищити ефективність обслуговування та вчасно реагувати на проблемні кейси.

Таким чином, SoundSifter не лише дозволяє автоматизувати процеси розмітки аудіо, але й може використовуватись у сферах машинного навчання, штучного інтелекту, аудіо архівування та мультимедійного контенту. А при невеликому доопрацюванні зможе також бути використаний у ширшому спектрі практичних сценаріїв – від безпеки до аналізу клієнтських дзвінків. Гнучкість формату експорту

та можливість реалізації інтеграції з іншими системами робить його універсальним інструментом для сучасних потреб у роботі з аудіоданими.

4.4 Висновки

У результаті проведеного тестування розробленого програмного забезпечення SoundSifter було підтверджено його працездатність та відповідність функціональним вимогам. Було протестовано всі основні функції. Усі функції працюють коректно, забезпечуючи обробку звукових файлів без збоїв або критичних помилок.

Обраний метод ручного тестування дозволив ефективно перевірити логіку роботи застосунку та виявити недоліки на етапі розробки. Окрім тестування, було розроблено інструкцію користувача, яка забезпечує зрозуміле та покрокове ознайомлення з інтерфейсом SoundSifter і його можливостями, включаючи технічні вимоги та послідовність дій для обробки звукових файлів.

Крім того, важливим результатом тестування стала підтверджена ефективність розмітки аудіо, яка автоматично виділяє різні елементи, такі як мовлення, музика або реклама. Це значно полегшує аналіз великих архівних записів, зокрема в радіостанціях та медіаіндустрії. Завдяки можливості експорту розмітки в сумісні з ML формати, застосунок може бути використаний для подальшої обробки або тренування моделей машинного навчання, що розширює його потенціал в професійних сферах.

Таким чином, можна зробити висновок, що створене програмне забезпечення готове до практичного використання кінцевими користувачами та здатне ефективно виконувати покладені на нього задачі з аналізу, обробки, розмітки та транскрибування звукових файлів.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі було розроблено програмний застосунок SoundSifter, призначений для аналізу, обробки та транскрибування звукових файлів. У процесі реалізації системи було визначено основні вимоги до функціоналу, спроєктовано архітектуру застосунку та обрано відповідні технології розробки. Завдяки використанню бібліотек Python, таких як pydub, soundfile, matplotlib, whisper, а також фреймворку customtkinter для створення графічного інтерфейсу, вдалося ефективно інтегрувати необхідні функції для роботи з звуковими файлами. Бакалаврську кваліфікаційну роботу виконано згідно методичних вказівок [30].

У межах виконання роботи були виконані наступні етапи:

1. Проаналізовано основні підходи до обробки звукових сигналів в звукових файлах.
2. Розроблено програмну архітектуру для обробки звукових файлів.
3. Реалізовано алгоритми виявлення мовлення, класифікації типів аудіо.
4. Інтегровано методи очищення аудіосигналу від шумів та покращення якості звуку.
5. Реалізовано модуль розпізнавання мовлення з подальшою транскрипцією.
6. Створено зручний графічний інтерфейс користувача.
7. Проведено тестування системи на реальних звукових даних.

Особливу увагу було приділено обробці винятків, стабільності роботи та коректності обробки звукових файлів. Графічний інтерфейс, розроблений за допомогою бібліотеки customtkinter, забезпечує інтуїтивно зрозумілу взаємодію з користувачем і адаптивний дизайн. Програмна логіка реалізована на Python, що дозволяє ефективно інтегрувати алгоритми обробки аудіо та доступ до сучасних бібліотек для аналізу звуку.

Під час тестування було перевірено всі режими роботи системи. Результати тестування підтвердили працездатність та коректність виконання кожного з

функціональних модулів. Усі основні сценарії роботи, від аналізу звукових файлів до їх транскрибування, були успішно виконані без критичних помилок.

Окремо варто відзначити впровадження можливості розмітки звукових файлів, що дозволяє автоматично класифікувати різні типи аудіо (мовлення, музика, реклама) та експортувати цю розмітку у формати, сумісні з машинним навчанням. Це значно розширює функціональність застосунку, надаючи можливості для використання в професійних задачах, таких як архівні системи для радіостанцій.

Таким чином, метою бакалаврської кваліфікаційної роботи було досягнуто – створено функціональний, зручний у використанні та надійний застосунок для обробки звукових файлів, який може бути використаний як у навчальних цілях, так і як інструмент для аналізу та транскрибування звукових даних у професійних задачах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. XAudio Labeling services for ml [Електронний ресурс] – Режим доступу до ресурсу: https://unidata.pro/data-labeling/audio/

2. Zhang, C. et al. (2022). Audio Segmentation Techniques and Applications Based on Deep Learning. MDPI Electronics [Електронний ресурс] – Режим доступу до ресурсу: https://www.mdpi.com/2079-9292/11/3/432

3. Extensive Guide to Audio Annotation. Everything You Need to Know! [Електронний ресурс] – Режим доступу до ресурсу: https://www.futurebeeai.com/blog/audio-annotation-guide

4. Explore The Challenges and Solutions in Audio Annotation [Електронний ресурс] – Режим доступу до ресурсу: https://annotationbox.com/explore-the-challenges-and-solutions-in-audio-annotation/

5. Ткаченко О. М., Дажура О. Р. Розробка програмного забезпечення для автоматизованої розмітки звукових файлів. Вінниця: ВНТУ, 2025. [Електронний ресурс]. Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/23901/19730.

6. Agarwal, S., Bist, L., Sharma, S. K., Dular, S. K., & Salvi, R. (2024). Intelligent IoT-Based Audio Signal Processing for Healthcare Applications. Journal of Intelligent Systems and Internet of Things, 83-89 p. [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/383092450_Intelligent_IOT_Based_Audio_Signal_Processing_for_Healthcare_Applications

7. Lerch, A. An Introduction to Audio Content Analysis: Music Information Retrieval Tasks and Applications (2nd ed.). – Hoboken, New Jersey: John Wiley & Sons,— 464 p.

8. A Review of Deep Learning Techniques for Speech Processing. [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/abs/2305.00359

9. Speech Emotion Recognition Using Deep Learning Techniques: A Review [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/

publication/335360469_Speech_Emotion_Recognition_Using_Deep_Learning_Techniques_A_Review

10. Automatic transcription software: fast, easy, accurate. [Електронний ресурс] – Режим доступу до ресурсу: https://sonix.ai/en/automated-transcription

11. Sonix.ai transcription software review [Електронний ресурс] – Режим доступу до ресурсу: https://www.techradar.com/reviews/sonixai

12. Descript AI Review 2025 [Електронний ресурс] – Режим доступу до ресурсу: https://www.elegantthemes.com/blog/business/descript-ai-review

13. Speechmatics review [Електронний ресурс] – Режим доступу до ресурсу: https://www.techradar.com/reviews/speechmatics-review

14. Annotate Audio Data In Encord. [Електронний ресурс] – Режим доступу до ресурсу: https://encord.com/blog/annotate-audio/

15. SileroVAD : Machine Learning Model to Detect Speech Segments [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/axinc-ai/silerovad-machine-learning-model-to-detect-speech-segments-e99722c0dd41

16. YAMNET for audio classification [Електронний ресурс] – Режим доступу до ресурсу: https://vtiya.medium.com/yamnet-for-audio-classification-867ff0ca9197

17. DeepFilterNet2: Towards Real-Time Speech Enhancement on Embedded Devices for Full-Band Audio [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/abs/2205.05474

18. Introducing Whisper [Електронний ресурс] – Режим доступу до ресурсу: https://openai.com/index/whisper/

19. Matplotlib documentation [Електронний ресурс] – Режим доступу до ресурсу: https://matplotlib.org/stable/index.html

20. Visual Paradigm. Business Process Modeling Using UML Activity Diagrams. [Електронний ресурс] – Режим доступу до ресурсу: https://guides.visual-paradigm.com/business-process-modeling-using-uml-activity-diagrams/

21. Go UML. Business Process Modeling Using UML Activity Diagrams. [Електронний ресурс] – Режим доступу до ресурсу: https://www.go-uml.com/business-process-modeling-using-uml-activity-diagrams/

22. Chaudhuri, A. B. Flowchart and Algorithm Basics: The Art of Programming. – Rockville, Maryland: Mercury Learning and Information, 2020. – 340 p.

23. Component Based Diagram - Unified Modeling Language (UML) [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/component-based-diagram/

24. Cervantes, H., Kazman, R. Designing Software Architectures: A Practical Approach 2nd Edition / Humberto Cervantes, Rick Kazman. – Boston: Addison-Wesley Professional, 2024. – 336 p.

25. Automated Functional Testing for Audio & Video Apps [Електронний ресурс] – Режим доступу до ресурсу: https://www.testdevlab.com/blog/how-do-we perform-automated-functional-testing-for-audio-and-video-apps

26. Okken B. Python Testing with pytest: Simple, Rapid, Effective, and Scalable / B. Okken. — 2nd ed. — Dallas: Pragmatic Bookshelf, 2022. — 274 p.

27. Unit testing audio processors with JUCE & Catch2 [Електронний ресурс] – Режим доступу до ресурсу: https://ejaaskel.dev/unit-testing-audio-processors-with juce-catch2/

28. Top Manual UI Testing Tools for Superior User Experience [Електронний ресурс] – Режим доступу до ресурсу: https://www.browserstack.com/guide/manual ui-testing-tools

29. Audio Labeling services for ml [Електронний ресурс] – Режим доступу до ресурсу: Audio Labeling Services for ML/AI Models - unidata.pro

30. Романюк О. Н., Чорноволик Г. О., Методичні вказівки до виконання бакалаврських дипломних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення»: методичні – вказівки Вінниця : ВНТУ, 2022. 51 с

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ

д.т.н., професор Романюк О. Н.

«25» березня 2025 року

Технічне завдання

на бакалаврську кваліфікаційну роботу

«Розробка програмного забезпечення для автоматичної розмітки звукових файлів»

за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:

к.т.н., доц. каф. ПЗ. О. М. Ткаченко

«24» березня 2025р.

Виконав:

студент гр. ЗПІ-21б О. Р. Дажура

«24» березня 2025р.

м. Вінниця – 2025

Додаток Б – Протокол перевірки роботи на плагіат

Назва роботи: Розробка програмного забезпечення для автоматичної розмітки звукових файлів

Тип роботи: Бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра ПЗ, ФІТКІ, ЗПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 49 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ☐ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Ткаченок О. М., к.т.н., доцент кафедри ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Дажура О. Р.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програми

states.py:

```
import customtkinter as ctk
import pygame
import numpy as np
import soundfile as sf
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt
from tkinter import messagebox, filedialog
from CTkMessagebox import CTkMessagebox
from gui.audio_functions_panel import AudioFunctionsPanel
from inspection_and_preparation.audio_preparation import prepare_audio
from classifier.silero_vad_yamnet_classifier import classify_audio
from processing.remove_interval import keep_only_speech, remove_silence
from noise_reduction.deep_filter_net2.deep_filter_net2_n_red import reduce_audio_noise_deepfilternet
from speech_enhancement.enhance_speech import enhance_speech
from speech_to_text.transcribe import transcribe_audio
from gui.popups import ProgressPopup, ExportPopup, ConfirmFileOpenPopup, AnalysisSettingsPopup
from gui.about_window import AboutWindow
import os
import json
from pathlib import Path
from gui.layout_of_main_interface import SegmentWidget, AccordionSoundType, SoundSegmentsPanel, BurgerMenu
from gui.localization import LocalizationManager
import sounddevice as sd
import time
from datetime import timedelta
from matplotlib.patches import Arc
import warnings
import whisper
from gui.theme_manager import ThemeManager

class StartScreenState(ctk.CTkFrame):
    def __init__(self, master, select_file_callback, show_about_callback, show_settings_callback, show_theme_callback, show_language_callback, localization, theme_manager):
        self.theme_manager = theme_manager
        super().__init__(master, fg_color=self.theme_manager.get_color("main_bg"))
        self.select_file_callback = select_file_callback
        self.show_about_callback = show_about_callback
        self.show_settings_callback = show_settings_callback
        self.show_theme_callback = show_theme_callback
        self.show_language_callback = show_language_callback
        self.localization = localization
```

```
        # Заголовок SoundSifter
        self.lbl_title = ctk.CTkLabel(self, text=self.localization.get("title"), font=("Arial", 16, "bold"))
        self.lbl_title.place(relx=0.5, y=10, anchor="n")

        # Кнопка меню
        self.btn_burger = ctk.CTkButton(self, text="☰", font=("Arial", 14), width=40, command=self.toggle_burger, fg_color=self.theme_manager.get_color("main_bg"), hover_color=self.theme_manager.get_color("block_bg"))
        self.btn_burger.place(x=5, y=5)

        self.burger_menu = BurgerMenu(self)
        self.burger_menu.btn_file.configure(command=self.select_file_callback, text=self.localization.get("select_file"))
        self.burger_menu.btn_theme.configure(command=self.show_theme_callback, text=self.localization.get("theme"))
        self.burger_menu.btn_about.configure(command=self.show_about_callback, text=self.localization.get("about"))
        self.burger_menu.btn_settings.configure(command=self.show_settings_callback, text=self.localization.get("settings"))
        self.burger_menu.btn_language.configure(command=self.show_language_callback, text=self.localization.get("language"))
        self.burger_menu.lower()

    # - - - # Блок отримання
        self.block_frame = ctk.CTkFrame(self, fg_color=self.master.theme_manager.get_color("block_bg"), corner_radius=10)
        self.block_frame.place(relx=0.015, rely=0.07, relwidth=0.97, relheight=0.9)  # Відступи: 15px з боків, 55px зверху

        # Текст "Обери файл" та підтримувані формати
        self.lbl_prompt = ctk.CTkLabel(
            self.block_frame,
            text=self.localization.get("choose_file"),
            font=("Arial", 24),
            wraplength=800
        )
        self.lbl_prompt.place(relx=0.5, rely=0.5, anchor="center")

        # Кнопка "Обрати файл"
        self.btn_select_file = ctk.CTkButton(
            self.block_frame,
            text=self.localization.get("select_file"),
            command=self.select_file_callback,
            font=("Arial", 12),
            fg_color=self.theme_manager.get_color("button_fg"),
            hover_color=self.theme_manager.get_color("button_hover")
        )
        self.btn_select_file.place(relx=0.5, rely=0.6, anchor="center")

        # Кнопка "Про програму"
        self.btn_about = ctk.CTkButton(
            self.block_frame,
```

```
            text=self.localization.get("about"),
            command=self.show_about_callback,
            font=("Arial", 12)
        )
        self.btn_about.place(relx=0.5, rely=0.7, anchor="center")

    def toggle_burger(self):
        self.burger_menu.toggle()
        self.btn_burger.lift()

class ActionSelectorState(ctk.CTkFrame):
    def __init__(self, master, start_processing_callback, select_file_callback, show_about_callback,
show_settings_callback, show_theme_callback, show_language_callback, localization, theme_manager):
        self.theme_manager = theme_manager
        super().__init__(master, fg_color=self.theme_manager.get_color("main_bg"))
        self.start_processing_callback = start_processing_callback
        self.select_file_callback = select_file_callback
        self.show_about_callback = show_about_callback
        self.show_settings_callback = show_settings_callback
        self.show_theme_callback = show_theme_callback
        self.show_language_callback = show_language_callback
        self.localization = localization

        # Заголовок SoundSifter
        self.lbl_title = ctk.CTkLabel(self, text=self.localization.get("title"), font=("Arial", 16, "bold"))
        self.lbl_title.place(relx=0.5, y=10, anchor="n")

        # Кнопка меню
        self.btn_burger = ctk.CTkButton(self, text="☰", font=("Arial", 14), width=40,
command=self.toggle_burger, fg_color=self.theme_manager.get_color("main_bg"),
hover_color=self.theme_manager.get_color("block_bg"))
        self.btn_burger.place(x=5, y=5)


        self.burger_menu = BurgerMenu(self)
        self.burger_menu.btn_file.configure(command=self.select_file_callback,
text=self.localization.get("select_file"))
        self.burger_menu.btn_theme.configure(command=self.show_theme_callback,
text=self.localization.get("theme"))
        self.burger_menu.btn_about.configure(command=self.show_about_callback,
text=self.localization.get("about"))
        self.burger_menu.btn_settings.configure(command=self.show_settings_callback,
text=self.localization.get("settings"))
        self.burger_menu.btn_language.configure(command=self.show_language_callback,
text=self.localization.get("language"))
        self.burger_menu.lower()

        # Панель вибору дій
        self.actions_frame = ctk.CTkFrame(self,
fg_color=self.master.theme_manager.get_color("block_bg"), corner_radius=10)
        self.actions_frame.place(relx=0.5, rely=0.5, anchor="center")
```

```
            ctk.CTkLabel(self.actions_frame, text=self.localization.get("select_actions"), font=("Arial", 14, "bold")).pack(pady=10, padx=20)

            self.actions = {
                "load_only": ctk.BooleanVar(value=False),
                "analyze": ctk.BooleanVar(value=True),
                "transcribe": ctk.BooleanVar(value=False),
                "denoise": ctk.BooleanVar(value=False),
                "enhance": ctk.BooleanVar(value=False)
            }

            ctk.CTkCheckBox(self.actions_frame, text=self.localization.get("load_only"), variable=self.actions["load_only"]).pack(anchor="w", padx=20, pady=5)
            ctk.CTkCheckBox(self.actions_frame, text=self.localization.get("analyze"), variable=self.actions["analyze"]).pack(anchor="w", padx=20, pady=5)
            ctk.CTkCheckBox(self.actions_frame, text=self.localization.get("transcribe"), variable=self.actions["transcribe"]).pack(anchor="w", padx=20, pady=5)
            ctk.CTkCheckBox(self.actions_frame, text=self.localization.get("denoise"), variable=self.actions["denoise"]).pack(anchor="w", padx=20, pady=5)
            ctk.CTkCheckBox(self.actions_frame, text=self.localization.get("enhance"), variable=self.actions["enhance"]).pack(anchor="w", padx=20, pady=5)

            self.btn_start = ctk.CTkButton(self.actions_frame, text=self.localization.get("start"), command=self.start_processing, font=("Arial", 12), fg_color=self.theme_manager.get_color("button_fg"), hover_color=self.theme_manager.get_color("button_hover"))
            self.btn_start.pack(pady=20)

        def toggle_burger(self):
            self.burger_menu.toggle()
            self.btn_burger.lift()

        def start_processing(self):
            actions_dict = {key: var.get() for key, var in self.actions.items()}
            self.start_processing_callback(actions_dict)

    class MainInterfaceState(ctk.CTkFrame):
        def __init__(self, master, select_file_callback, show_about_callback, show_settings_callback, show_theme_callback, show_language_callback, segment_manager, localization, theme_manager):
            self.theme_manager = theme_manager
            super().__init__(master, fg_color=self.theme_manager.get_color("main_bg"))
            self.master = master
            self.select_file_callback = select_file_callback
            self.show_about_callback = show_about_callback
            self.show_settings_callback = show_settings_callback
            self.show_theme_callback = show_theme_callback
            self.segment_manager = segment_manager
            self.show_language_callback = show_language_callback
            self.localization = localization
            self.audio_data = None
            self.sample_rate = None
            self.duration = 0
            self.current_position = 0
```

```
        self.is_playing = False # було просто playing а не is_playing
        self.paused = False
        self.after_id = None
        self.plot_data = None
        self.vertical_line = None
        self.stream = None
        self.is_slider_dragging = False
        self.current_frame = 0
        self.start_time = 0
        self.is_updating_waveform = False

        # Ініціалізація аудіоданих
        self.initialize_audio_data()

        # Заголовок SoundSifter
        self.lbl_title = ctk.CTkLabel(self, text=self.localization.get("title"), font=("Arial", 16, "bold"))
        self.lbl_title.place(relx=0.5, y=10, anchor="n")

        # Кнопка меню
        self.btn_burger = ctk.CTkButton(self, text="☰", font=("Arial", 14), width=40,
command=self.toggle_burger, fg_color=self.theme_manager.get_color("main_bg"),
hover_color=self.theme_manager.get_color("block_bg"))
        self.btn_burger.place(x=5, y=5)

        self.burger_menu = BurgerMenu(self)
        self.burger_menu.btn_file.configure(command=self.select_file_callback,
text=self.localization.get("select_file"))
        self.burger_menu.btn_theme.configure(command=self.show_theme_callback,
text=self.localization.get("theme"))
        self.burger_menu.btn_about.configure(command=self.show_about_callback,
text=self.localization.get("about"))
        self.burger_menu.btn_settings.configure(command=self.show_settings_callback,
text=self.localization.get("settings"))
        self.burger_menu.btn_language.configure(command=self.show_language_callback,
text=self.localization.get("language"))
        self.burger_menu.lower()

        # Назва файлу
        self.file_name_label = ctk.CTkLabel(self, text=os.path.basename(self.master.prepared_file),
font=("Arial", 14, "bold"))
        self.file_name_label.pack(anchor="nw", padx=20, pady=(40, 0))

        # Інформація про файл
        file_format = Path(self.master.prepared_file).suffix[1:].upper()
        self.file_info_label = ctk.CTkLabel(self, text=f"Формат: {file_format} | Частота:
{self.sample_rate/1000:.1f}kHz | Бітність: 16bit" if self.sample_rate else "Формат: {file_format} |
Частота: Н/Д | Бітність: Н/Д", font=("Arial", 10))
        self.file_info_label.pack(anchor="nw", padx=20, pady=(0, 0))

        # Основний фрейм
        self.main_frame = ctk.CTkFrame(self, fg_color="transparent")
        self.main_frame.pack(fill=ctk.BOTH, expand=True, padx=10, pady=(2, 10))
```

```
            # Блок 1: Хвиля
            if self.audio_data is not None:
                self.setup_waveform()
            else:
                self.setup_error_waveform()

            # Блок 2: Сегменти
            self.sound_segments_panel = SoundSegmentsPanel(self.main_frame,
self.on_update_segments)
            self.sound_segments_panel.grid(row=1, column=0, sticky="nsew", padx=5, pady=5)
            self.sound_segments_panel.canvas_seg.configure(width=400)
            self.populate_segments()

            # Перевизначення методу export_popup
            def custom_export_popup(self):
                has_transcription = os.path.exists(os.path.join("output",
f"{self.segment_manager.data['file_name'].rsplit('.', 1)[0]}_transcription.txt"))
                ExportPopup(self.sound_segments_panel, self.segment_manager, has_transcription)

            self.sound_segments_panel.export_popup =
custom_export_popup.__get__(self.sound_segments_panel, SoundSegmentsPanel)

            def custom_delete_type(self, widget):
                confirm = CTkMessagebox(title="Підтвердження", message=f"Ви дійсно хочете
видалити тип звуку '{widget.sound_type_data['name']}'?", icon="warning", option_1="Так",
option_2="Ні")
                if confirm.get() == "Так":
                    self.sound_segments_panel.delete_sound_type(widget)
                    self.on_update_segments()

            for widget in self.sound_segments_panel.sound_types_widgets:
                widget.delete_type = lambda: custom_delete_type(self, widget)

            # Блок 3: Donut chart
            self.setup_chart()

            # Блок 4: Текстовий результат
            self.setup_text_analysis()

            # Блок 5: Функції
            self.setup_functions()

            # Налаштування сітки
            self.main_frame.grid_columnconfigure(0, weight=2, minsize=400)
            self.main_frame.grid_columnconfigure(1, weight=2, minsize=2)
            self.main_frame.grid_columnconfigure(2, weight=3, minsize=300)
            self.main_frame.grid_columnconfigure(3, weight=3, minsize=300)
            self.main_frame.grid_rowconfigure(1, weight=1)

            # Оновлення текстового аналізу
            self.update_text_analysis()
```

```
            # Ініціалізація відтворення
            #if self.audio_data is not None:
            #    pygame.mixer.init()
            #    pygame.mixer.music.load(self.master.prepared_file)


        def initialize_audio_data(self):
            """Ініціалізація аудіоданих із перевіркою файлу"""
            print(f"Початок ініціалізації аудіоданих для файлу: {self.master.prepared_file}")
            if not os.path.exists(self.master.prepared_file):
                print(f"Помилка: Файл {self.master.prepared_file} не існує")
                CTkMessagebox(title="Помилка", message=f"Файл {self.master.prepared_file} не існує!", icon="cancel")
                return

            # Перевірка формату файлу
            supported_formats = ['.wav', '.flac', '.ogg', '.mp3', '.aiff', '.aif', '.opus']
            file_ext = os.path.splitext(self.master.prepared_file)[1].lower()
            if file_ext not in supported_formats:
                print(f"Помилка: Непідтримуваний формат файлу {file_ext}")
                CTkMessagebox(title="Помилка", message=f"Непідтримуваний формат файлу: {file_ext}", icon="cancel")
                return

            try:
                print(f"Спроба читання файлу: {self.master.prepared_file}")
                print(f"Перевірка доступності soundfile: {sf.__version__}")
                self.audio_data, self.sample_rate = sf.read(self.master.prepared_file)
                if self.audio_data.ndim > 1:
                    self.audio_data = self.audio_data[:, 0]  # Вибираємо mono канал
                self.audio_data = self.audio_data[:, np.newaxis]  # Перетворюємо в двовимірний масив
                print(f"Файл успішно прочитано: {self.master.prepared_file}, sample_rate: {self.sample_rate}, audio_data shape: {self.audio_data.shape}")
                self.duration = len(self.audio_data) / self.sample_rate
            except sf.LibsndfileError as e:
                print(f"Помилка Libsndfile при читанні файлу {self.master.prepared_file}: {e}")
                CTkMessagebox(title="Помилка", message=f"Помилка бібліотеки soundfile: {e}", icon="cancel")
                self.audio_data = None
                self.sample_rate = None
                self.duration = 0
            except Exception as e:
                print(f"Невідома помилка при читанні файлу {self.master.prepared_file}: {e}")
                import traceback
                traceback.print_exc()
                CTkMessagebox(title="Помилка", message=f"Не вдалося прочитати аудіофайл: {e}", icon="cancel")
                self.audio_data = None
                self.sample_rate = None
                self.duration = 0
```

```
        def setup_error_waveform(self):
            """Налаштування відображення, якщо аудіодані недоступні"""
            self.waveform_frame = ctk.CTkFrame(self.main_frame, fg_color=self.theme_manager.get_color("block_bg"), corner_radius=10, border_width=2, border_color=self.theme_manager.get_color("border_color"))
            self.waveform_frame.grid(row=0, column=0, columnspan=4, sticky="ew", padx=1, pady=(0, 5))
            ctk.CTkLabel(self.waveform_frame, text="Не вдалося завантажити аудіохвилю", font=("Arial", 12)).pack(pady=20)

        def toggle_burger(self):
            self.burger_menu.toggle()
            self.btn_burger.lift()

        def update_waveform_segments(self):
            if self.audio_data is None or not hasattr(self, 'ax'):
                return

            # Не очищаємо всю вісь, лише видаляємо старі сегменти
            for artist in self.ax.patches:  # Видаляємо попередні прямокутники (сегменти)
                artist.remove()
            if self.plot_data:
                self.plot_data.remove()  # Видаляємо попередню хвильову форму

            t = np.linspace(0, self.duration, len(self.signal))
            self.plot_data, = self.ax.plot(t, self.signal, color=self.theme_manager.get_color("waveform_line"))
            #self.ax.set_title("Waveform with detected segments", color="white")
            self.ax.axis('off')

            # Додаємо виділення сегментів з коригуванням часу
            for type_name, type_data in self.segment_manager.data["types"].items():
                if not type_data.get("visible", True):  # Перевірка видимості типу
                    continue
                color = type_data["color"]
                for segment in type_data["segments"]:
                    if segment["visible"]:
                        start_time = segment["start"]
                        end_time = segment["end"]
                        # Коригування меж для уникнення пропусків
                        adjusted_start = np.floor(start_time)
                        adjusted_end = np.ceil(end_time) - 0.2
                        self.ax.axvspan(adjusted_start, adjusted_end, alpha=0.3, color=color)

            # Оновлюємо вертикальну лінію
            if self.vertical_line is not None:
                try:
                    self.vertical_line.remove()
                except ValueError:
                    pass  # Ігноруємо, якщо об’єкт уже видалений
            self.vertical_line = self.ax.axvline(x=self.current_position, color='red', linestyle='--', linewidth=1)
```

```
            self.canvas.draw()


        def setup_waveform(self):
            self.waveform_frame = ctk.CTkFrame(self.main_frame, fg_color=self.theme_manager.get_color("block_bg"), corner_radius=10, border_width=2, border_color=self.theme_manager.get_color("border_color"))
            self.waveform_frame.grid(row=0, column=0, columnspan=4, sticky="ew", padx=1, pady=(0, 5))

            #self.waveform_frame.update_idletasks()  # Оновити геометрію перед викликом

            # Ініціалізація з початковою шириною
            self.update_waveform_width()

            def on_frame_configure(event):
                self.update_waveform_width()

            self.waveform_frame.bind("<Configure>", on_frame_configure)


            # Встановлюємо ширину fig залежно від доступного простору
            # Динамічна ширина на основі батьківського фрейму
            '''
            available_width = self.waveform_frame.winfo_width() - 20  # Віднімаємо 20px для відступів
            if available_width <= 0:
                available_width = self.winfo_screenwidth() * 0.8 - 20
            self.fig = Figure(figsize=(available_width / 100, 1.5), dpi=100)
            self.ax = self.fig.add_subplot(111)
            '''
            t = np.linspace(0, self.duration, len(self.signal))
            self.fig = Figure(figsize=(self.available_width / 100, 1.5), dpi=100)
            self.ax = self.fig.add_subplot(111)
            self.fig.set_facecolor(self.theme_manager.get_color("waveform_bg"))
            self.ax.set_facecolor(self.theme_manager.get_color("waveform_bg"))
            self.plot_data, = self.ax.plot(t, self.signal, color=self.theme_manager.get_color("waveform_line"))
            #self.ax.set_title("Waveform with detected segments", color="white")
            self.ax.axis('off')
            self.fig.tight_layout(pad=0)
            self.fig.subplots_adjust(left=-0.03, right=1.03, top=0.95, bottom=0.05)
            self.canvas = FigureCanvasTkAgg(self.fig, master=self.waveform_frame)
            self.canvas.draw()
            self.canvas.get_tk_widget().pack(fill=ctk.BOTH, expand=True, padx=10, pady=(10, 5))  # Відступи 10px


            # Додаємо вертикальну лінію
            self.vertical_line = self.ax.axvline(x=0, color='red', linestyle='--', linewidth=1)

            # Синхронізація ширини смужки прогресу з хвилею з відступами
```

```
            self.progress_bar = ctk.CTkSlider(self.waveform_frame, from_=0, to=self.duration, width=self.available_width - 20, button_color=self.theme_manager.get_color("progress_bar_button"), progress_color=self.theme_manager.get_color("progress_bar_progress"), command=self.on_slider_move)
            self.progress_bar.set(0)
            self.progress_bar.pack(pady=5, padx=(30, 30))  # Відступи 10px з обох боків
            self.progress_bar.bind("<ButtonRelease-1>", self.slider_released)

            self.controls_container = ctk.CTkFrame(self.waveform_frame, fg_color="transparent")
            self.controls_container.pack(fill="x", padx=(4, 4), pady=5)

            self.time_label_left = ctk.CTkLabel(self.controls_container, text="00:00:00", font=("Arial", 12))
            self.time_label_left.pack(side="left", padx=(20, 0))

            self.controls_frame = ctk.CTkFrame(self.controls_container, fg_color="transparent")
            self.controls_frame.pack(side="left", expand=True)

            self.btn_rewind = ctk.CTkButton(self.controls_frame, text="<< 5s", font=("Arial", 12), fg_color=self.theme_manager.get_color("block_1_button"), hover_color=self.theme_manager.get_color("block_1_button_hover"), command=self.rewind_5s)
            self.btn_rewind.pack(side="left", padx=5)
            self.btn_play_pause = ctk.CTkButton(self.controls_frame, text="⏯", font=("Arial", 12), fg_color=self.theme_manager.get_color("block_1_button"), hover_color=self.theme_manager.get_color("block_1_button_hover"), command=self.toggle_play_pause)
            self.btn_play_pause.pack(side="left", padx=5)
            self.btn_forward = ctk.CTkButton(self.controls_frame, text="5s >>", font=("Arial", 12), fg_color=self.theme_manager.get_color("block_1_button"), hover_color=self.theme_manager.get_color("block_1_button_hover"), command=self.forward_5s)
            self.btn_forward.pack(side="left", padx=5)

            self.time_label_right = ctk.CTkLabel(self.controls_container, text=self.format_time(self.duration), font=("Arial", 12))
            self.time_label_right.pack(side="right", padx=(0, 20))

            # Оновлення таймкоду та слайдера
            self.update_progress()

            self.update_waveform_segments()  # Додаємо виклик після ініціалізації



        def update_waveform_width(self):
            available_width = self.waveform_frame.winfo_width() - 20  # Віднімаємо 20px для відступів
            if available_width <= 0:
                available_width = self.winfo_screenwidth() * 0.8 - 20
            self.available_width = available_width

            # Оновлюємо Figure, якщо вона вже створена
            if hasattr(self, 'fig'):
                self.fig.set_size_inches(available_width / 100, 1.5)
                self.canvas.draw()
```

```
            # Оновлюємо ширину прогрес-бару з урахуванням відступів
            effective_width = available_width - 60  # Віднімаємо 10px з кожного боку (загалом 20px)
            if hasattr(self, 'progress_bar'):
                self.progress_bar.configure(width=effective_width)



        def setup_chart(self):
            self.chart_frame = ctk.CTkFrame(self.main_frame, fg_color=self.theme_manager.get_color("block_bg"), corner_radius=10, border_width=2, border_color=self.theme_manager.get_color("border_color"))
            self.chart_frame.grid(row=1, column=1, sticky="nsew", padx=5, pady=5)
            ctk.CTkLabel(self.chart_frame, text=self.localization.get("visual_analysis"), font=("Arial", 14, "bold")).pack(pady=2)

            # Додаємо випадаючий список
            self.chart_mode = ctk.StringVar(value="Відносно інших типів звуків")
            self.mode_dropdown = ctk.CTkComboBox(
                self.chart_frame,
                values=["Відносно інших типів звуків", "Відносно тривалості аудіо файлу"],
                variable=self.chart_mode,
                command=self.update_chart_mode,
                #width=int(self.chart_frame.winfo_width() * 0.7)  # 70% ширини фрейму
                width=300  # Фіксована ширина, наприклад, 300 пікселів
            )
            self.mode_dropdown.pack(pady=2)

            self.chart_inner_frame = ctk.CTkFrame(self.chart_frame, fg_color=self.theme_manager.get_color("chart_bg"), corner_radius=5)
            self.chart_inner_frame.pack(fill=ctk.BOTH, expand=True, padx=10, pady=5)

            self.update_chart()

        def update_chart_mode(self, event=None):
            self.update_chart()


        def update_chart(self):
            for widget in self.chart_inner_frame.winfo_children():
                widget.destroy()

            stats = self.segment_manager.data["stats"]
            labels = list(stats.keys())
            colors = [self.segment_manager.data["types"][label]["color"] for label in labels]

            fig = Figure(figsize=(5, 4), dpi=100, facecolor=self.theme_manager.get_color("chart_bg"))
            fig.subplots_adjust(left=0.1, right=0.9, top=0.9, bottom=0.1)
            ax = fig.add_subplot(111)
            ax.set_facecolor(self.theme_manager.get_color("chart_bg"))

            if self.chart_mode.get() == "Відносно інших типів звуків":
```

```
                sizes = [stats[label] for label in labels]
                total = sum(sizes) if sum(sizes) > 0 else 1
                sizes = [s / total * 100 for s in sizes]
                if sum(sizes) > 0:
                    ax.pie(
                        sizes,
                        labels=None,
                        colors=colors,
                        autopct='%1.1f%%',
                        startangle=90,
                        wedgeprops={'width': 0.4},
                        textprops={'color': self.theme_manager.get_color("char_text")},
                        pctdistance=1.2
                    )
                ax.axis('equal')

            else:  # "Відносно тривалості аудіо файлу"
                with warnings.catch_warnings():
                    warnings.filterwarnings("ignore", category=UserWarning, message="Ignoring fixed x limits")
                    n = len(stats)
                    fig.set_size_inches(5, max(4, 2 + n * 0.5))
                    ax.set_aspect('equal', adjustable='datalim')  # Use set_aspect instead of axis('equal')
                    ax.set_xlim(-1.5, 1.5)
                    ax.set_ylim(-(1 + n * 0.25), 1 + n * 0.25)
                    ax.set_xticks([])
                    ax.set_yticks([])
                    for spine in ax.spines.values():
                        spine.set_visible(False)

                    sizes = [stats[label] for label in labels]
                    #angles = [360 / (100 / p) for p in sizes]
                    angles = [360 / (100 / p) if p != 0 else 0 for p in sizes]

                    start_angle = 90
                    for i, (angle, color) in enumerate(zip(angles, colors)):
                        end_angle = start_angle - angle
                        arc = Arc((0.0, 0.0), 2.0 + i * 0.5, 2.0 + i * 0.5, theta1=end_angle, theta2=start_angle, color=color, lw=15, zorder=10)
                        ax.add_patch(arc)
                        ax.text(-0.1, 2 - (n - i) * 0.25, f'{round(sizes[i], 1)}%', ha='right', va='center', color=self.theme_manager.get_color("char_text"), fontsize=10)

                    for i in range(n):
                        circle = plt.Circle((0, 0), 1.0 + i * 0.25, fc='none', ec=self.theme_manager.get_color("char_2_circules"), lw=1)
                        ax.add_artist(circle)

            canvas = FigureCanvasTkAgg(fig, master=self.chart_inner_frame)
            canvas.draw()
            canvas.get_tk_widget().pack(fill=ctk.BOTH, expand=True, padx=0, pady=0)
```

```
        def setup_text_analysis(self):
            self.text_result_frame = ctk.CTkFrame(self.main_frame, fg_color=self.theme_manager.get_color("block_bg"), corner_radius=10, border_width=2, border_color=self.master.theme_manager.get_color("border_color"))
            self.text_result_frame.grid(row=1, column=2, sticky="nsew", padx=5, pady=5)
            ctk.CTkLabel(self.text_result_frame, text=self.localization.get("text_analysis"), font=("Arial", 14, "bold")).pack(pady=5)

            self.text_area_frame = ctk.CTkFrame(self.text_result_frame, fg_color=self.theme_manager.get_color("inner_block_bg"), corner_radius=5)
            self.text_area_frame.pack(fill=ctk.BOTH, expand=True, padx=10, pady=5)

            self.text_area = ctk.CTkTextbox(self.text_area_frame, wrap="word", width=300, fg_color=self.theme_manager.get_color("inner_block_bg"), border_width=0)
            self.text_area.pack(side="left", fill=ctk.BOTH, expand=True)

            #scrollbar_text = ctk.CTkScrollbar(self.text_area_frame, orientation="vertical", command=self.text_area.yview)
            #scrollbar_text.pack(side="right", fill="y")
            #self.text_area.configure(yscrollcommand=scrollbar_text.set)



        def update_text_analysis(self):
            self.text_area.delete("0.0", "end")
            text = f"Audio file: {self.segment_manager.data['file_name']}\n"
            text += f"Total duration of the audio file: {self.format_time(self.segment_manager.data['duration_sec'])}\n"
            text += f"Time for analysis: {self.segment_manager.data['analysis_time_sec']:.2f} сек\n\n"

            total_duration = self.segment_manager.data['duration_sec']
            for type_name, type_data in self.segment_manager.data["types"].items():
                segments = type_data["segments"]
                if segments:
                    text += f"{type_name.capitalize()}:\n"
                    total_type_duration = 0
                    for seg in segments:
                        segment_duration = seg["end"] - seg["start"]
                        total_type_duration += segment_duration
                        text += f"  {seg['name']}: {seg['start']:.2f}s - {seg['end']:.2f}s ({segment_duration:.2f} сек)\n"
                    percentage = (total_type_duration / total_duration) * 100 if total_duration > 0 else 0
                    text += f"  Total duration: {total_type_duration:.2f} сек\n"
                    text += f"  Percentage of total audio file duration: {percentage:.1f}%\n\n"

            # Додавання підсумків із заокругленими відсотками
            text += "Results:\n"
            for type_name, value in self.segment_manager.data["stats"].items():
                rounded_percentage = round(value, 1)
                text += f"{type_name.capitalize()}: {rounded_percentage}%\n"
```

```
            self.text_area.insert("0.0", text)



        def setup_functions(self):
            self.functions_frame = ctk.CTkFrame(self.main_frame, fg_color=self.theme_manager.get_color("block_bg"), corner_radius=10, border_width=2, border_color=self.theme_manager.get_color("border_color"))
            self.functions_frame.grid(row=1, column=3, sticky="nsew", padx=5, pady=5)
            ctk.CTkLabel(self.functions_frame, text=self.localization.get("functions"), font=("Arial", 14, "bold")).pack(pady=5)

            # Function selection dropdown
            self.function_var = ctk.StringVar(value="Оберіть функцію")
            self.function_dropdown = ctk.CTkComboBox(
                self.functions_frame,
                values=["Оберіть функцію", "Аналіз - розмітка звуків", "Видалити все, крім мовлення",
                        "Видалення проміжків тиші вказаної довжини", "Шумозаглушення", "Покращення звуку",
                        "Транскрибування мовлення в текст"],
                variable=self.function_var,
                command=self.on_function_select,
                width=300
            )
            self.function_dropdown.pack(pady=5, padx=10)

            # Parameters frame with scrollbar
            self.params_container = ctk.CTkFrame(self.functions_frame, fg_color=self.theme_manager.get_color("block_bg"), corner_radius=5)
            self.params_container.pack(fill=ctk.BOTH, expand=True, padx=10, pady=5)

            self.canvas = ctk.CTkCanvas(self.params_container, highlightthickness=0, bg=self.theme_manager.get_color("block_bg"))
            self.scrollbar = ctk.CTkScrollbar(self.params_container, orientation="vertical", command=self.canvas.yview)
            self.params_frame = ctk.CTkFrame(self.canvas, fg_color=self.theme_manager.get_color("block_bg"), width=280)

            self.params_frame.bind("<Configure>", lambda e: self.canvas.configure(scrollregion=self.canvas.bbox("all")))
            self.canvas.create_window((0, 0), window=self.params_frame, anchor="nw")
            self.canvas.configure(yscrollcommand=self.scrollbar.set)

            self.canvas.pack(side="left", fill="both", expand=True)
            self.scrollbar.pack(side="right", fill="y")

            # Execute button
            self.btn_execute = ctk.CTkButton(
                self.functions_frame,
```

```
            text="Виконати функцію",
            font=("Arial", 12),
            fg_color=self.theme_manager.get_color("button_fg"),
            hover_color=self.theme_manager.get_color("button_hover"),
            command=self.execute_function,
            width=300
        )
        self.btn_execute.pack(pady=10)

        # Initialize parameters
        self.clear_params()
        self.on_function_select("Оберіть функцію")  # Initial setup

    def on_function_select(self, selection):
        """Handle function selection and update parameters frame."""
        self.clear_params()
        self.function_var.set(selection)

        if selection == "Оберіть функцію":
            return

        if selection == "Аналіз - розмітка звуків":
            self.noise_suppression_var = ctk.BooleanVar(value=False)
            self.enhance_audio_var = ctk.BooleanVar(value=False)
            ctk.CTkCheckBox(self.params_frame, text="Виконати шумозаглушення перед аналізом", variable=self.noise_suppression_var).pack(anchor="w", padx=5, pady=5)
            ctk.CTkCheckBox(self.params_frame, text="Виконати покращення звуку перед аналізом", variable=self.enhance_audio_var).pack(anchor="w", padx=5, pady=5)
            ctk.CTkButton(self.params_frame, text="Глибокі налаштування", command=lambda: AnalysisSettingsPopup(self)).pack(pady=10)

        elif selection == "Видалити все, крім мовлення":
            self.output_folder_var = ctk.StringVar(value="Не вибрано")
            self.overwrite_original_var = ctk.BooleanVar(value=False)
            self.silence_start_var = ctk.StringVar(value="0")
            self.silence_between_var = ctk.StringVar(value="0")

            folder_frame = ctk.CTkFrame(self.params_frame)
            folder_frame.pack(fill="x", padx=5, pady=5)
            ctk.CTkLabel(folder_frame, text="Папка для збереження:").pack(anchor="w")
            ctk.CTkLabel(folder_frame, textvariable=self.output_folder_var, wraplength=260, justify="left").pack(anchor="w", pady=2)
            ctk.CTkButton(folder_frame, text="Вибрати", command=self.choose_output_folder).pack(anchor="w", pady=2)

            ctk.CTkLabel(self.params_frame, text="Додати проміжки тиші", font=("Arial", 12, "bold")).pack(anchor="w", padx=5, pady=5)
            start_frame = ctk.CTkFrame(self.params_frame)
            start_frame.pack(fill="x", padx=5, pady=2)
            ctk.CTkLabel(start_frame, text="На початку:").pack(side="left")
            ctk.CTkEntry(start_frame, textvariable=self.silence_start_var, width=60).pack(side="left", padx=5)
```

```
                ctk.CTkLabel(start_frame, text="сек.").pack(side="left")
                between_frame = ctk.CTkFrame(self.params_frame)
                between_frame.pack(fill="x", padx=5, pady=2)
                ctk.CTkLabel(between_frame, text="Між сегментами:").pack(side="left")
                ctk.CTkEntry(between_frame, textvariable=self.silence_between_var, width=60).pack(side="left", padx=5)
                ctk.CTkLabel(between_frame, text="сек.").pack(side="left")
                ctk.CTkCheckBox(self.params_frame, text="Перезаписати оригінал", variable=self.overwrite_original_var).pack(anchor="w", padx=5, pady=5)

            elif selection == "Видалення проміжків тиші":
                self.output_folder_var = ctk.StringVar(value="Не вибрано")
                self.overwrite_original_var = ctk.BooleanVar(value=False)
                self.silence_option_var = ctk.StringVar(value="Видалити всі проміжки тиші")
                self.silence_duration_var = ctk.StringVar(value="0")
                self.selected_silence_segments = {}

                folder_frame = ctk.CTkFrame(self.params_frame)
                folder_frame.pack(fill="x", padx=5, pady=5)
                ctk.CTkLabel(folder_frame, text="Папка для збереження:").pack(anchor="w")
                ctk.CTkLabel(folder_frame, textvariable=self.output_folder_var, wraplength=260, justify="left").pack(anchor="w", pady=2)
                ctk.CTkButton(folder_frame, text="Вибрати", command=self.choose_output_folder).pack(anchor="w", pady=2)

                combo_width = self.params_frame.winfo_width() - 5 if self.params_frame.winfo_width() else 260  # 10px відступи з кожного боку
                self.silence_combo = ctk.CTkComboBox(
                    self.params_frame,
                    values=["Видалити всі проміжки тиші", "Видалити проміжки тиші довші за", "Видалити конкретні проміжки тиші"],
                    variable=self.silence_option_var,
                    command=self.update_silence_options,
                    width=combo_width,
                    state="readonly"  # Додано, щоб текст не обрізався
                )
                self.silence_combo.pack(pady=5, padx=5)

                self.silence_dynamic_frame = ctk.CTkFrame(self.params_frame)
                self.silence_dynamic_frame.pack(fill="x", padx=5, pady=5)
                self.update_silence_options()

                ctk.CTkCheckBox(self.params_frame, text="Перезаписати оригінал", variable=self.overwrite_original_var).pack(anchor="w", padx=5, pady=5)

            elif selection in ["Шумозаглушення", "Покращення звуку"]:
                self.output_folder_var = ctk.StringVar(value="Не вибрано")
                self.overwrite_original_var = ctk.BooleanVar(value=False)

                folder_frame = ctk.CTkFrame(self.params_frame)
                folder_frame.pack(fill="x", padx=5, pady=5)
                ctk.CTkLabel(folder_frame, text="Папка для збереження:").pack(anchor="w")
```

```
            ctk.CTkLabel(folder_frame, textvariable=self.output_folder_var, wraplength=260, justify="left").pack(anchor="w", pady=2)
            ctk.CTkButton(folder_frame, text="Вибрати", command=self.choose_output_folder).pack(anchor="w", pady=2)

            ctk.CTkCheckBox(self.params_frame, text="Перезаписати оригінал", variable=self.overwrite_original_var).pack(anchor="w", padx=5, pady=5)

        elif selection == "Транскрибування мовлення в текст":
            self.output_folder_var = ctk.StringVar(value="Не вибрано")
            self.noise_suppression_var = ctk.BooleanVar(value=False)
            self.enhance_audio_var = ctk.BooleanVar(value=False)
            self.language_var = ctk.StringVar(value="uk")
            self.prompt_var = ctk.StringVar(value="")

            folder_frame = ctk.CTkFrame(self.params_frame)
            folder_frame.pack(fill="x", padx=5, pady=5)
            ctk.CTkLabel(folder_frame, text="Папка для збереження (.srt):").pack(anchor="w")
            ctk.CTkLabel(folder_frame, textvariable=self.output_folder_var, wraplength=260, justify="left").pack(anchor="w", pady=2)
            ctk.CTkButton(folder_frame, text="Вибрати", command=self.choose_output_folder).pack(anchor="w", pady=2)

            ctk.CTkCheckBox(self.params_frame, text="Поперелньо виконати шумозаглушення", variable=self.noise_suppression_var).pack(anchor="w", padx=5, pady=5)
            ctk.CTkCheckBox(self.params_frame, text="Поперелньо виконати покращення звуку", variable=self.enhance_audio_var).pack(anchor="w", padx=5, pady=5)

            language_frame = ctk.CTkFrame(self.params_frame)
            language_frame.pack(fill="x", padx=5, pady=5)
            ctk.CTkLabel(language_frame, text="Мова транскрибування:").pack(anchor="w")
            ctk.CTkEntry(language_frame, textvariable=self.language_var, width=60).pack(anchor="w", padx=5)

            prompt_frame = ctk.CTkFrame(self.params_frame)
            prompt_frame.pack(fill="x", padx=5, pady=5)
            ctk.CTkLabel(prompt_frame, text="Промт для контексту:").pack(anchor="w")
            prompt_text = ctk.CTkTextbox(prompt_frame, width=200, height=60, wrap="word")
            prompt_text.pack(anchor="w", padx=5, pady=2)
            prompt_text.bind("<KeyRelease>", lambda e: self.prompt_var.set(prompt_text.get("1.0", "end-1c")))

    def clear_params(self):
        """Safely clear all widgets in the parameters frame."""
        for widget in list(self.params_frame.winfo_children()):
            try:
                widget.destroy()
            except AttributeError:
                continue
```

```
        def choose_output_folder(self):
            folder = filedialog.askdirectory()
            if folder:
                if hasattr(self, 'output_folder_var'):
                    self.output_folder_var.set(folder)

        def update_silence_options(self, event=None):
            for widget in self.silence_dynamic_frame.winfo_children():
                widget.destroy()

            option = self.silence_option_var.get()
            if option == "Видалити проміжки тиші довші за":
                duration_frame = ctk.CTkFrame(self.silence_dynamic_frame)
                duration_frame.pack(fill="x")
                ctk.CTkLabel(duration_frame, text="Довжина (сек.):").pack(side="left")
                ctk.CTkEntry(duration_frame, textvariable=self.silence_duration_var, width=60).pack(side="left", padx=5)
                ctk.CTkLabel(duration_frame, text="сек.").pack(side="left")
            elif option == "Видалити конкретні проміжки тиші":
                silence_segments = self.segment_manager.data["types"].get("silence", {}).get("segments", [])
                if not silence_segments:
                    ctk.CTkLabel(self.silence_dynamic_frame, text="Сегментів звуку тиші немає.").pack(anchor="w", padx=5, pady=5)
                    return
                for i, seg in enumerate(silence_segments, 1):
                    var = ctk.BooleanVar(value=False)
                    self.selected_silence_segments[i] = var
                    ctk.CTkCheckBox(self.silence_dynamic_frame, text=f"{i}. {seg['name']} ({seg['start']:.2f}s - {seg['end']:.2f}s)", variable=var).pack(anchor="w", padx=5, pady=2)

        def execute_function(self):
            function = self.function_var.get()
            if function == "Оберіть функцію":
                CTkMessagebox(title="Помилка", message="Оберіть функцію з випадаючого списку!", icon="warning")
                return

            if function in ["Видалити все, крім мовлення", "Видалення проміжків тиші вказаної довжини", "Шумозаглушення", "Покращення звуку"]:
                if self.output_folder_var.get() == "Не вибрано" and not self.overwrite_original_var.get():
                    CTkMessagebox(title="Помилка", message="Оберіть папку для збереження результату або увімкніть перезапис оригіналу!", icon="warning")
                    return
                if self.overwrite_original_var.get():
                    confirm = CTkMessagebox(title="Підтвердження", message=f"Функція '{function}' буде виконана над оригінальним файлом, це змінить його. Ви впевнені?", icon="warning", option_1="Так", option_2="Ні")
                    if confirm.get() != "Так":
                        return

            if function == "Аналіз - розмітка звуків":
```

```
            tasks = [("Аналіз", "analyze", 100)]
            if self.noise_suppression_var.get():
                tasks.insert(0, ("Шумозаглушення", "denoise", 30))
            if self.enhance_audio_var.get():
                tasks.insert(0, ("Покращення звуку", "enhance", 30))
            self.process_function(tasks)

        elif function == "Видалити все, крім мовлення":
            tasks = [("Обробка", "keep_speech", 100)]
            self.process_function(tasks, {
                "output_folder": self.output_folder_var.get(),
                "overwrite_original": self.overwrite_original_var.get(),
                "silence_start": float(self.silence_start_var.get()),
                "silence_between": float(self.silence_between_var.get())
            })

        elif function == "Видалення проміжків тиші вказаної довжини":
            tasks = [("Обробка", "remove_silence", 100)]
            silence_params = {}
            if self.silence_option_var.get() == "Видалити проміжки тиші довші за":
                silence_params["duration"] = float(self.silence_duration_var.get())
            elif self.silence_option_var.get() == "Видалити конкретні проміжки тиші":
                silence_segments = self.segment_manager.data["types"].get("silence",
{}).get("segments", [])
                silence_params["segments"] = [i for i, var in self.selected_silence_segments.items() if
var.get()]
            self.process_function(tasks, {
                "output_folder": self.output_folder_var.get(),
                "overwrite_original": self.overwrite_original_var.get(),
                "silence_option": self.silence_option_var.get(),
                **silence_params
            })

        elif function == "Шумозаглушення":
            tasks = [("Обробка", "denoise", 100)]
            self.process_function(tasks, {
                "output_folder": self.output_folder_var.get(),
                "overwrite_original": self.overwrite_original_var.get()
            })

        elif function == "Покращення звуку":
            tasks = [("Обробка", "enhance", 100)]
            self.process_function(tasks, {
                "output_folder": self.output_folder_var.get(),
                "overwrite_original": self.overwrite_original_var.get()
            })

        elif function == "Транскрибування мовлення в текст":
            if not self.validate_transcribe_params():
                return
            if self.output_folder_var.get() == "Не вибрано" and not self.overwrite_original_var.get():
```

```
                CTkMessagebox(title="Помилка", message="Оберіть папку для збереження результату або увімкніть перезапис оригіналу!", icon="warning")
                return
            tasks = [("Транскрибування", "transcribe", 100)]
            if self.noise_suppression_var.get():
                tasks.insert(0, ("Шумозаглушення", "denoise", 30))
            if self.enhance_audio_var.get():
                tasks.insert(0, ("Покращення звуку", "enhance", 30))
            self.process_function(tasks, {
                "output_folder": self.output_folder_var.get(),
                "noise_suppression": self.noise_suppression_var.get(),
                "enhance_audio": self.enhance_audio_var.get(),
                "language": self.language_var.get(),
                "prompt": self.prompt_var.get()
            })

        def validate_transcribe_params(self):
            """Validate transcription parameters."""
            language = self.language_var.get().strip().lower()
            if language and language not in whisper.tokenizer.LANGUAGES.keys():
                CTkMessagebox(title="Помилка", message=f"Мова '{language}' не підтримується Whisper.", icon="cancel")
                return False
            prompt = self.prompt_var.get().strip()
            if prompt and len(prompt) > 500:
                CTkMessagebox(title="Помилка", message="Промт занадто довгий (максимум 500 символів).", icon="cancel")
                return False
            return True

        def on_function_update(self):
            self.update_callback()


        def populate_segments(self):
            self.sound_segments_panel.sound_types_data = self.segment_manager.data["types"]
            self.sound_segments_panel.populate_sound_types()

        def on_update_segments(self):
            self.segment_manager.data["types"] = self.sound_segments_panel.sound_types_data
            self.segment_manager.update_stats()
            self.segment_manager.save_to_json()
            self.update_chart()
            self.update_text_analysis()
            self.update_waveform_segments()  # Оновлюємо хвилю при зміні сегментів


        def play_audio(self):
            """Відтворення аудіо в окремому потоці"""
            def callback(outdata, frames, time, status):
                if status:
                    print(status)
```

```
            start = self.current_frame
            end = start + frames
            if end > len(self.audio_data):
                end = len(self.audio_data)
                outdata[:end-start] = self.audio_data[start:end]
                outdata[end-start:] = 0
                self.is_playing = False
                self.paused = False
                self.btn_play_pause.configure(text="⏯")
                raise sd.CallbackStop()
            outdata[:] = self.audio_data[start:end]
            self.current_frame = end

        try:
            if self.stream is not None:
                self.stream.stop()
                self.stream.close()
            self.stream = sd.OutputStream(
                samplerate=self.sample_rate,
                channels=1,
                callback=callback,
                blocksize=2048
            )
            self.stream.start()
        except Exception as e:
            print(f"Помилка відтворення: {e}")
            self.is_playing = False
            self.paused = False
            self.btn_play_pause.configure(text="⏯")

    def toggle_play_pause(self):
        if not self.is_playing:
            if self.paused:
                if self.current_position > 1:
                    self.current_position = max(0, self.current_position - 1)
                self.current_frame = int(self.current_position * self.sample_rate)
                self.play_audio()
                self.start_time = time.time() - self.current_position
            else:
                self.current_frame = 0
                self.current_position = 0
                self.play_audio()
                self.start_time = time.time()
            self.is_playing = True
            self.paused = False
            self.btn_play_pause.configure(text="⏸")
        else:
            self.current_position = time.time() - self.start_time
            self.current_frame = int(self.current_position * self.sample_rate)
            if self.stream is not None:
                self.stream.stop()
```

```
            self.stream.close()
        self.is_playing = False
        self.paused = True
        self.btn_play_pause.configure(text="⏯")

    def rewind_5s(self):
        self.current_position = max(0, self.current_position - 5)
        self.current_frame = int(self.current_position * self.sample_rate)
        self.progress_bar.set(self.current_position)
        self.update_waveform_line()
        self.update_time_label()
        if self.is_playing:
            if self.stream is not None:
                self.stream.stop()
                self.stream.close()
            self.play_audio()
            self.start_time = time.time() - self.current_position

    def forward_5s(self):
        self.current_position = min(self.duration, self.current_position + 5)
        self.current_frame = int(self.current_position * self.sample_rate)
        self.progress_bar.set(self.current_position)
        self.update_waveform_line()
        self.update_time_label()
        if self.is_playing:
            if self.stream is not None:
                self.stream.stop()
                self.stream.close()
            self.play_audio()
            self.start_time = time.time() - self.current_position


    def on_slider_move(self, value):
        if not self.is_slider_dragging:
            self.is_slider_dragging = True
        self.current_position = float(value)
        self.current_frame = int(self.current_position * self.sample_rate)
        self.update_waveform_line()
        self.update_time_label()

    def slider_released(self, event):
        self.current_frame = int(self.current_position * self.sample_rate)
        if self.is_playing:
            if self.stream is not None:
                self.stream.stop()
                self.stream.close()
            self.play_audio()
            self.start_time = time.time() - self.current_position
        self.is_slider_dragging = False  # Скидаємо прапорець після відпускання

    def update_progress(self):
```

```
            #print(f"update_progress:                                    is_playing={self.is_playing},
is_slider_dragging={self.is_slider_dragging}")
            if self.is_playing and not self.is_slider_dragging:  # Спрощуємо умову
                self.current_position = time.time() - self.start_time
                if self.current_position >= self.duration:
                    self.current_position = self.duration
                    self.is_playing = False
                    self.paused = False
                    self.btn_play_pause.configure(text="⏯")
                    if self.stream is not None:
                        self.stream.stop()
                        self.stream.close()
                self.progress_bar.set(self.current_position)
                self.update_waveform_line()
                self.update_time_label()
            self.after_id = self.after(100, self.update_progress)

        def update_waveform_line(self):
            #print(f"update_waveform_line: vertical_line={self.vertical_line}")
            if self.is_updating_waveform:
                return
            self.is_updating_waveform = True
            if self.vertical_line is not None:
                try:
                    self.vertical_line.remove()
                except ValueError as e:
                    print(f"Помилка видалення vertical_line: {e}")
            self.vertical_line = self.ax.axvline(x=self.current_position, color='red', linestyle='--',
linewidth=1)
            self.canvas.draw()
            self.is_updating_waveform = False

        def update_time_label(self):
            self.time_label_left.configure(text=self.format_time(self.current_position))

        def format_time(self, seconds):
            hours = int(seconds // 3600)
            minutes = int((seconds % 3600) // 60)
            secs = int(seconds % 60)
            return f"{hours:02}:{minutes:02}:{secs:02}"


        def process_function(self, tasks, params=None):
            progress_popup = ProgressPopup(self, tasks=tasks)
            progress_popup.grab_set()

            try:
                current_file = self.master.prepared_file
                output_file = None

                for task_name, task_type, weight in tasks:
                    progress_popup.update_progress(task_name, 0)
```

```
                if task_type == "analyze":
                    if params.get("noise_suppression", False):
                        current_file = reduce_audio_noise_deepfilternet(current_file, "output")
                    if params.get("enhance_audio", False):
                        current_file = enhance_speech(current_file, "output")
                    speech, singing, music, silence = classify_audio(current_file)
                    self.segment_manager.update_segments(current_file, speech, singing, music, silence,
0)
                elif task_type == "keep_speech":
                    output_path = os.path.join(params["output_folder"],
f"{os.path.basename(current_file).rsplit('.', 1)[0]}_speech.wav") if not params["overwrite_original"] else
current_file
                    current_file = keep_only_speech(current_file, output_path,
start_silence=params["initial_silence"], inter_silence=params["inter_silence"])
                    output_file = current_file
                elif task_type == "remove_silence":
                    output_path = os.path.join(params["output_folder"],
f"{os.path.basename(current_file).rsplit('.', 1)[0]}_no_silence.wav") if not params["overwrite_original"]
else current_file
                    if params["silence_option"] == "Видалити всі проміжки тиші":
                        current_file = remove_silence(current_file, output_path,
silence_intervals=self.segment_manager.data["types"]["silence"]["segments"])
                    elif params["silence_option"] == "Видалити проміжки тиші довші за":
                        current_file = remove_silence(current_file, output_path,
silence_intervals=self.segment_manager.data["types"]["silence"]["segments"], mode="longer_than",
threshold=params["duration"])
                    elif params["silence_option"] == "Видалити конкретні проміжки тиші":
                        silence_segments = self.segment_manager.data["types"]["silence"]["segments"]
                        segments_to_remove = [(seg["start"], seg["end"]) for i, seg in
enumerate(silence_segments) if i in params["segments"]]
                        current_file = remove_silence(current_file, output_path,
silence_intervals=segments_to_remove, mode="selected")
                    output_file = current_file
                elif task_type == "denoise":
                    output_path = os.path.join(params["output_folder"],
f"{os.path.basename(current_file).rsplit('.', 1)[0]}_denoised.wav") if not params["overwrite_original"] else
current_file
                    current_file = reduce_audio_noise_deepfilternet(current_file, output_path)
                    output_file = current_file
                elif task_type == "enhance":
                    output_path = os.path.join(params["output_folder"],
f"{os.path.basename(current_file).rsplit('.', 1)[0]}_enhanced.wav") if not params["overwrite_original"]
else current_file
                    current_file = enhance_speech(current_file, output_path)
                    output_file = current_file
                elif task_type == "transcribe":
                    output_path = os.path.join(params["output_folder"],
f"{os.path.basename(current_file).rsplit('.', 1)[0]}_transcription.srt")
                    if params["noise_suppression"]:
                        current_file = reduce_audio_noise_deepfilternet(current_file, "output")
                    if params["enhance_audio"]:
                        current_file = enhance_speech(current_file, "output")
```

```
                    speech_intervals = self.segment_manager.get_speech_intervals()
                    transcribe_audio(current_file, output_path, speech_intervals,
language=params["language"], prompt=params["prompt"])
                    output_file = output_path
                if progress_popup.cancelled:
                    break
                progress_popup.update_progress(task_name, weight)
            else:
                if output_file and function != "Аналіз - розмітка звуків":
                    if current_file != self.master.prepared_file or function == "Транскрибування":
                        self.master.prepared_file = current_file
                        if function != "Транскрибування":
                            self.audio_data, self.sample_rate = sf.read(current_file)
                            if self.audio_data.ndim > 1:
                                self.audio_data = self.audio_data[:, 0]
                            self.audio_data = self.audio_data[:, np.newaxis]
                            self.duration = len(self.audio_data) / self.sample_rate
                        if self.stream is not None:
                            self.stream.stop()
                            self.stream.close()
                        self.current_frame = 0
                        self.current_position = 0
                        self.update_waveform()
                        self.populate_segments()
                        self.update_chart()
                        self.update_text_analysis()
                    ConfirmFileOpenPopup(self, output_file, self.open_new_file)
            except Exception as e:
                CTkMessagebox(title="Помилка", message=f"Помилка виконання: {e}", icon="cancel")
            finally:
                self.after(100, progress_popup.destroy)


        def open_new_file(self, file_path):
            from gui.main_gui import UIState
            self.master.current_file = file_path
            self.master.prepared_file = prepare_audio(file_path, "output")
            self.audio_data, self.sample_rate = sf.read(self.master.prepared_file)
            if self.audio_data.ndim > 1:
                self.audio_data = self.audio_data[:, 0]  # Mono
            self.audio_data = self.audio_data[:, np.newaxis]
            self.duration = len(self.audio_data) / self.sample_rate
            self.master.set_ui_state(UIState.SELECT_ACTION)

        def delete_sound_type(self, widget):
            name = widget.sound_type_data["name"]
            del self.sound_segments_panel.sound_types_data[name]
            self.sound_segments_panel.populate_sound_types()
            self.on_update_segments()

        def delete_segment(self, seg_num):
            for type_name, type_data in self.sound_segments_panel.sound_types_data.items():
```

```
            if seg_num <= len(type_data["segments"]):
                type_data["segments"].pop(seg_num - 1)
                self.sound_segments_panel.populate_sound_types()
                break
        self.on_update_segments()

    def update_waveform(self):
        self.plot_data.set_ydata(self.signal)
        self.ax.relim()
        self.ax.autoscale_view()
        self.canvas.draw()

    @property
    def signal(self):
        if self.audio_data.ndim > 1:
            signal = np.mean(self.audio_data, axis=1)
        else:
            signal = self.audio_data
        return signal / np.max(np.abs(signal)) if np.max(np.abs(signal)) != 0 else signal


    def destroy(self):
        try:

            # Зупинка потоку, якщо він є
            if self.is_playing:
                if self.stream:
                    self.stream.stop()
                    self.stream.close()

            # Cancel all scheduled after callbacks
            if self.after_id:
                self.after_cancel(self.after_id)
                self.after_id = None

            # Clear chart widgets
            for widget in self.chart_inner_frame.winfo_children():
                widget.destroy()

            # Clear matplotlib figures to prevent memory leaks
            if hasattr(self, 'fig'):
                plt.close(self.fig)
            if hasattr(self, 'canvas'):
                self.canvas.get_tk_widget().destroy()

        except Exception as e:
            print(f"Помилка при знищенні: {e}")

        # Викликаємо метод destroy батьківського класу
        super().destroy()
```

Додаток Г – Ілюстративна частина

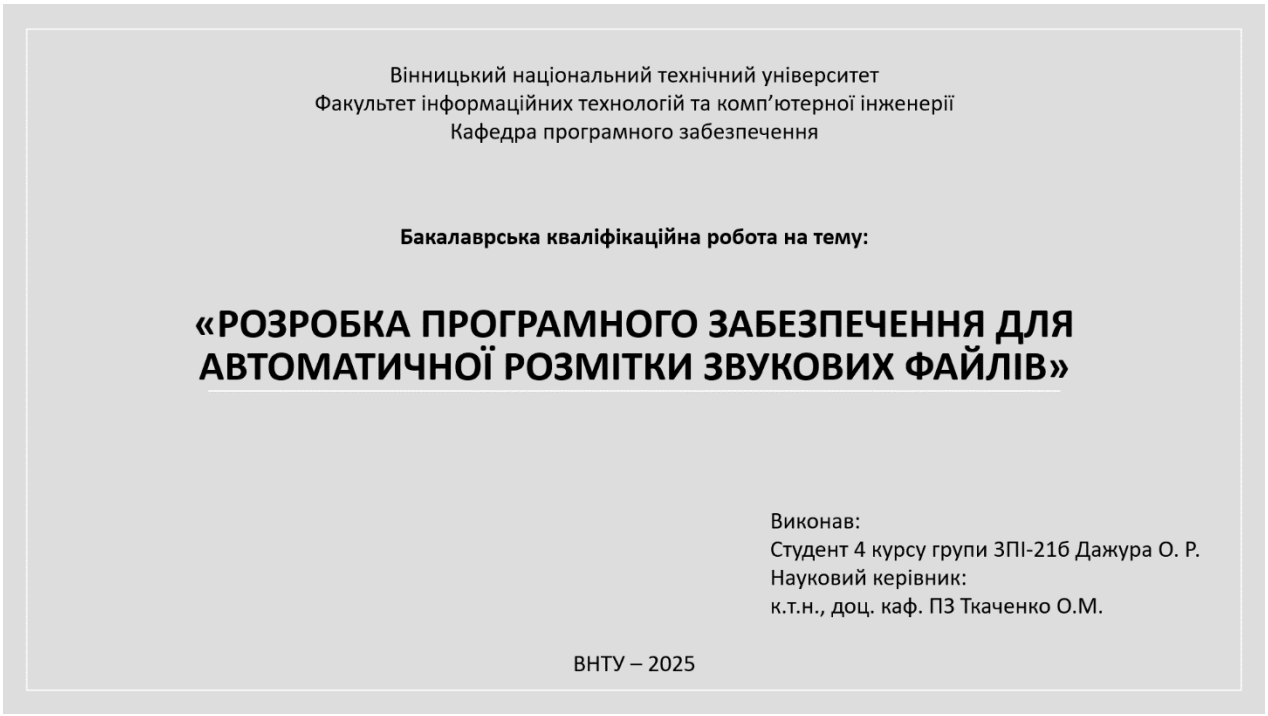

Рисунок Г.1 – Титульний слайд

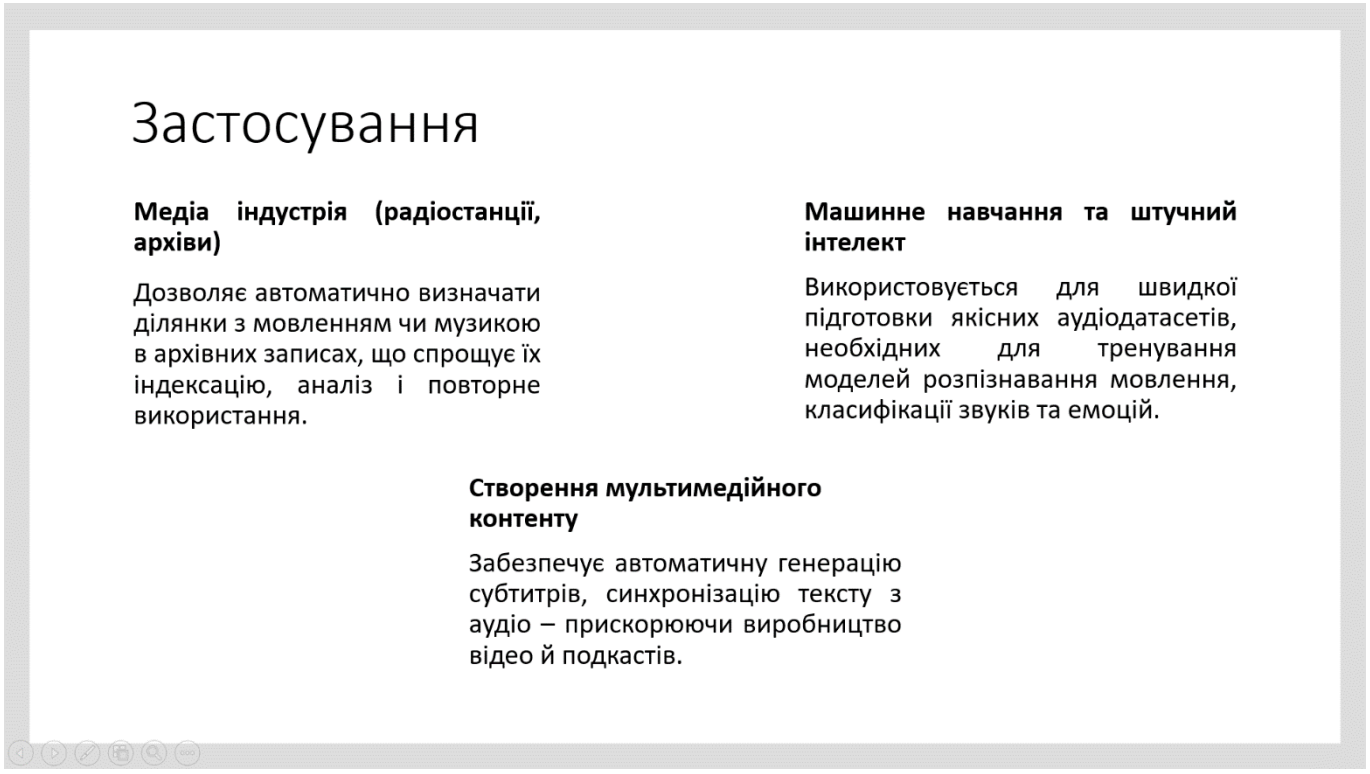

Рисунок Г.2 – Слайд застосувань

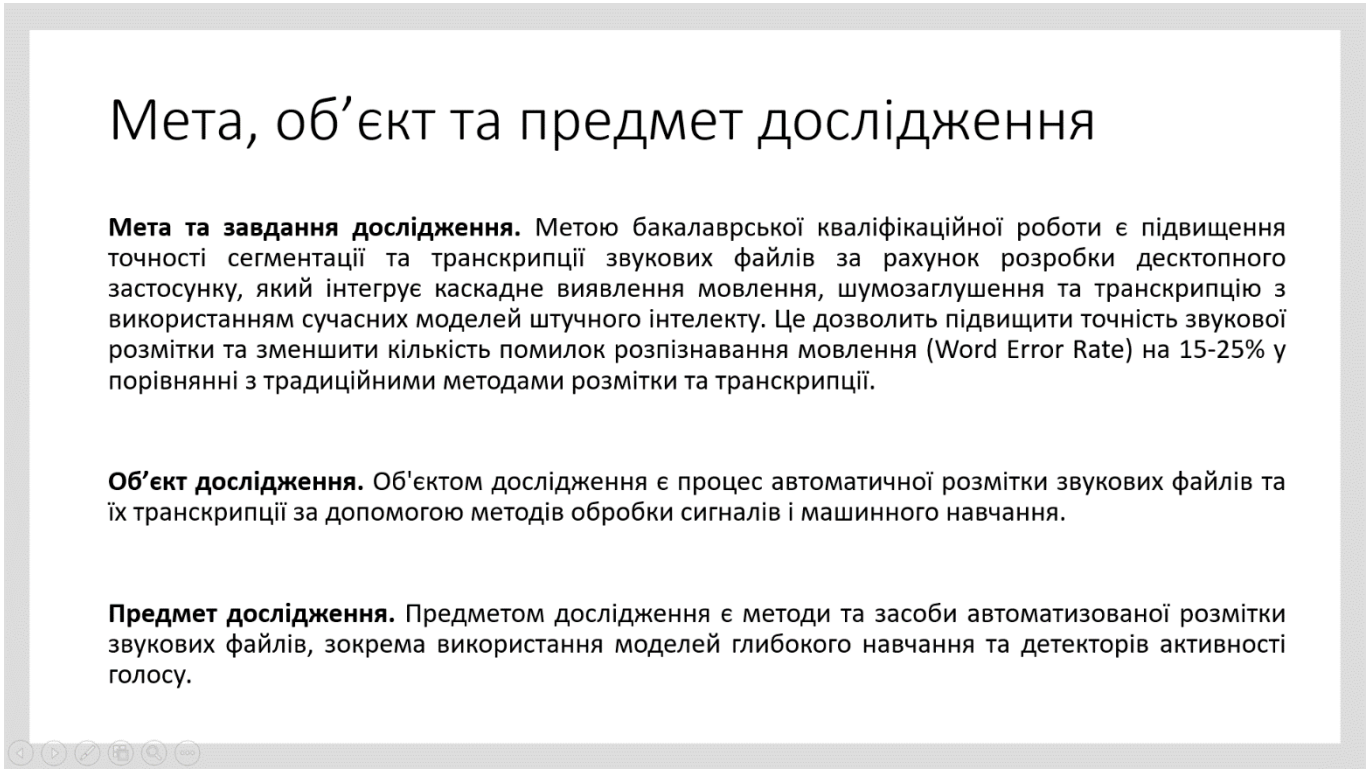

Рисунок Г.3 – Слайд мети, об’єкта та предмету дослідження

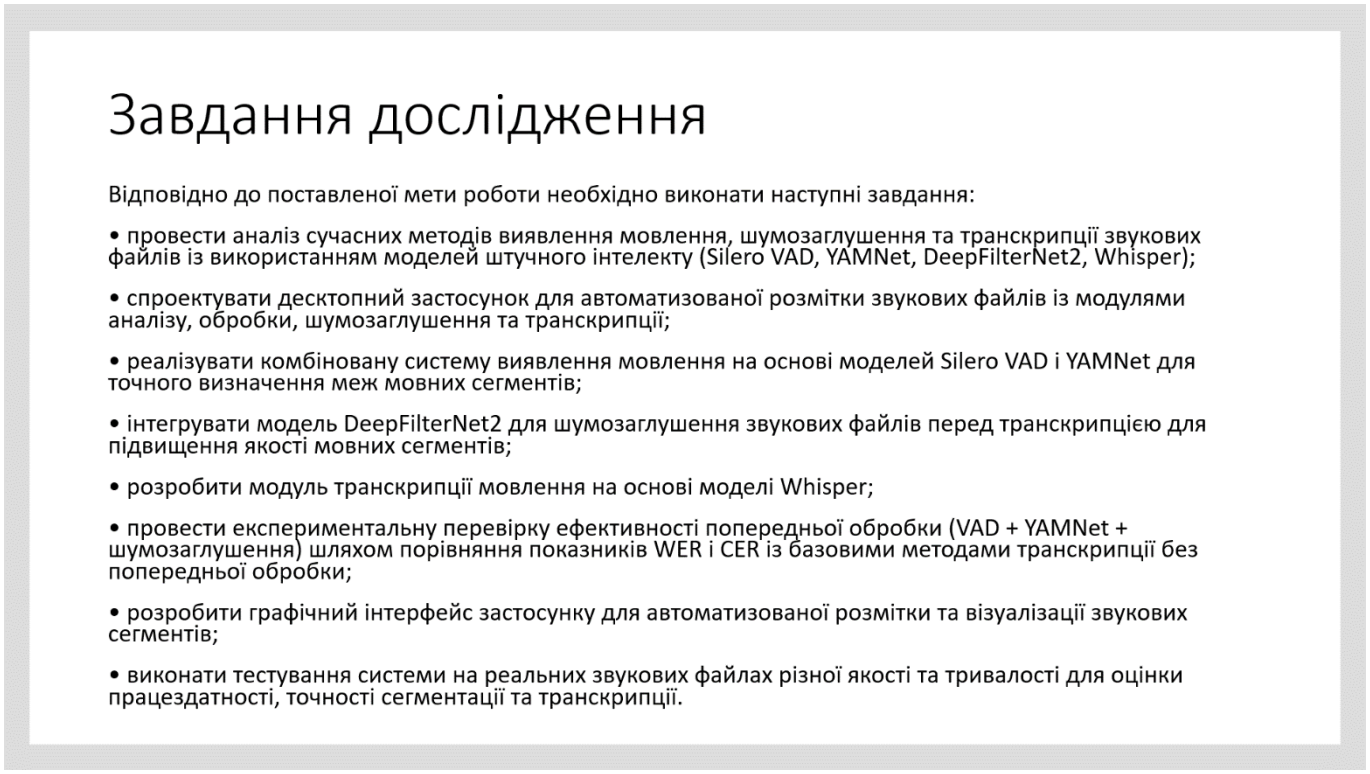

Рисунок Г.4 – Слайд завдань дослідження

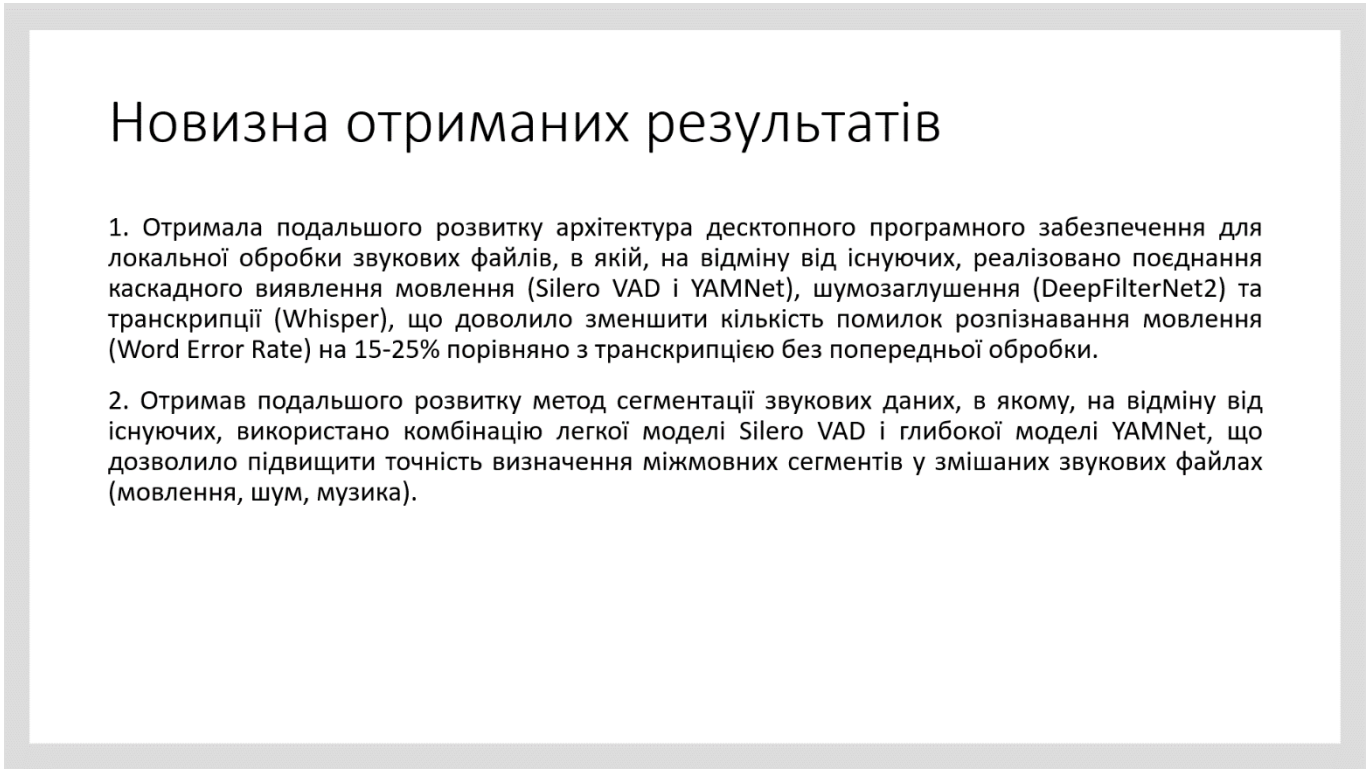

Рисунок Г.5 – Слайд новизни отриманих результатів

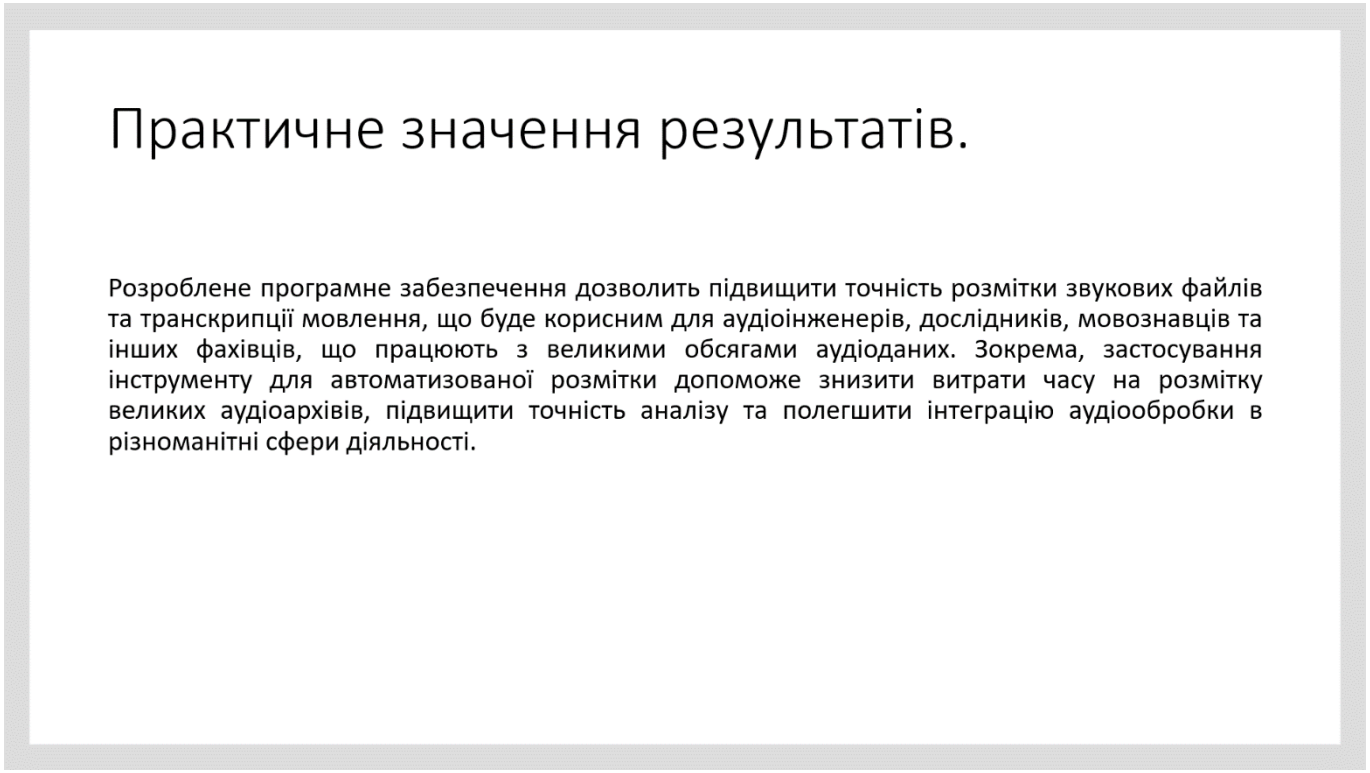

Рисунок Г.6 – Слайд практичного значення результатів

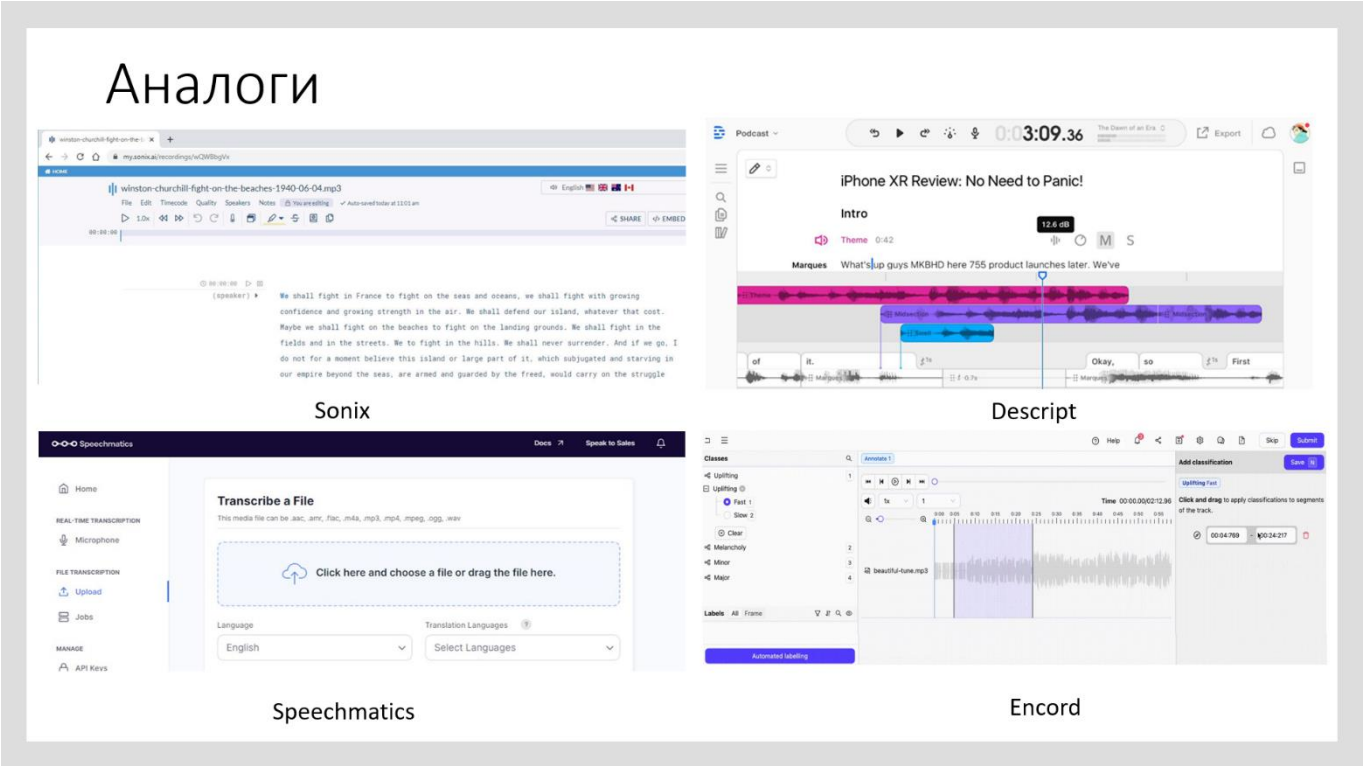

Рисунок Г.7 – Слайд аналогів

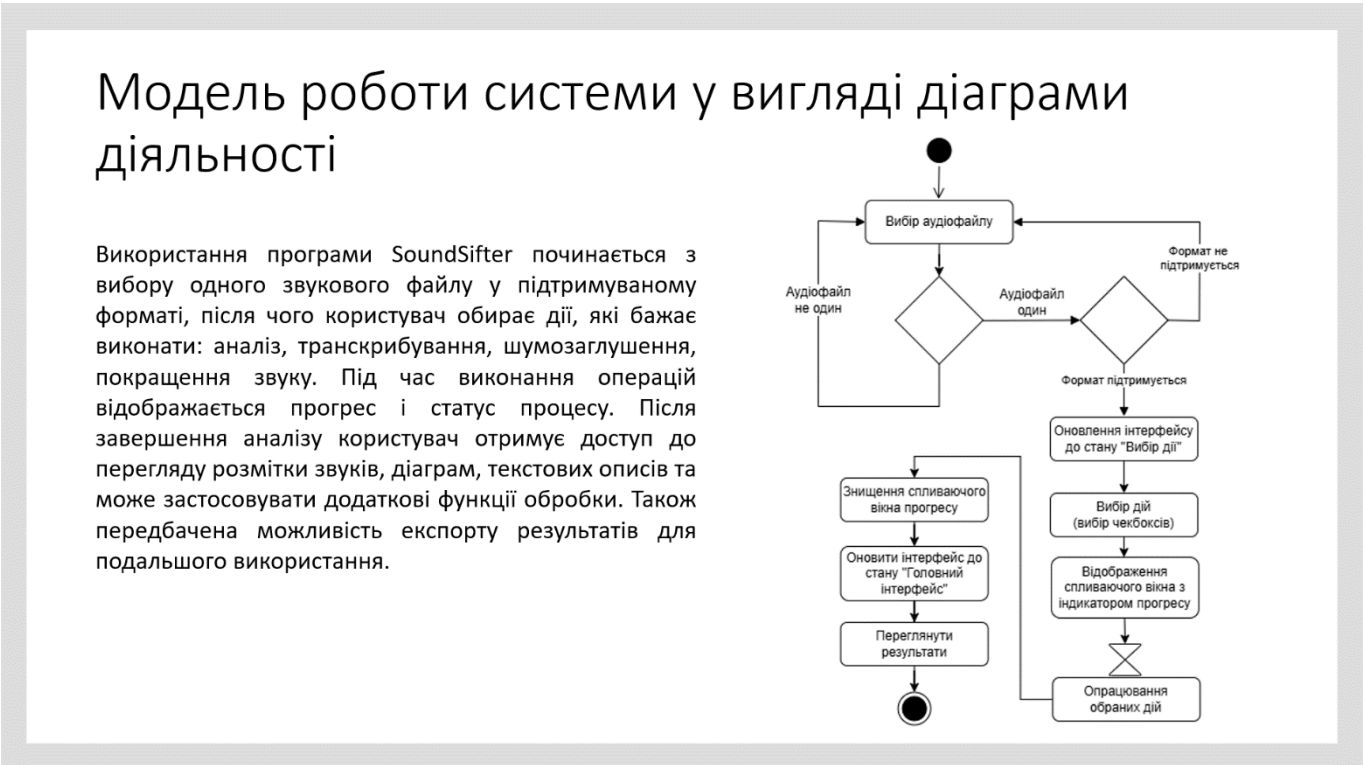

Рисунок Г.8 – Слайд моделі роботи системи

Блок схема алгоритму методу classify_audio()

Метод classify_audio() у SoundSifter виконує автоматичну класифікацію аудіо на мовлення, спів, музику та тишу, поєднуючи моделі Silero VAD та YAMNet для досягнення високої точності. Він обробляє звукові файли, нормалізує їх, визначає сегменти за ймовірнісними оцінками й фільтрує короткі або шумові ділянки. Особливістю є гнучка логіка об'єднання та класифікації сегментів із урахуванням пауз і середніх значень ймовірностей, розрізнення співу та мовлення шляхом порівняння середніх ймовірностей для кожного сегмента, визначених за допомогою моделі Silero VAD та YAMNet. Результатом є структуровані часові інтервали, придатні для подальшого використання

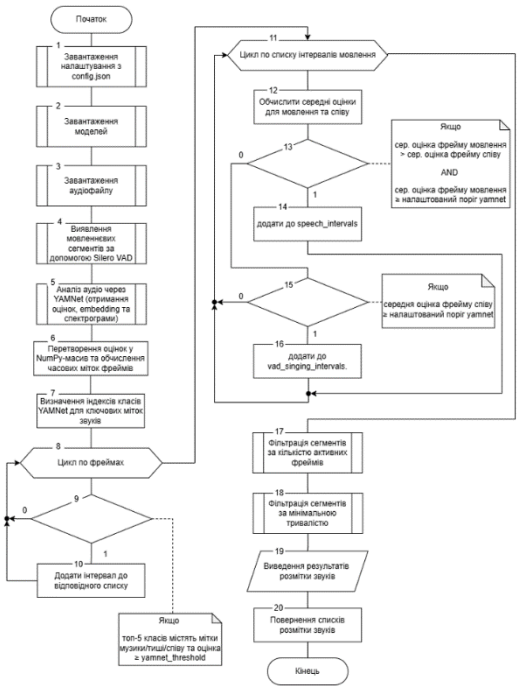

Рисунок Г.9 – Слайд блок схеми алгоритму розмітки звуків

Діаграма компонентів модулів аналізу та обробки системи

Архітектура десктопного ПЗ для локальної обробки аудіо, поєднує виявлення мовлення, шумозаглушення та транскрипцію. Особливістю є каскадна структура обробки з використанням моделей Silero VAD, YAMNet, DeepFilterNet2 та Whisper, що дозволяє зменшити кількість помилок розпізнавання мовлення (Word Error Rate) на 15–25% порівняно з транскрипцією необроблених звукових файлів. Система працює офлайн, гарантує конфіденційність даних і ефективно обробляє складні змішані аудіофайли. Це робить її придатною для різних звукових файлів.

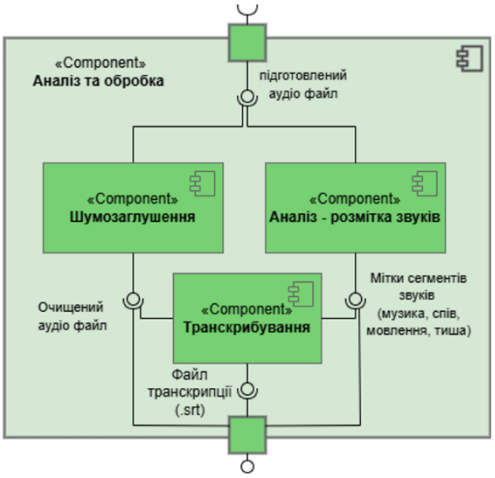

Рисунок Г.10 – Слайд діаграми компонентів модулів аналізу та обробки

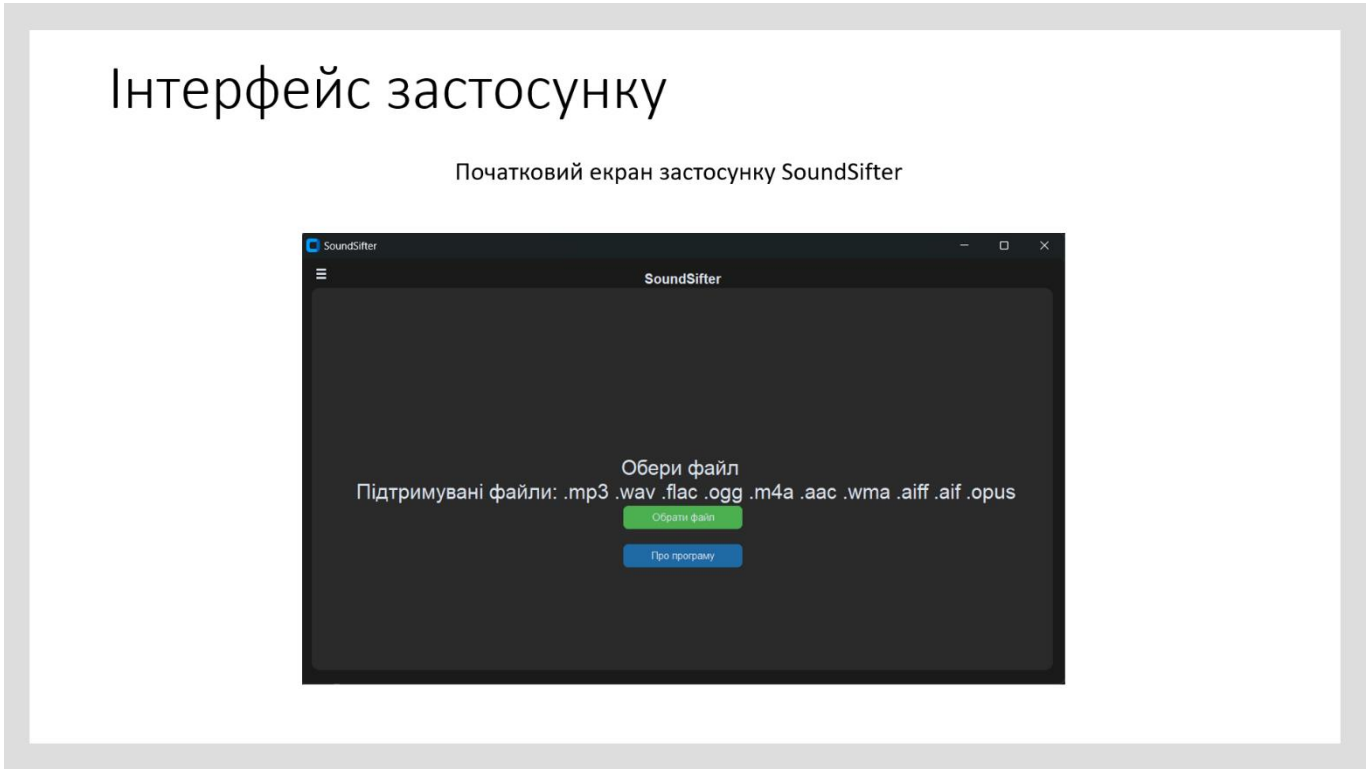

Рисунок Г.11 – Слайд початкового екрану застосунку

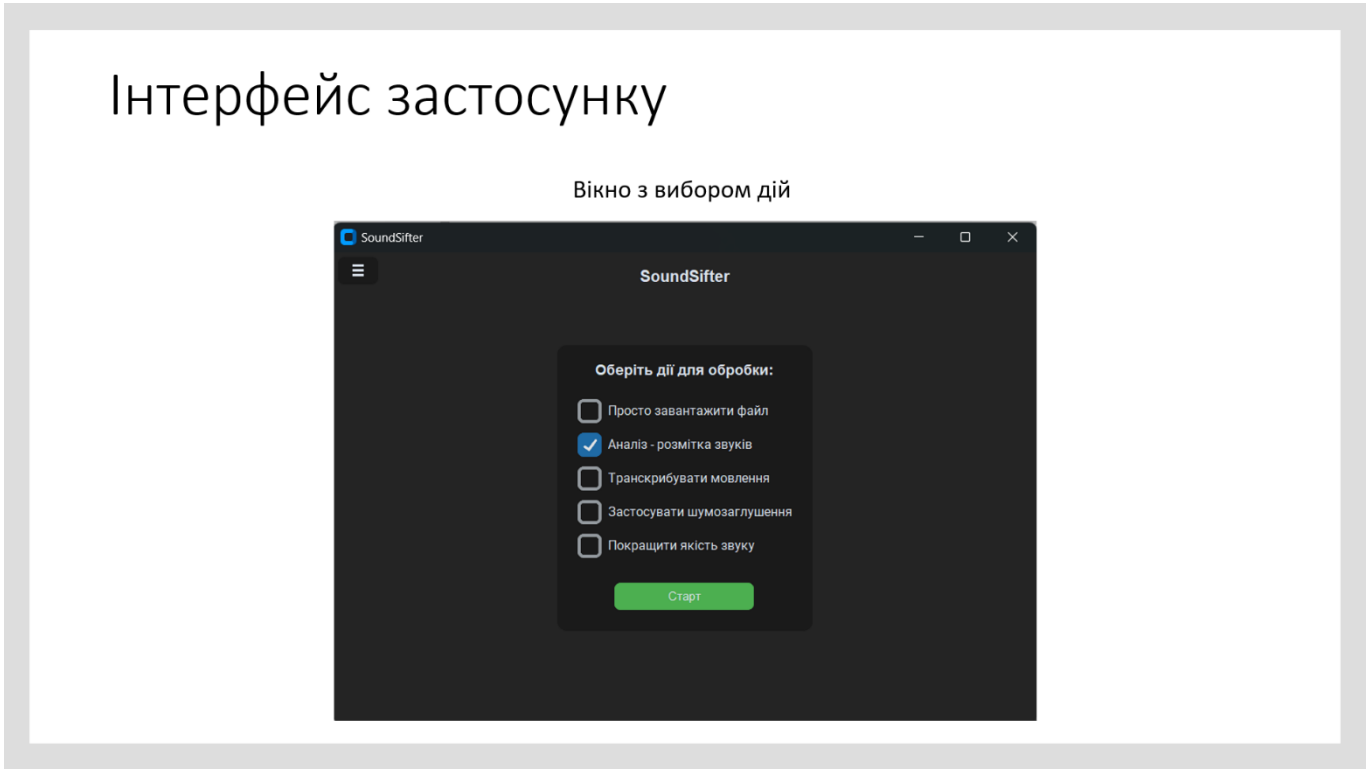

Рисунок Г.12 – Слайд вікна з вибором дій

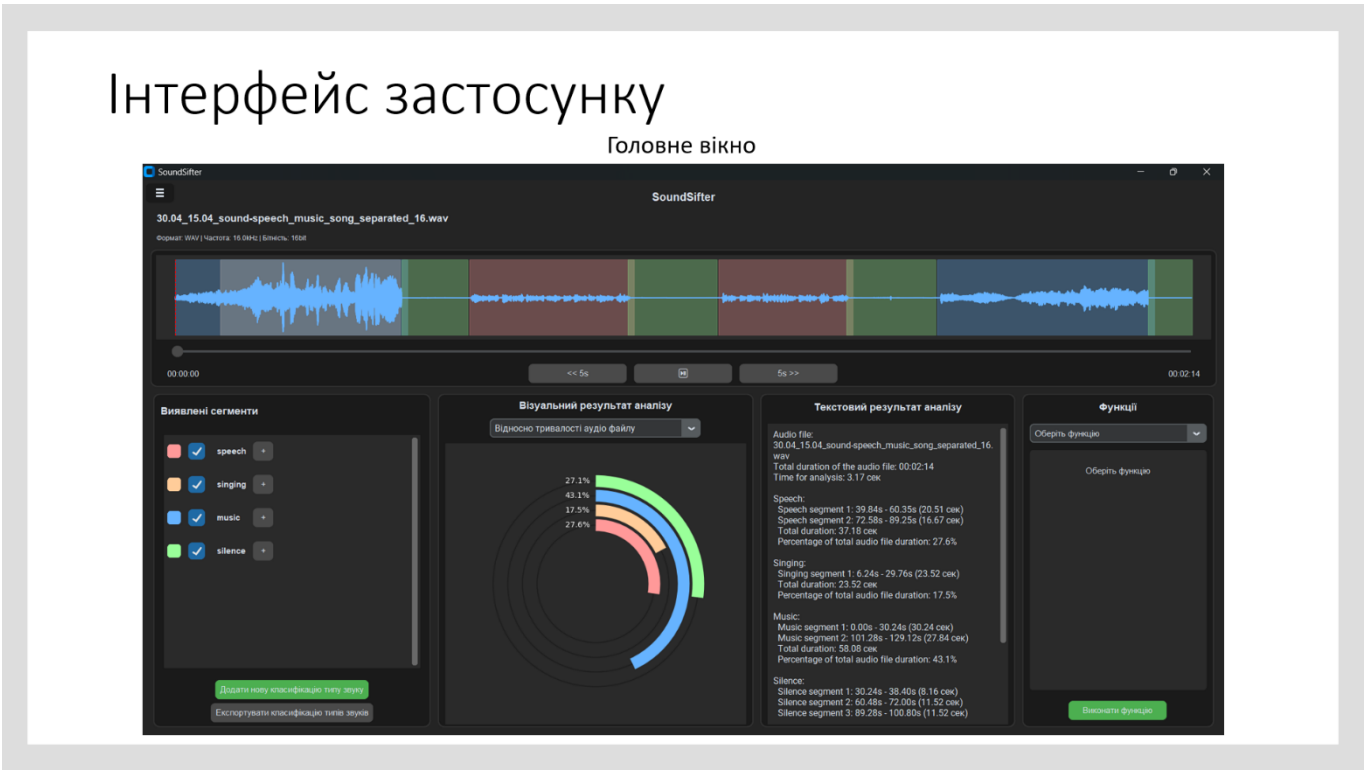

Рисунок Г.13 – Слайд головного вікна

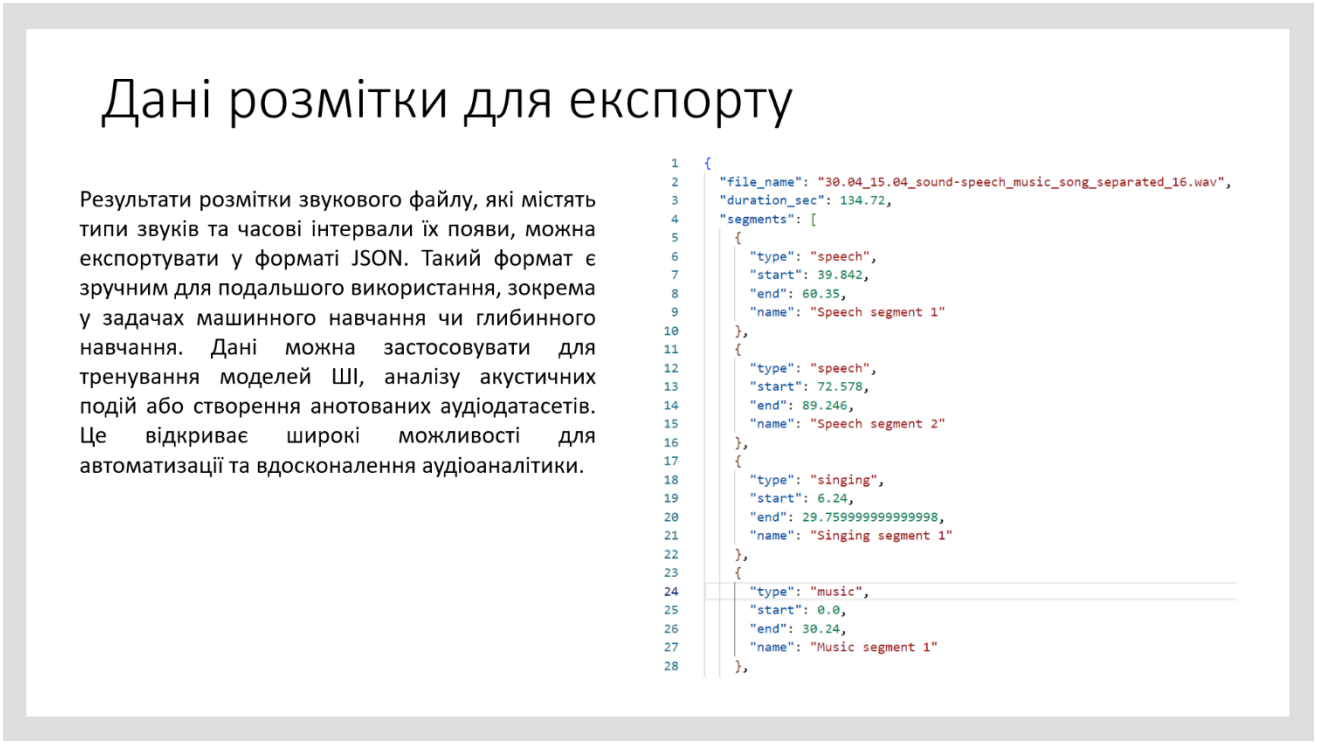

Рисунок Г.14 – Слайд даних для експорту розмітки звуків

Дані транскрипції мовлення у форматі.srt

Після завершення виконання функції транскрипції мовлення у текст, у попередньо обраній папці буде створено файл .srt із розпізнаним текстом, розбитим за часовими мітками. Такий підхід забезпечує зручність подальшого використання результатів, зокрема для створення субтитрів або аналізу мовлення.

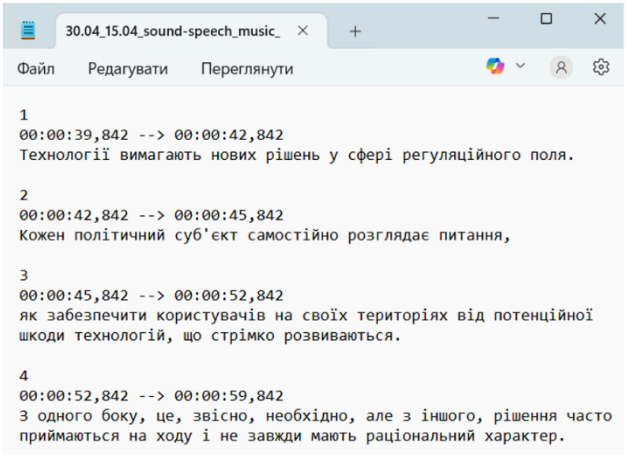

Рисунок Г.15 – Слайд результату транскрипції мовлення

Використані технології для реалізації програмної частини системи

Мова програмування: Python

Середовище розробки: Microsoft VS Code

Моделі: Silero VAD, YamNet, DeepFilterNet2, Whisper

Рисунок Г.16 – Слайд використаних технологій

Апробація та публікація матеріалів бакалаврської кваліфікаційної роботи

Апробація матеріалів кваліфікаційної роботи. Основні положення роботи доповідалися та обговорювалися на Всеукраїнській науково- технічній конференції підрозділів Вінницького національного технічного університету (Вінниця, 2025).

Публікації. Основні результати досліджень було опубліковано у матеріалах Всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету.

Рисунок Г.17 – Слайд апробації та публікацій

Висновок

У бакалаврській кваліфікаційній роботі було розроблено програмний застосунок SoundSifter для аналізу, обробки та транскрибування звукових файлів. В ході роботи спроєктовано архітектуру, реалізовано основні функціональні модулі – виявлення мовлення, класифікація типів звуків, очищення сигналу, розпізнавання мовлення – та створено графічний інтерфейс на базі бібліотеки customtkinter.

Для розробки використано мову Python та популярні бібліотеки pydub, soundfile, matplotlib, torch.Застосунок протестовано на реальних даних, підтверджено його стабільність і коректну роботу в усіх режимах. Особливістю системи є підтримка автоматичної розмітки аудіо з можливістю експорту у формати для машинного навчання, що відкриває перспективи використання не лише в освітніх, а й у професійних сферах, зокрема для архівування мовного контенту.

Рисунок Г.18 – Слайд висновку

Рисунок Г.19 – Кінцевий слайд