НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**

Завідувач кафедри
комп'ютерних наук
_____ /Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

«_____»_____ р

# БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

## «Програмне забезпечення інформаційної системи перетворення мовних повідомлень в текстовий формат»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент / Голуб Б.Л./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи : / Семко В.В/
підпис ПІБ

Виконав: / Котик О.В./
підпис ПІБ

**КИЇВ-2025**

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

**ЗАТВЕРДЖУЮ**

Завідувач кафедри
комп'ютерних наук

_____/ Голуб Б.Л., доцент, к.т.н /
підпис

"  " _____р.

# ЗАВДАННЯ

**на виконання бакалаврської кваліфікаційної роботи**

студенту Котику Олегу Володимировичу

Спеціальність 121 «Інженерія програмного забезпечення»

1. Тема роботи: Програмне забезпечення інформаційної системи перетворення мовних повідомлень в текстовий формат

Затверджена наказом ректора НУБіП України № 2249 "С" від 16.12.2024

2. Термін подання завершеної роботи на кафедру _____._____._____
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань що розглядаються:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів для розробки системи.
3. Проектування інформаційної системи.
4. Висновки.

Керівник бакалаврської кваліфікаційної роботи_____/ Семко В.В. /
підпис ініціали та прізвище

Завдання прийняв до виконання _____/ Котик О.В. /
підпис ініціали та прізвище

Дата отримання завдання _____._____._____
рік, місяць, чссло

Зміст

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. DTW - Dynamic Time Warping
2. AI - Artificial Intelligence
3. HMM - Hidden Markov Models
4. KNN - K-Nearest Neighbors
5. MFCC - Mel-frequency Cepstral Coefficients
6. GPU - Graphics Processing Unit
7. CPU – Central Processing Unit
8. UML – Unified Modeling Language
9. ER – Entity-Relationship

# ВСТУП

У сучасному світі інформаційні технології відіграють ключову роль у обробці та аналізі мовних даних. Перетворення аудіо- та відеоповідомлень у текстовий формат є актуальним завданням, яке знаходить застосування в багатьох сферах: від автоматизації створення субтитрів і стенограм до забезпечення доступності контенту для людей з обмеженими можливостями слуху. Особливо важливим є створення програмного забезпечення, здатного точно транскрибувати мовлення, зокрема українською мовою, яка часто є недостатньо представленою в подібних системах.

Метою даної дипломної роботи є розробка десктопного програмного забезпечення для перетворення мовних повідомлень у текстовий формат з підтримкою української та англійської мов. Запропонований застосунок забезпечує зручний інтерфейс для вибору файлів, налаштування параметрів транскрипції, перегляду та експорту результатів у текстові формати. Система працює локально, що гарантує автономність, безпеку даних і доступність для користувачів без необхідності підключення до мережі.

Актуальність роботи зумовлена зростаючою потребою в ефективних і доступних інструментах для автоматизації транскрибування мовлення. Розробка програмного забезпечення, яке поєднує сучасні методи розпізнавання мовлення та забезпечує гнучкість у виборі підходів до обробки даних, сприятиме підвищенню якості текстової інтерпретації аудіо- та відеоматеріалів.

Об’єктом дослідження є процеси перетворення мовних повідомлень у текстовий формат. Предметом дослідження виступає програмне забезпечення, яке реалізує алгоритми розпізнавання мовлення та забезпечує зручний інтерфейс для користувача.

У рамках роботи вирішуються такі завдання:

- аналіз сучасних методів і технологій розпізнавання мовлення;
- проектування архітектури десктопного програмного забезпечення;
- розробка функціоналу для транскрибування аудіо- та відеофайлів;

- створення зручного графічного інтерфейсу; тестування та оцінювання ефективності системи.

Новизна роботи полягає в розробці програмного забезпечення, яке інтегрує різні підходи до розпізнавання мовлення, забезпечуючи гнучкість вибору оптимального методу залежно від потреб користувача, а також у створенні інструменту для обробки україномовного контенту з урахуванням його лінгвістичних особливостей.

Практична цінність роботи полягає в створенні універсального програмного продукту, який може бути використаний для автоматизації транскрибування в освітніх, медійних, юридичних та інших сферах. Локальна робота застосунку забезпечує його доступність і незалежність від зовнішніх ресурсів, що робить його зручним для широкого кола користувачів.

Засобами розробки було обрано сучасні інструменти, такі як Python, openAI-Whisper, PyQT6, алгоритм DTW.

Записка складається з чотирьох розділів. Перший розділ описує аналіз предметної області, огляд існуючих рішень та формує постановку задачі. У другому розділі це проєктування системи з використанням UML діаграм та моделювання логічної моделі даних за допомогою ER діаграм.У третьому розділі описується технічні аспекти реалізації програмної системи, вибір та обгрунтування технологій, які потрібні для реалізації системи. Четвертий розділ описує вимоги до апаратних вимог, тестування та рекомендації для впровадження системи.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Опис предметної області

Предметна область дослідження охоплює сферу автоматизації обробки мовних даних, зокрема технології перетворення аудіо- та відеоповідомлень у текстовий формат. Це є частиною ширшого напрямку обробки природної мови і має значний вплив на різні аспекти сучасного життя, включаючи освіту, медіа, юриспруденцію, медицину та забезпечення доступності інформації для людей із порушеннями слуху.

Останніми роками спостерігається значне зростання попиту на технології транскрибування, що пов’язано з експоненційним збільшенням обсягу аудіо- та відеоконтенту в цифровому просторі. За даними звіту YouTube за 2024 рік, щоденно на платформу завантажується понад 500 годин нового відеоконтенту, значна частина якого потребує субтитрів для забезпечення доступності та локалізації. Аналогічні тенденції спостерігаються в інших сферах: подкасти, онлайн-курси та вебінари стають дедалі популярнішими. Наприклад, за оцінками UNESCO, у 2023 році кількість слухачів подкастів у світі перевищила 450 мільйонів, і значна їхня частина потребує транскрипцій для індексації, пошуку чи перекладу. Це підкреслює актуальність створення інструментів, які здатні швидко та точно перетворювати мовлення у текст, особливо в умовах, коли ручне транскрибування є трудомістким і дорогим процесом.

Перетворення мовних повідомлень у текстовий формат, або транскрибування, є важливим завданням, яке дозволяє документувати усну мову, створювати субтитри до відеоконтенту, готувати стенограми для офіційних заходів, а також аналізувати аудіодані для подальшого використання. Наприклад, у медіаіндустрії транскрибування використовується для створення субтитрів до фільмів і телепередач, у юридичній сфері — для документування судових засідань, а в освіті — для підготовки текстових матеріалів із лекцій і семінарів.

Основними користувачами технологій транскрибування є клієнти та стенографісти. Клієнти — це широкий спектр користувачів, які потребують текстового представлення аудіо- чи відеоматеріалів для особистих або професійних цілей, наприклад, студенти, дослідники або журналісти, які документують інтерв'ю чи лекції. Стенографісти — це фахівці, які займаються підготовкою точних текстових стенограм і субтитрів на основі мовних матеріалів, зокрема для медіа, юридичних чи офіційних потреб. Обидві категорії користувачів стикаються з викликами, такими як необхідність високої точності розпізнавання, підтримка різних мов, а також обробка матеріалів із шумом або нечіткою вимовою.

Технології розпізнавання мовлення мають довгу історію, яка бере початок із середини XX століття. Перші системи, такі як Audrey, розроблена Bell Labs у 1952 році, могли розпізнавати лише цифри, вимовлені одним голосом, із точністю до 90% [1]. У 1970-х роках з'явилися системи на основі Hidden Markov Models (HMM), які стали основою для розпізнавання мовлення в наступні десятиліття. HMM дозволяли моделювати часові послідовності звуків, враховуючи ймовірності переходів між станами, що значно підвищило точність розпізнавання. У 1990-х роках DARPA профінансувала проєкти, які привели до створення систем, здатних розпізнавати мовлення в реальному часі, таких як Sphinx від Carnegie Mellon University. На початку 2000-х років почали активно застосовуватися методи машинного навчання, зокрема нейронні мережі, які покращили обробку складних мовних моделей. Переломним моментом став 2010-ті роки, коли глибоке навчання (Deep Learning) революціонізувало сферу розпізнавання мовлення. Системи, такі як Deep Speech від Mozilla і Google Speech-to-Text, почали використовувати глибокі нейронні мережі (DNN) і трансформерні архітектури, що дозволило досягти високої точності навіть у шумних умовах. Технології сьогодення, такі як openai-whisper, використовують попередньо навчені моделі машинного навчання на великих масивах даних, що забезпечує підтримку багатьох мов, включно з українською, хоча й із певними обмеженнями.

Предметна область також включає аналіз сучасних тенденцій у технологіях розпізнавання мовлення. Останніми роками активно розвиваються методи, засновані на глибокому навчанні, які забезпечують високу точність транскрипції, однак вони часто потребують значних обчислювальних ресурсів і доступу до великих масивів даних для навчання. Водночас традиційні методи, такі як шаблонне розпізнавання, залишаються актуальними для сценаріїв із обмеженими ресурсами або специфічними завданнями. Одним із ключових викликів у предметній області є створення систем, які поєднують високу точність, доступність і можливість роботи в автономному режимі, без залежності від мережевих ресурсів.

Предметна область охоплює як теоретичні, так і практичні аспекти розробки технологій транскрибування, з урахуванням потреб користувачів, особливостей обробки різних мов і сучасних тенденцій у розвитку методів розпізнавання мовлення. Зростання попиту на автоматизацію обробки аудіо- та відеоконтенту, широке застосування транскрибування в різних сферах і виклики, пов’язані з підтримкою української мови, підкреслюють актуальність створення доступних і автономних рішень. Історичний розвиток технологій показує еволюцію від простих систем до складних моделей глибокого навчання, але сучасні виклики, такі як шум і багатомовність, потребують подальших досліджень. Розробка програмного забезпечення, яке враховує ці аспекти, сприятиме підвищенню якості текстової інтерпретації мовних матеріалів і розширенню можливостей для україномовних користувачів.

## 1.2 Аналіз вимог до програмної системи

Розробка програмної системи для перетворення мовних повідомлень у текстовий формат потребує чіткого визначення вимог, які забезпечать її функціональність, зручність використання та відповідність потребам користувачів. На основі аналізу предметної області та потреб користувачів було сформовано наступні функціональні та нефункціональні вимоги до системи.

Функціональні вимоги визначають основні можливості, які повинна надавати система для виконання завдань користувачів — клієнтів і стенографістів.

Функціональні вимоги:

1. Завантаження аудіо- та відеофайлів. Система повинна дозволяти користувачу завантажувати аудіо- чи відеофайли для подальшої обробки. Підтримувані формати мають включати поширені типи файлів, такі як MP3, WAV для аудіо та MP4 для відео.
2. Вибір параметрів транскрибування. Система повинна надавати можливість вибору мови транскрибування (українська або англійська), а також режиму розпізнавання (на основі глибокого навчання або шаблонного розпізнавання). Крім того, користувач повинен мати змогу обирати пристрій для обробки процесор або відеокарту.
3. Виконання транскрибування. Система має забезпечувати транскрибування аудіо- чи відеофайлів із можливістю використання двох різних методів розпізнавання мовлення, залежно від вибору користувача.
4. Перегляд результатів транскрибування. Користувач повинен мати можливість переглядати текстовий результат транскрибування безпосередньо в інтерфейсі системи.
5. Експорт результатів. Система має підтримувати експорт результатів транскрибування у файли форматів TXT і SRT для подальшого використання.

Нефункціональні вимоги визначають якісні характеристики системи, які впливають на її ефективність, зручність і надійність.

Нефункціональні вимоги:

1. Локальна робота. Система повинна працювати повністю локально, без необхідності підключення до мережі, що забезпечує автономність і безпеку даних.
2. Зручність інтерфейсу. Графічний інтерфейс має бути інтуїтивно зрозумілим, забезпечувати легкий доступ до всіх функцій і відповідати принципам ергономіки. Користувач повинен швидко орієнтуватися в системі без додаткового навчання.
3. Продуктивність. Система має забезпечувати розумний час обробки файлів: транскрибування аудіофайлу тривалістю 1 хвилина не повинно перевищувати 2 хвилин на середньостатистичному обладнанні (CPU) і 1 хвилини при використанні GPU.
4. Надійність. Система має коректно обробляти помилки, наприклад, у разі завантаження пошкоджених файлів, і повідомляти користувача про проблеми без аварійного завершення роботи.
5. Безпека. Оскільки система працює локально, вона не повинна передавати дані користувача на зовнішні сервери, що гарантує конфіденційність оброблюваної інформації.

Сформовані вимоги відображають основні потреби користувачів у процесі транскрибування аудіо- та відеофайлів, а також враховують сучасні тенденції у розробці програмного забезпечення. Виконання цих вимог забезпечить створення функціональної, зручної та ефективної системи, яка відповідатиме очікуванням цільової аудиторії.

## 1.3 Моделювання предметної області

Моделювання предметної області є ключовим етапом у розробці програмного забезпечення, який дозволяє формалізувати процеси, взаємодії та структуру даних у межах сфери перетворення мовних повідомлень у текстовий формат. Для цього використовується Unified Modeling Language (UML) —

стандартизована мова моделювання, що надає широкий набір діаграм для опису систем із різних перспектив [2]. UML підтримує розробку через візуалізацію, специфікацію, конструювання та документування, що робить її незамінним інструментом для аналізу та проектування. У цьому підрозділі розглянуто можливості UML, а також детально описано діаграми прецедентів і послідовності, які застосовані для моделювання предметної області транскрибування.

UML, розроблена Object Management Group (OMG), включає 14 типів діаграм, які поділяються на структурні (наприклад, класова діаграма, діаграма компонентів) і поведінкові (наприклад, діаграма прецедентів, діаграма послідовності, діаграма станів). Ці діаграми дозволяють описати систему на різних рівнях абстракції: від загального уявлення про акторів і їхні взаємодії до детального аналізу внутрішніх процесів і потоків даних. Для предметної області транскрибування найбільшу цінність мають поведінкові діаграми, оскільки вони фокусуються на динаміці системи, відображаючи, як користувачі взаємодіють із програмним забезпеченням і як дані обробляються в реальному часі. Використання UML сприяє чіткому розумінню вимог, полегшує комунікацію між розробниками та замовниками та слугує основою для подальшого проектування архітектури. У цьому проєкті обрано діаграми прецедентів і послідовності як найбільш доречні для моделювання взаємодії користувачів із системою транскрибування.

Діаграма прецедентів (Use Case Diagram) відображає основних акторів предметної області та їхню взаємодію із системою транскрибування(рис. 1.1). Акторами є клієнти та стенографісти, які використовують систему для обробки аудіо- та відеофайлів. Основні прецеденти включають завантаження файлів, перегляд доступних файлів, транскрибування, отримання результатів і створення текстових файлів із результатами.

Діаграма показує, що обидва актори мають схожі потреби: вони можуть завантажувати файли, переглядати доступні опції, ініціювати транскрибування та отримувати результати. Однак стенографісти додатково орієнтовані на

створення файлів у форматі SRT для субтитрів, що відображає їхню професійну специфіку. Зв’язки між акторами та прецедентами представлені спрямованими стрілками, які вказують на те, хто ініціює дію. Ця діаграма допомагає визначити функціональні вимоги та зрозуміти, як різні групи користувачів використовуватимуть систему, що є важливим для подальшого проектування інтерфейсу.

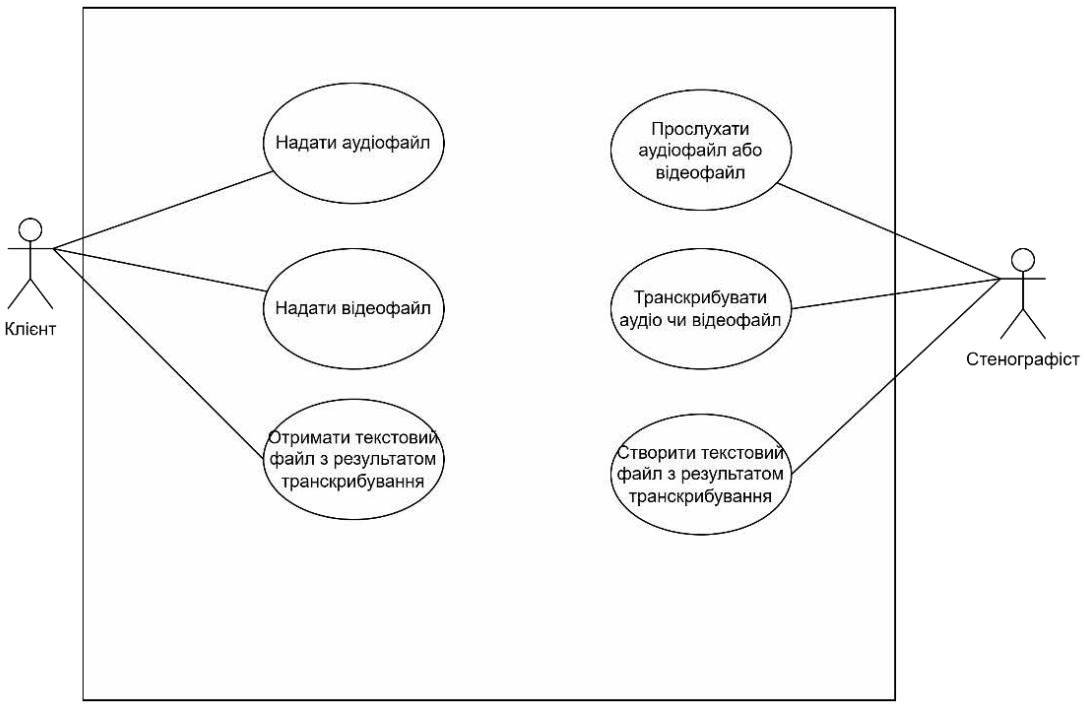


Рис. 1.1 – Діаграма прецедентів предметної області

Діаграма послідовності деталізує процес транскрибування аудіо- чи відеофайлу для актора "Клієнт", відображаючи послідовність дій і взаємодію між об’єктами системи (рис. 1.2). Вона включає актора "Клієнт", об’єкти "Графічний інтерфейс", "Модуль обробки файлів" і "Модуль транскрибування", а також тимчасову вісь, яка показує хронологію операцій. Процес починається з того, що клієнт через графічний інтерфейс завантажує файл і обирає параметри (мова, метод, пристрій). Графічний інтерфейс передає запит до модуля обробки файлів, який витягує аудіодоріжку, сегментує її на частини та готує дані для транскрибування. Потім модуль обробки викликає модуль транскрибування, який виконує розпізнавання за обраним методом (DTW або openai-whisper) і

повертає результат. Нарешті, графічний інтерфейс відображає текстовий вивід, дозволяючи клієнту переглянути його або зберегти у файл.

Діаграма також ілюструє асинхронні операції, наприклад, відображення прогресу транскрибування через сигнали, що підвищує зручність використання. Усі обміни даними відбуваються локально, що відповідає вимозі автономності системи. Ця діаграма деталізує, як система реагує на дії користувача, забезпечуючи чіткий ланцюжок операцій від завантаження до отримання результату, і слугує основою для реалізації логіки програми.

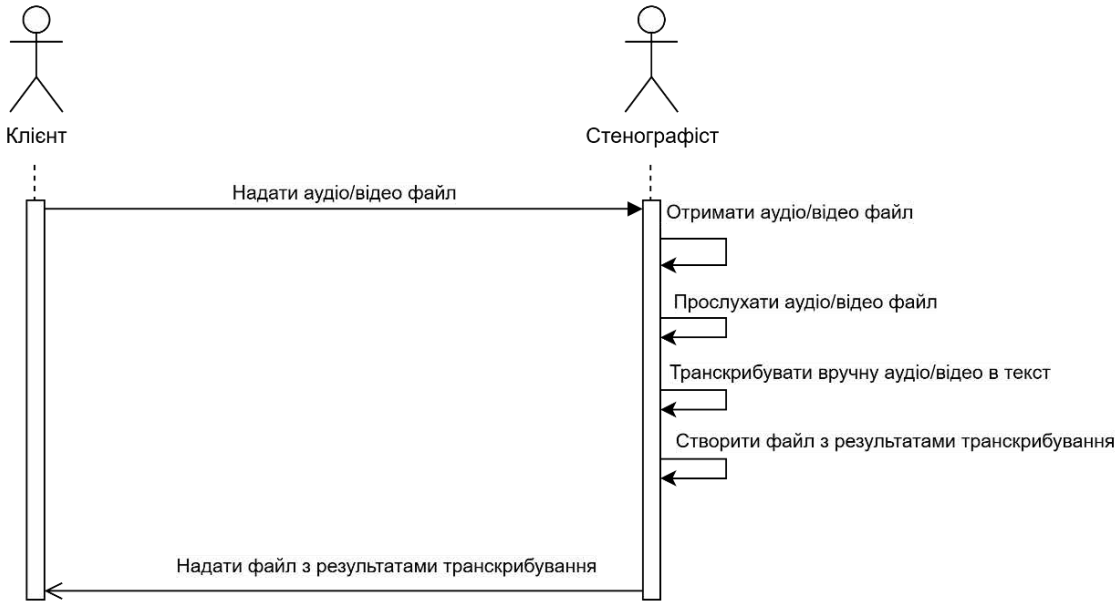


Рис. 1.2 – Діаграма послідовності

Моделювання предметної області за допомогою UML забезпечило комплексне розуміння взаємодії користувачів із системою, послідовності операцій і руху даних. Діаграми прецедентів і послідовності слугують основою для подальшого проектування архітектури та реалізації програмного забезпечення, а також допомагають у тестуванні та документуванні системи. Цей підхід підкреслює важливість візуалізації для аналізу складних процесів і сприяє створенню ефективного рішення для транскрибування з урахуванням особливостей української та англійської мов.

## 1.4 Аналіз існуючих систем

Для оцінки сучасного стану технологій транскрибування аудіо- та відеофайлів було проаналізовано кілька існуючих систем, які використовуються для перетворення мовних повідомлень у текстовий формат. Розглянуто їхні функціональні можливості, переваги, недоліки та проведено порівняння з розробленою системою, що працює локально, підтримує українську та англійську мови, а також використовує два методи транскрибування (глибоке навчання і шаблонне розпізнавання).

- Mozilla DeepSpeech — це система з відкритим кодом, яка використовує глибоке навчання для розпізнавання мовлення. Вона базується на нейронних мережах і була розроблена для забезпечення високої точності транскрибування. Однією з ключових переваг DeepSpeech є її здатність адаптуватися до нових мов і діалектів, що робить її гнучкою для використання в різних лінгвістичних контекстах. Система також приділяє значну увагу конфіденційності даних, забезпечуючи шифрування інформації користувача, що є важливим для обробки чутливих даних.

Недоліком DeepSpeech є залежність від великих обсягів даних для навчання моделей, що може ускладнити її використання в умовах обмежених ресурсів. Крім того, система потребує значних обчислювальних потужностей для ефективної роботи, що може бути проблематичним для локального використання на пристроях із низькою продуктивністю. У порівнянні з розробленою системою, DeepSpeech не підтримує шаблонне розпізнавання, а також не надає можливості вибору між CPU і GPU для обробки, що обмежує її гнучкість у налаштуваннях.

- IBM Watson Speech to Text — це хмарна система, яка використовує передові алгоритми глибокого навчання для транскрибування мовлення. Вона відома своєю масштабованістю та легкістю інтеграції в робочі процеси, що робить її популярною серед бізнес-користувачів. Система підтримує широкий спектр мов і має

інтуїтивно зрозумілий інтерфейс із розвиненою документацією. IBM Watson також використовує великі набори даних, такі як TIMIT, для підвищення точності розпізнавання [3].

Проте залежність від хмарної інфраструктури є суттєвим недоліком, оскільки це вимагає постійного підключення до мережі та може створювати ризики для конфіденційності даних. У порівнянні з розробленою системою, IBM Watson не працює локально, що суперечить вимозі автономності. Крім того, система не підтримує шаблонне розпізнавання, яке є однією з особливостей розробленого рішення.

- Amazon Transcribe — це хмарний сервіс, який забезпечує транскрибування аудіо- та відеофайлів у реальному часі. Система підтримує функції, такі як розпізнавання кількох мов, автоматичне створення субтитрів і аналіз розмов (наприклад, для кол-центрів). Amazon Transcribe Medical, спеціалізована версія сервісу, дозволяє транскрибувати медичні розмови для створення клінічних записів. Система ефективно інтегрується з іншими сервісами AWS, що робить її зручною для бізнес-додатків [4].

Основним недоліком Amazon Transcribe є її хмарна природа, що унеможливлює локальне використання та може створювати проблеми з конфіденційністю даних. Також система не підтримує шаблонне розпізнавання і не надає можливості налаштування сегментації, як це реалізовано в розробленій системі. Підтримка української мови в Amazon Transcribe є обмеженою, що робить її менш придатною для україномовного контенту порівняно з розробленим рішенням.

- Google Speech-to-Text — це ще одна хмарна система, яка підтримує розпізнавання мовлення для понад 100 мов і діалектів. Вона використовується в багатьох продуктах Google, таких як YouTube (для створення субтитрів) і Google Assistant. Система застосовує глибоке навчання для підвищення точності та працює як у реальному часі, так і з попередньо записаними файлами. Google Speech-to-Text

також підтримує розпізнавання контексту через інтеграцію з великими мовними моделями [5].

Недоліками системи є її хмарна архітектура, що виключає локальну роботу, і обмежена підтримка української мови, яка поступається якості розпізнавання англійської. Google Speech-to-Text не пропонує альтернативних методів транскрибування, таких як шаблонне розпізнавання, і не надає можливості налаштування сегментації, що є перевагою розробленої системи.

Розглянуті системи переважно базуються на хмарній інфраструктурі (IBM Watson, Amazon Transcribe, Google Speech-to-Text), що відрізняє їх від розробленого прототипу, який орієнтований на локальну роботу без необхідності підключення до мережі. Mozilla DeepSpeech, як єдина система з відкритим кодом, подібна до прототипу за можливістю локального використання, але потребує значних ресурсів для навчання, на відміну від прототипу, який включає шаблонне розпізнавання як альтернативу.

Хмарні системи пропонують масштабованість і інтеграцію з іншими сервісами, але їхня залежність від мережі може бути недоліком для автономного використання. Прототип, у свою чергу, передбачає підтримку української мови та два методи транскрибування (глибоке навчання і шаблонне розпізнавання), що може розширити його привабливість для україномовних користувачів і сценаріїв із обмеженими обчислювальними ресурсами. Однак його можливості ще потребують подальшого тестування та доопрацювання для повного порівняння з комерційними рішеннями.

## 1.5 Постановка задачі

На основі аналізу предметної області, вимог до програмної системи, моделювання та оцінки існуючих рішень визначено основну проблему, яка потребує вирішення: відсутність доступного локального програмного забезпечення для транскрибування аудіо- та відеофайлів із підтримкою української та англійської мов, яке б поєднувало сучасні методи глибокого

навчання з традиційними методами шаблонного розпізнавання, забезпечуючи при цьому зручний графічний інтерфейс і автономність роботи.

Метою роботи є розробка прототипу десктопного програмного забезпечення для перетворення мовних повідомлень у текстовий формат, яке працює локально, підтримує українську та англійську мови, дозволяє користувачам обирати між двома методами транскрибування (глибоке навчання та шаблонне розпізнавання) і забезпечує зручний графічний інтерфейс для обробки файлів і перегляду результатів.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. Розробити архітектуру програмного забезпечення, яка забезпечить локальну обробку даних без необхідності підключення до мережі.
2. Реалізувати два методи транскрибування: на основі глибокого навчання (з використанням сучасних бібліотек) та шаблонного розпізнавання (з порівнянням із еталонними зразками).
3. Забезпечити підтримку транскрибування українською та англійською мовами з урахуванням їхніх лінгвістичних особливостей.
4. Розробити графічний інтерфейс, який дозволяє користувачу завантажувати аудіо- чи відеофайли, обирати мову, метод транскрибування, пристрій (CPU або GPU), переглядати результати та експортувати їх у формати TXT і SRT.
5. Реалізувати функціонал для шаблонного розпізнавання, який включає налаштування параметрів сегментації, візуалізацію сегментів і порівняння з еталонними зразками.
6. Провести тестування прототипу для оцінки його працездатності, точності транскрибування та зручності використання.
7. Проаналізувати отримані результати та порівняти ефективність двох методів транскрибування для різних типів аудіо- та відеофайлів.

Отримані результати можуть бути використані для подальшого вдосконалення системи, а також як основа для створення повноцінного

програмного продукту для автоматизації транскрибування в різних сферах, таких як освіта, медіа та юриспруденція.

# 2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1 Загальна архітектура програмної системи

Загальна архітектура розробленого десктопного програмного забезпечення для транскрибування аудіо- та відеофайлів базується на модульному підході, який забезпечує гнучкість, розширюваність і локальну обробку даних. Система складається з чотирьох основних компонентів: графічного інтерфейсу, модуля завантаження та обробки файлів, модуля транскрибування та модуля експорту результатів. Кожен компонент взаємодіє з іншими через чітко визначені інтерфейси, що дозволяє підтримувати автономну роботу та інтеграцію двох методів розпізнавання мовлення: глибокого навчання (на основі бібліотеки) і шаблонного розпізнавання (з використанням алгоритму Dynamic Time Warping).

Основні компоненти:

1. Графічний інтерфейс: Цей компонент реалізовано за допомогою бібліотеки PyQt6 і забезпечує зручний доступ до функціональності системи. Він включає панелі для завантаження файлів, вибору мови (українська або англійська), методу транскрибування, пристрою (CPU або GPU), а також перегляду результатів і налаштування параметрів сегментації для шаблонного розпізнавання. Інтерфейс передає команди та параметри до інших модулів і відображає отримані текстові дані або графіки сегментів.
2. Модуль завантаження та обробки файлів: цей модуль відповідає за імпорт аудіо- та відеофайлів, їхню початкову обробку (наприклад, екстракцію аудіодоріжки з відео) і підготовку до транскрибування. Модуль використовує бібліотеки: librosa, numpy, для аналізу аудіосигналів і передачі їх до модуля транскрибування. Модуль також забезпечує підтримку різних форматів файлів (MP3, WAV, MP4).

3. Модуль транскрибування: центральний компонент системи, який реалізує два режими розпізнавання. Перший режим базується на бібліотеці openai-whisper для глибокого навчання, що дозволяє обробляти мовлення з високою точністю за умови доступу до GPU. Другий режим базується на використанні алгоритму Dynamic Time Warping із MFCC-ознаками та еталонними зразками аудіо файлів для шаблонного розпізнавання, надаючи альтернативу для сценаріїв із обмеженими ресурсами. Модуль отримує параметри від графічного інтерфейсу і повертає текстові результати.
4. Модуль експорту результатів: цей модуль відповідає за збереження результатів транскрибування у текстових форматах (TXT і SRT). Він дозволяє користувачу переглядати результати перед експортом і підтримує налаштування для стенографістів, таких як форматування субтитрів. Модуль працює локально, зберігаючи файли в директорії, визначеній користувачем.

Компоненти системи взаємодіють через об’єктно-орієнтований підхід із чітким розподілом відповідальності. Графічний інтерфейс виступає основним входом для користувача, передаючи запити до модуля завантаження та обробки файлів, який підготовляє дані для модуля транскрибування. Після обробки результати передаються до модуля експорту для збереження. Усі операції виконуються локально, що виключає необхідність у мережевих запитах. Система підтримує вибір між CPU і GPU, щоб користувач міг вибирати режим роботи для підвищення продуктивності.

## 2.2 Логічна модель даних у вигляді ER-діаграми

Логічна модель даних описує структуру даних, які обробляються в межах програмного забезпечення для транскрибування аудіо- та відеофайлів. Оскільки система працює локально і не використовує базу даних, дані зберігаються в

пам’яті програми під час обробки, а вхідні файли та результати транскрибування зберігаються у файловій системі. Логічна модель представлена у вигляді ER-діаграми, яка відображає шість основних сутностей, їхні атрибути та зв’язки між ними.

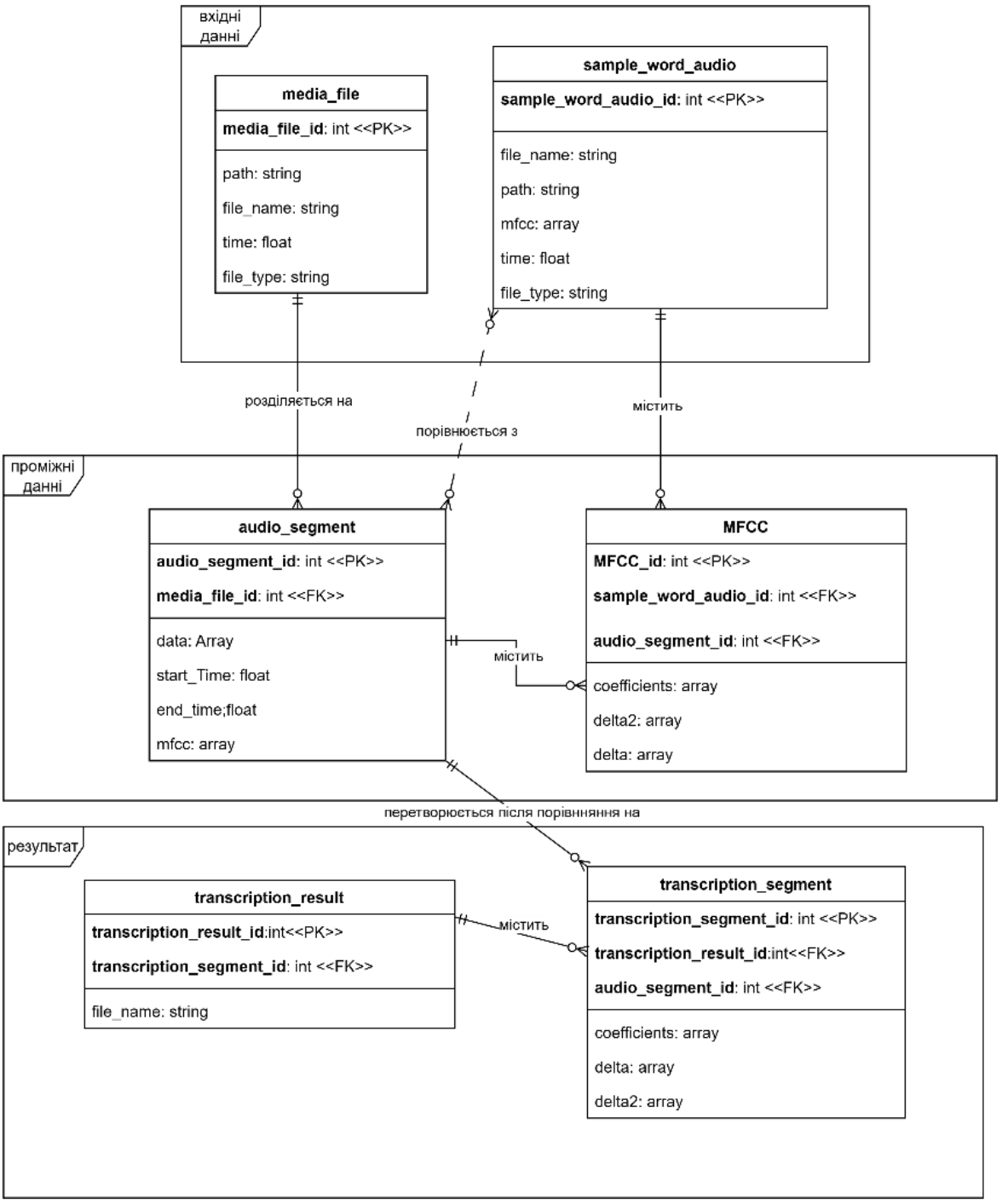


Рис. 2.1 – ER-діаграма логічної моделі даних

На ER-діаграмі(рис. 2.1) представлено шість сутностей, які відображають логіку:

- media_file — вхідний аудіо- чи відеофайл із атрибутами media_file_id, path, file_name, time, file_type.
- sample_word_audio — зразок аудіофайлу для порівняння в шаблонному розпізнаванні, з атрибутами sample_word_audio_id, file_name, path, mfcc, time, file_type.
- audio_segment — сегмент аудіо після розбиття вхідного файлу, з атрибутами audio_segment_id, media_file_id, data, start_time, end_time, mfcc.
- MFCC — MFCC-ознаки для шаблонного розпізнавання, з атрибутами MFCC_id, sample_word_audio_id, audio_segment_id, coefficients, delta, delta2.
- transcription_segment — сегмент транскрипції після обробки, з атрибутами transcription_segment_id, transcription_result_id, audio_segment_id, coefficients, delta, delta2.
- transcription_result — кінцевий результат транскрипції, з атрибутами transcription_result_id, transcription_segment_id, file_name.

Дані представлені як об’єкти в пам’яті програми. Вхідний файл і зразок завантажуються з файлової системи, обробляються (розбиття на сегменти, витягнення MFCC, порівняння, транскрибування), а результат зберігається у файл (TXT або SRT). Це забезпечує автономність роботи.

Логічна модель у вигляді ER-діаграми описує структуру даних у системі, відображаючи етапи обробки від вхідного файлу до результату. Відсутність бази даних спрощує архітектуру, але потребує ефективного управління пам’яттю.

## 2.3 Діаграма пакетів

Діаграма пакетів представляє модульну організацію програмного забезпечення для транскрибування аудіо- та відеофайлів, показуючи, як компоненти системи розподілені на логічні блоки та як вони взаємодіють між собою. Такий підхід дозволяє чітко структурувати код, полегшуючи розробку,

підтримку та можливе розширення функціональності в майбутньому. Кожен пакет відповідає за певну частину роботи системи, а зв’язки між ними відображають передачу даних і залежності.

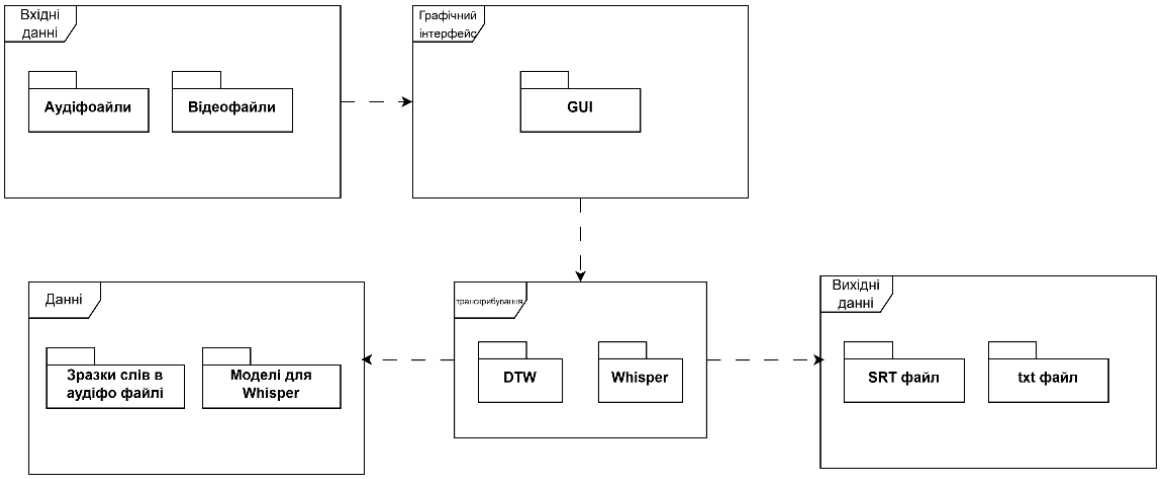


Рис. 2.2 – Діаграма пакетів

Пакет Вхідні данні об’єднує вхідні ресурси, які користувач завантажує в систему. Він включає підпакети «аудіофайли» для аудіофайлів у форматах, таких як MP3 чи WAV, і «відеофайли» для відеофайлів, наприклад MP4, з яких витягується аудіодоріжка. Пакет Графічний інтерфейс відповідає за взаємодію з користувачем через підпакет GUI, реалізований із використанням PyQt6. Цей підпакет забезпечує відображення інтерфейсу, де користувач може завантажувати файли, налаштовувати параметри транскрибування та переглядати результати. Пакет транскрибування є центральним у системі, включаючи підпакети DTW для шаблонного розпізнавання мовлення з використанням алгоритму Dynamic Time Warping і Whisper для глибокого навчання на основі бібліотеки openai-whisper. Пакет Данні забезпечує необхідні ресурси для транскрибування: підпакет Зразки слів в аудіофайлі містить еталонні аудіозразки для порівняння в DTW, а Моделі для Whisper зберігає попередньо навчені моделі для глибокого навчання. Пакет Вихідні данні відповідає за збереження результатів транскрибування, включаючи підпакети SRT файл для субтитрів у форматі SRT і txt файл для текстових документів.

Модульна структура системи забезпечує чіткий розподіл функціональності між пакетами, що спрощує тестування та подальший розвиток. Локальна обробка даних виключає залежність від мережевих ресурсів, а використання підпакетів дозволяє легко замінювати або додавати нові методи транскрибування. Наприклад, підпакет DTW може бути розширений новими алгоритмами порівняння, а Whisper — новими моделями для інших мов. Графічний інтерфейс ізольований від логіки обробки, що полегшує його оновлення без впливу на інші частини системи.

Діаграма пакетів демонструє логічну організацію системи, підкреслюючи її модульність і автономність. Такий підхід забезпечує гнучкість і масштабованість, дозволяючи ефективно управляти складністю розробки.

## 2.4 Діаграма компонентів

Діаграма компонентів описує внутрішню структуру програмного забезпечення для транскрибування аудіо- та відеофайлів, показуючи основні компоненти системи, їх взаємодію та залежності від зовнішніх бібліотек. Це дозволяє зрозуміти, як організовано код, як компоненти обмінюються даними і які інструменти використовуються для реалізації функціональності. Компоненти розподілені за їхньою роллю в системі, а залежності відображають використання зовнішніх бібліотек для виконання завдань.

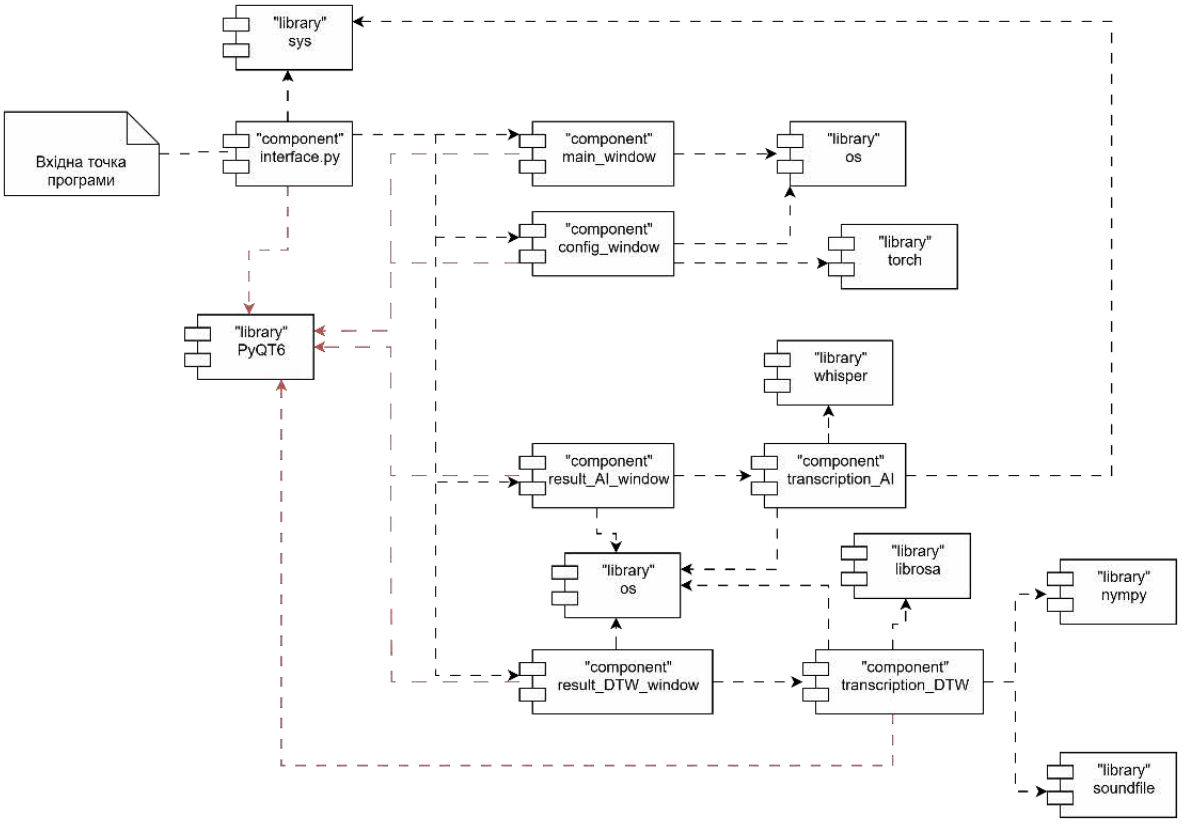


Рис. 2.3 – Діаграма компонентів

Компонент interface.py є точкою входу програми, ініціалізуючи запуск і координуючи взаємодію між іншими компонентами. Він залежить від бібліотеки sys для системних операцій і PyQt6 для створення графічного інтерфейсу, а також взаємодіє з компонентами main_window, config_window, result_AI_window і result_DTW_window. Компонент main_window представляє головне вікно програми, де користувач завантажує файли та запускає транскрибування. Він використовує os для роботи з файловою системою і залежить від PyQt6 для відображення інтерфейсу. Компонент config_window відповідає за налаштування параметрів транскрибування, таких як вибір мови, методу чи пристрою. Він залежить від os для доступу до файлової системи і torch для перевірки доступності GPU. Компонент result_AI_window відображає результати транскрибування, отримані методом глибокого навчання, використовуючи бібліотеки os і PyQt6. Він взаємодіє з transcription_AI, який безпосередньо виконує транскрибування за допомогою бібліотеки whisper, а також залежить від sys і os. Компонент result_DTW_window показує результати шаблонного розпізнавання, отримані через компонент transcription_DTW. Він залежить від

PyQt6 для інтерфейсу, os для файлових операцій і librosa для обробки аудіо. Компонент transcription_DTW виконує шаблонне розпізнавання з використанням алгоритму DTW, залежачи від librosa для витягнення ознак, numpy для обчислень і soundfile для роботи з аудіофайлами.

Діаграма компонентів відображає чітку організацію системи, підкреслюючи модульність і автономність. Структура сприяє ефективній реалізації функціональності та спрощує майбутнє розширення.

# 3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Опис концепції програмного продукту

Розроблений програмний продукт є десктопним застосунком, призначеним для перетворення аудіо та відеофайлів у текстовий формат, із локальною обробкою даних, що забезпечує автономність, безпеку та незалежність від зовнішніх мережевих ресурсів. Основна ідея полягає в створенні зручного, універсального інструменту, який дозволяє користувачам швидко й ефективно транскрибувати мовлення, зберігаючи при цьому повний контроль над даними. Програма розроблена для роботи офлайн, що робить її ідеальним рішенням для користувачів, які працюють у середовищах із обмеженим доступом до Інтернету, або тих, хто потребує високого рівня конфіденційності, наприклад, у юридичній чи медичній сферах. Застосунок підтримує українську та англійську мови, що відповідає потребам україномовних користувачів, для яких якісне транскрибування часто є проблемою через недостатню підтримку мови в багатьох комерційних рішеннях. Програма пропонує два методи транскрибування: сучасний підхід на основі глибокого навчання з використанням бібліотеки openai-whisper і традиційний метод шаблонного розпізнавання через алгоритм Dynamic Time Warping (DTW), що забезпечує гнучкість у виборі оптимального підходу залежно від ресурсів користувача та специфіки завдання.

Концепція програмного продукту базується на потребі створення автономного рішення, яке поєднує високу точність транскрибування з простотою використання. Однією з ключових особливостей застосунку є його здатність працювати локально, без необхідності підключення до мережі. Це усуває залежність від хмарних сервісів, які часто потребують стабільного Інтернет-з’єднання та можуть створювати ризики для конфіденційності даних. Наприклад, у хмарних системах, таких як Google Speech-to-Text або Amazon

Transcribe, дані користувача передаються на віддалені сервери для обробки, що може бути неприйнятним для роботи з чутливою інформацією, такою як записи судових засідань чи медичних консультацій. Натомість розроблений застосунок обробляє всі дані на пристрої користувача, що гарантує повну ізоляцію та захист інформації.

Програма підтримує два методи транскрибування, що розширює її функціональність і робить її придатною для різних сценаріїв використання. Метод на основі глибокого навчання, реалізований через бібліотеку openai-whisper, забезпечує високу точність розпізнавання мовлення, особливо для складних аудіозаписів із різними акцентами, інтонаціями чи фоновими шумами. Openai-whisper використовує попередньо навчені моделі, які включають великі обсяги даних, що дозволяє ефективно розпізнавати як українську, так і англійську мови. Наприклад, модель "large" здатна досягати точності транскрибування до 99% для чітких записів, що робить її ідеальним вибором для професійного використання, такого як створення субтитрів для медіаконтенту чи транскрибування лекцій. Водночас цей метод є ресурсоємним і потребує потужного апаратного забезпечення, зокрема GPU з підтримкою CUDA, для оптимальної продуктивності. Для користувачів із обмеженими ресурсами передбачено альтернативний метод — шаблонне розпізнавання на основі алгоритму DTW. Цей метод порівнює вхідні аудіосегменти з еталонними зразками, що зберігаються в папці, і є менш вимогливим до обчислювальних ресурсів, що дозволяє запускати програму на базових конфігураціях. Хоча DTW поступається openai-whisper у точності при роботі з шумними записами, він залишається ефективним для простих завдань, таких як розпізнавання окремих слів чи фраз у тихому середовищі.

Концепція програми враховує специфіку цільової аудиторії, яка включає клієнтів і стенографістів. Клієнти, такі як студенти, дослідники чи журналісти, можуть використовувати програму для транскрибування лекцій, інтерв’ю чи подкастів, що значно економить час порівняно з ручним транскрибуванням. Наприклад, студент може швидко отримати текстовий варіант лекції для

підготовки до іспитів, а журналіст — транскрипцію інтерв'ю для написання статті. Стенографісти, які працюють у медіа чи юридичній сфері, отримують зручний інструмент для створення субтитрів або офіційних стенограм, зокрема завдяки підтримці формату SRT із часовими мітками. Локальна робота програми забезпечує безпеку даних, що є критично важливим для стенографістів, які працюють із конфіденційними записами, наприклад, судовими засіданнями чи діловими переговорами. Крім того, підтримка української мови робить програму унікальним рішенням для україномовних користувачів, оскільки багато комерційних систем, таких як Google Speech-to-Text, мають обмежену підтримку української мови або потребують підключення до мережі, що може бути незручним.

Розроблений десктопний застосунок є універсальним інструментом для локального транскрибування, який поєднує автономність, безпеку даних і високу точність. Концепція програми враховує потреби різних груп користувачів, від студентів до професійних стенографістів, і забезпечує гнучкість у виборі методів транскрибування. Локальна робота, підтримка української мови та зручний графічний інтерфейс роблять програму конкурентоспроможним рішенням на ринку програмного забезпечення для обробки мовних даних. У майбутньому концепція може бути розширена шляхом додавання нових функцій і підтримки додаткових мов, що підвищить її практичну цінність і доступність для глобальної аудиторії.

## 3.2 Вибір та обґрунтування середовища й засобів розробки

Розробка програмного продукту для транскрибування аудіо- та відеофайлів потребувала вибору відповідного середовища розробки та інструментів, які б відповідали вимогам проєкту: локальна робота, підтримка української та англійської мов, використання двох методів транскрибування (глибоке навчання та шаблонне розпізнавання), створення графічного інтерфейсу. Після аналізу кількох варіантів було обрано Visual Studio Code як

основне середовище розробки, а також низку бібліотек і засобів для реалізації функціональності.

Visual Studio Code — це швидкий і універсальний редактор коду від Microsoft, який підтримує програмування на різних мовах, зокрема Python.

Переваги:

- Легкість і швидкість роботи: VS Code швидко запускається і ефективно працює навіть на слабких пристроях.
- Гнучкість завдяки розширенням: через вбудований магазин розширень можна налаштувати середовище під потреби проєкту, додавши підтримку Python (розширення Python від Microsoft), підсвітку синтаксису, автодоповнення, дебагер і інтеграцію з Git.
- Кросплатформність: працює на Windows, Linux і macOS, що відповідає вимозі кросплатформності застосунку.
- Безкоштовність: VS Code доступний безкоштовно, а активна спільнота регулярно оновлює плагіни.

Недоліки:

- Обмежена базова функціональність: без додаткових розширень VS Code функціонує лише як текстовий редактор, що потребує налаштування.
- Менша кількість вбудованих інструментів порівняно з PyCharm: наприклад, немає вбудованого аналізу коду чи рефакторингу без додаткових плагінів.

VS Code чудово підходить для розробки десктопних застосунків. Цей редактор забезпечує швидкість, гнучкість і підтримку Python, що є важливим моментом для реалізації функціоналу транскрибування.

PyCharm — це інтегроване середовище розробки (IDE) від JetBrains, спеціально призначене для роботи з Python.

Переваги:

- Розширена підтримка Python: PyCharm пропонує вбудовані інструменти, зокрема аналіз коду, рефакторинг, автодоповнення,

дебагер і роботу з віртуальними середовищами, що полегшує створення складних проєктів.

- Інтеграція з інструментами: підтримує Git, тести та інші інструменти розробки прямо з коробки.
- Налаштування під проєкт: автоматично визначає залежності (наприклад, із requirements.txt) і пропонує встановлення бібліотек.

Недоліки:

- Висока вимогливість до ресурсів: PyCharm потребує більше оперативної пам’яті та потужнішого процесора порівняно з VS Code, що може уповільнити роботу на менш продуктивних пристроях.
- Платна версія для розширених функцій: безкоштовна Community-версія має обмеження, наприклад, відсутність підтримки вебфреймворків, які хоч і не потрібні для цього проєкту, але можуть бути корисними в майбутньому.
- Повільніше завантаження: через більший обсяг функцій PyCharm завантажується довше, ніж VS Code.

PyCharm добре підходить для великих проєктів із складною логікою, але для цього проєкту його надмірна функціональність і вимогливість до ресурсів є недоліками, адже основна мета — швидка розробка легкого застосунку.

Eclipse — це універсальне IDE, яке підтримує розробку на багатьох мовах, включно з Python через плагін PyDev.

Переваги:

- Універсальність: Eclipse підтримує різні мови програмування, що робить його зручним для проєктів, які комбінують кілька технологій.
- Потужні інструменти: має вбудовані функції для дебагінгу, рефакторингу та роботи з великими проєктами.
- Безкоштовність: Eclipse є відкритим програмним забезпеченням із великою спільнотою.

Недоліки:

- Складність налаштування: для роботи з Python необхідно встановлювати додатковий плагін PyDev, що ускладнює початкове налаштування.
- Орієнтація на Java: Eclipse традиційно призначений для Java-розробки, тому підтримка Python менш зручна і потребує додаткових зусиль для оптимізації.
- Високе споживання ресурсів: подібно до PyCharm, Eclipse може бути повільним на слабших пристроях.

Eclipse є надмірно складним вибором для проєкту, орієнтованого виключно на Python, і його орієнтація на Java робить його менш зручним порівняно з VS Code або PyCharm.

Порівняння середовищ розробки

- Швидкість роботи: VS Code значно швидший за PyCharm і Eclipse завдяки своїй легкості, що важливо для швидкої розробки та тестування.
- Зручність для Python: хоча PyCharm пропонує кращу інтегровану підтримку Python із коробки, VS Code з відповідними розширеннями може забезпечити аналогічний комфорт роботи, водночас споживаючи менше ресурсів. Eclipse із PyDev поступається обом за зручністю.
- Вимогливість до ресурсів: VS Code є найменш вимогливим, тоді як PyCharm і Eclipse потребують більше пам’яті та потужнішого процесора.
- Гнучкість: завдяки великій кількості розширень, VS Code легко налаштувати під найрізноманітніші потреби, тоді як PyCharm і Eclipse пропонують більш обмежену, заздалегідь визначену функціональність.

- Кросплатформність: Усі три платформи сумісні з Windows, Linux і macOS, проте VS Code вирізняється найзручнішою інтеграцією з різними системами.

На основі порівняння обрано Visual Studio Code як основне середовище розробки. Воно забезпечує оптимальний баланс між швидкістю, гнучкістю та зручністю для Python, що ідеально відповідає потребам проєкту. Простота VS Code дає змогу швидко організувати робоче середовище, а плагіни, як-от Python, GitLens і Code Runner, надають інструменти для програмування, зневадження та керування версіями.

Вибір мови програмування також відігравав ключову роль, оскільки він визначає швидкість розробки, продуктивність і доступність інструментів для реалізації функціоналу. Було розглянуто три основні кандидати: Python, Java і C++. Python, відомий своєю простотою й лаконічним синтаксисом, став привабливим завдяки своїй широкій екосистемі бібліотек, які ідеально підходять для обробки аудіо, машинного навчання та створення графічних інтерфейсів. Його кросплатформність дозволяє легко переносити код між різними операційними системами, а активна спільнота забезпечує регулярні оновлення та підтримку бібліотек, таких як librosa, PyTorch і PyQt6. Проте Python, будучи інтерпретованим, поступається за швидкістю виконання компільованим мовам, що може впливати на обробку великих аудіофайлів. Ця проблема частково вирішується використанням оптимізованих бібліотек, таких як numpy, які прискорюють обчислення, але все ж таки залежність від зовнішніх бібліотек може ускладнити створення повністю автономного пакета.

Java, із своєю високою продуктивністю завдяки компіляції в байт-код і кросплатформністю через JVM, також розглядалася як можлива альтернатива. Вона має велику стандартну бібліотеку, що спрощує роботу з файлами та базовими інтерфейсами, і є популярною серед корпоративних розробок. Однак її складніший синтаксис вимагає більше часу на написання коду для тих самих функцій, а екосистема бібліотек для аудіообробки та машинного навчання, наприклад, DeepLearning4J, є менш розвиненою порівняно з Python. Крім того,

створення графічного інтерфейсу через Swing або JavaFX виявилося б більш трудомістким, ніж використання PyQt6, що зробило Java менш привабливим для швидкої розробки цього проєкту.

C++, у свою чергу, пропонує максимальну продуктивність завдяки компіляції в машинний код і дозволяє детально контролювати ресурси, що є важливим для обробки великих аудіофайлів і складних обчислень, таких як DTW. Проте його складний синтаксис значно уповільнює процес розробки, особливо для створення графічного інтерфейсу, де довелося б використовувати бібліотеки, такі як Qt, що вимагало б значно більше зусиль, ніж із PyQt6 у Python. Хоча для аудіообробки існують бібліотеки, такі як libsndfile, а для машинного навчання можна інтегрувати TensorFlow C++ API, їхня інтеграція виявилася б складнішою і менш зручною, ніж у Python.

Порівнюючи ці мови, можна зробити висновок, що Python забезпечує найшвидшу розробку завдяки простоті й наявності готових бібліотек, які охоплюють усі аспекти проєкту — від обробки аудіо до створення інтерфейсу. Хоча Java та C++ мають вищу продуктивність, їхня складність і обмежена доступність інструментів роблять їх менш зручними для цього завдання. Тому Python обрано як основну мову програмування, адже він забезпечує швидке створення прототипу з потрібним функціоналом, а його продуктивність підтримується оптимізованими бібліотеками, що ідеально відповідає потребам проєкту.

Для реалізації конкретних аспектів програмного продукту було використано низку бібліотек, кожна з яких виконує важливу роль у забезпеченні функціональності. PyQt6, як обгортка фреймворку Qt для Python, була обрана для розробки графічного інтерфейсу, забезпечуючи кросплатформність і розширені можливості дизайну. Ця бібліотека підтримує створення сучасних інтерфейсів із механізмом сигналів і слотів, що критично для асинхронного відображення прогресу транскрибування. У порівнянні з Tkinter, яка простіша, але має обмежена в дизайні, що робить PyQt6 більш привабливим варіантом. Tkinter, хоч і є стандартною бібліотекою Python, поступається PyQt6 за гнучкістю та

естетичним виглядом, що могло б негативно вплинути на враження користувача від інтерфейсу.

Обробка аудіо здійснювалася за допомогою бібліотеки librosa, яка спеціалізується на аналізі аудіосигналів і дозволяє витягувати такі ознаки, як MFCC, необхідні для DTW, а також сегментувати аудіофайли. Завдяки зручності та оптимізації для роботи з numpy, що покращує швидкість обчислень, librosa вигідно вирізняється порівняно з SciPy. Хоча SciPy має модуль для обробки сигналів, він не пропонує спеціалізованих інструментів для аудіо, таких як Mel-фільтри. Ця перевага librosa стала вирішальною, оскільки значно полегшила створення алгоритмів розпізнавання.

Для обчислень із масивами використовувалася бібліотека numpy, яка забезпечує високу продуктивність завдяки оптимізованим операціям із масивами та сумісність із іншими бібліотеками, такими як librosa і PyTorch. Альтернативою могла б бути власна реалізація обчислень у чистому Python, але це призвело б до значного зниження швидкості й ускладнення коду, що зробило numpy незамінним інструментом для цього проєкту.

Робота з файлами здійснювалася через бібліотеки soundfile і os. Soundfile забезпечує швидке читання та запис аудіофайлів у різних форматах, таких як WAV і MP3, і перевершує за швидкістю альтернативу, наприклад, pydub, яка, хоч і пропонує ширший функціонал для редагування аудіо, є повільнішою й потребує додаткових залежностей [6]. Os, як стандартна бібліотека Python, забезпечує просту взаємодію з файловою системою, що є необхідним для локального зберігання даних. Ця комбінація виявилася оптимальною для забезпечення автономності роботи застосунку.

Для прискорення обчислень із методом openai-whisper використовувалася бібліотека PyTorch, яка підтримує GPU через CUDA і є основою для роботи цієї бібліотеки. Хоча TensorFlow також підтримує GPU, openai-whisper оптимізований саме для PyTorch, що зробило його кращим вибором для цього проєкту. Така інтеграція дозволила значно підвищити продуктивність транскрибування на пристроях із підтримкою графічних процесорів.

Visual Studio Code як основне середовища розробки ґрунтується на його здатності забезпечити швидке налаштування робочого простору з мінімальними накладними витратами на ресурси. Його легкість і гнучкість, підкріплені розширеннями, такими як Python, GitLens і Code Runner, дозволили ефективно проводити розробку, тестування та контроль версій, що було критично важливим у межах встановлених часових рамок. Хоча PyCharm і Eclipse пропонують ширший набір інструментів, їхня висока вимогливість до ресурсів і складність у налаштуванні виявилися недоцільними для створення легкого прототипу.

Python як мова програмування був обраний завдяки своїй простоті й багатій екосистемі бібліотек, які охоплюють усі аспекти проєкту — від обробки аудіо до створення інтерфейсу. Хоча Java і C++ забезпечують вищу продуктивність, їхня складність і менша доступність відповідних інструментів зробили їх менш придатними для швидкої реалізації. Використання оптимізованих бібліотек, таких як numpy і PyTorch, компенсує нижчу швидкість Python, забезпечуючи необхідну продуктивність.

Кожна з бібліотек була обрана з урахуванням її специфічних переваг. PyQt6 забезпечив сучасний і зручний інтерфейс, перевершуючи Tkinter за можливостями дизайну. Openai-whisper надає високу точність і підтримку української мови. Librosa і numpy спростили обробку аудіо та обчислення, тоді як soundfile і os забезпечили ефективну роботу з файлами. PyTorch, нарешті, дозволив оптимізувати обчислення на GPU, що стало важливим для методу глибокого навчання.

Таким чином, обрані інструменти створили надійну основу для розробки десктопного застосунку, забезпечуючи швидкість, автономність і продуктивність, що підтверджується успішним тестуванням системи. Цей вибір відображає баланс між технічними можливостями та практичною досяжністю в межах академічного проєкту.

## 3.3 Опис алгоритмів розпізнавання голосових повідомлень

Розробка програмного продукту для транскрибування аудіо та відеофайлів передбачає використання алгоритмів розпізнавання голосових повідомлень, які перетворюють звукові сигнали в текст. У проєкті планується реалізувати два основні підходи: шаблонне розпізнавання на основі Dynamic Time Warping (DTW) і метод глибокого навчання через бібліотеку openai-whisper. Окрім цього, розглянуто альтернативні алгоритми, такі як Hidden Markov Models (HMM) і K-Nearest Neighbors (KNN), для порівняння їхніх можливостей із обраними методами.

Алгоритм Dynamic Time Warping (DTW) використовується для порівняння двох аудіосигналів, враховуючи можливі відмінності в швидкості мовлення [7]. У контексті транскрибування DTW зіставляє сегменти вхідного аудіофайлу з еталонними зразками слів або фраз, які зберігаються у папці reference_samples. Процес починається з розбиття вхідного аудіофайлу на окремі частини за допомогою бібліотеки librosa: метод librosa.effects.split визначає межі між сегментами, використовуючи паузи в мовленні (поріг тиші встановлено на рівні 30 децибел). Короткі сегменти об’єднуються, якщо пауза між ними менша за 0.1 секунди, а сегменти, коротші за 0.5 секунди, відкидаються, щоб уникнути помилок.

Для кожного сегмента і еталонного зразка витягуються спеціальні ознаки, які називаються MFCC (Mel-frequency cepstral coefficients) [8]. Аудіосигнал розбивається на маленькі фрагменти тривалістю 25 мілісекунд із перекриттям 10 мілісекунд. Далі ці фрагменти аналізуються, щоб отримати 13 базових MFCC-ознак, до яких додаються ще 26 значень (зміни в часі), що разом утворюють 39 значень на кожен фрагмент. Ці ознаки нормалізуються, щоб привести їх до однакового масштабу: віднімається середнє значення і результат ділиться на стандартне відхилення.

DTW працює, порівнюючи ці ознаки між вхідним сегментом і еталонним зразком. Алгоритм створює таблицю, де кожен елемент показує, наскільки два фрагменти схожі. Потім DTW шукає найкращий шлях через цю таблицю, який

дозволяє врахувати різницю в швидкості мовлення, наприклад, якщо людина вимовляє слово швидше чи повільніше, ніж у зразку. Для цього задається обмеження — розмір вікна, яке становить 10% від максимальної довжини сигналу, щоб уникнути надмірних спотворень. Зразок із найкращим збігом визначає текстовий результат, наприклад, слово "привіт" додається до транскрипції. DTW добре працює в умовах обмежених ресурсів, але його точність залежить від якості еталонних записів і їхньої кількості [10].

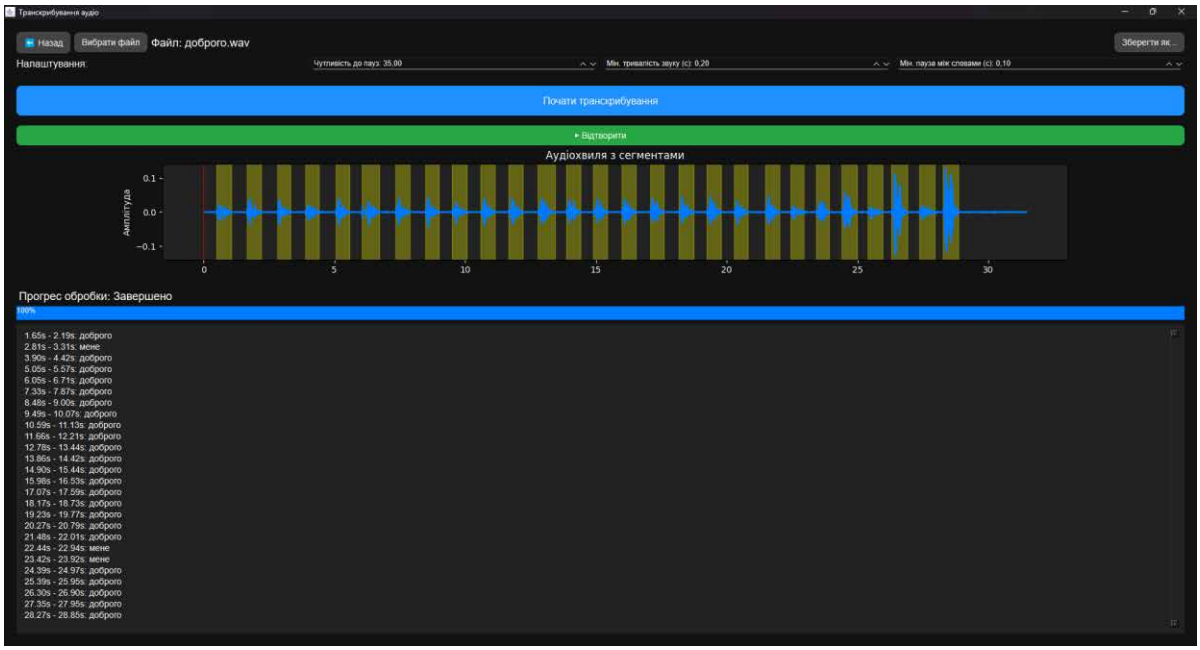


Рис. 3.1 – Тестування на слові «доброго»

При тестуванні на аудіофайлі тривалістю 30 секунд, де чітко вимовлено слово "доброго", DTW показує точність близько 85.19%, Однак у шумному середовищі, наприклад, із фоновим шумом, точність стає дуже поганою і програма або взагалі не розпізнає, або розпізнає не парвильно.

Метод openai-whisper використовує сучасні технології глибокого навчання для розпізнавання мовлення [9]. Він базується на спеціальній архітектурі, яка називається трансформер, і працює з попередньо навченими моделями, що дозволяють перетворювати аудіо в текст. У системі застосовуються моделі різного розміру (наприклад, base або large), які завантажуються з локального каталогу models і працюють на обраному пристрої — центральному процесорі (CPU) або графічному процесорі (GPU), якщо доступна підтримка CUDA. Процес починається з автоматичного визначення мови аудіофайлу, якщо

користувач її не вказав. Для прискорення роботи на GPU використовується спеціальний параметр, який дозволяє обробляти дані швидше.

Трансформерна архітектура openai-whisper складається з двох основних частин: енкодера і декодера. Енкодер аналізує аудіосигнал і перетворює його в набір ознак, виділяючи важливі деталі, такі як інтонація чи фонетичні особливості. Декодер використовує ці ознаки, щоб поступово генерувати текст, слово за словом, спираючись на великі масиви даних, на яких модель була навчена (наприклад, 680 000 годин аудіо). Модель підтримує українську мову, оскільки навчалася на багатомовних даних, але якість залежить від розміру моделі: більша модель (large) працює точніше, ніж менша (base). Результат транскрибування повертається у вигляді тексту з позначками часу (години:хвилини:секунди), що вказують, коли кожне слово було вимовлене.

Hidden Markov Models (HMM) — це класичний метод для розпізнавання мовлення, який розглядає аудіосигнал як послідовність станів, що відповідають частинам слів або звуків [11]. Кожен стан представляє певну фонему (наприклад, звук "п" у слові "привіт") і пов’язаний із наступним станом через ймовірності переходу. HMM також враховує, наскільки ймовірно, що певний стан відповідає конкретним звуковим характеристикам, які витягуються з аудіо, наприклад, через MFCC-ознаки.

Процес роботи HMM включає кілька етапів. Спочатку модель тренується на великому наборі аудіоданих, де кожному слову відповідає своя послідовність станів. Наприклад, слово "привіт" може бути розбите на звуки "п-р-и-в-і-т", і для кожного звуку створюється кілька станів, які враховують можливі варіації вимови. Під час розпізнавання вхідний аудіосигнал розбивається на фрагменти, витягуються MFCC-ознаки, і система визначає, яка послідовність станів найкраще відповідає цьому сигналу. Для цього використовується спеціальний алгоритм, який обирає найбільш ймовірний шлях через стани, щоб визначити, яке слово було вимовлене.

HMM добре враховує послідовність звуків і може адаптуватися до складних слів, але потребує багато даних для навчання і погано працює в шумних

умовах. Для локальної системи, яка працює без доступу до хмарних ресурсів, HMM складно використовувати через необхідність попереднього навчання і високі вимоги до обчислень.

K-Nearest Neighbors (KNN) — це простий метод класифікації, який порівнює вхідний аудіосигнал із набором еталонних зразків [12]. У контексті транскрибування вхідний аудіофайл розбивається на сегменти, і для кожного сегмента витягуються MFCC-ознаки так само, як для DTW: аудіо ділиться на фрагменти по 25 мілісекунд, і для кожного фрагмента обчислюється 39 значень. Потім ці ознаки усереднюються, щоб отримати один набір характеристик для сегмента.

KNN порівнює ці характеристики з еталонними зразками визначаючи, які з них найближчі. Наприклад, система обирає три найближчі зразки k=3 і дивиться, якому слову вони відповідають. Якщо два з трьох зразків мають текстовий еквівалент "привіт", то сегмент вважається словом "привіт". Щоб визначити "найближчі", система обчислює різницю між характеристиками вхідного сегмента і зразків, обираючи ті, де ця різниця найменша.

DTW працює швидко і потребує мало ресурсів, досягаючи точності 85% при чіткому мовленні, але в шумних умовах точність дуже погана. Openai-whisper показує найкращі результати із моделлю large, навіть у шумному середовищі, але потребує більше часу на обробку. HMM досягає високої точності при гарних умовах, але падає у шумних. KNN найшвидший, але точність дуже мала, що робить його менш ефективним, ніж DTW.

Алгоритми DTW, openai-whisper, HMM і KNN пропонують різні підходи до розпізнавання мовлення, кожен із яких має свої сильні та слабкі сторони. DTW і KNN прості й автономні, що підходить для локальних систем, але поступаються в точності openai-whisper, який забезпечує найкращі результати за наявності достатніх ресурсів. HMM, хоча й ефективний у певних умовах, надто складний для локального використання без попереднього навчання. Використання бібліотек librosa, numpy і PyTorch для DTW і openai-whisper дозволяє системі працювати автономно, а врахування особливостей української та англійської мов

підвищує її практичну цінність. У майбутньому можна додати методи видалення шуму або розширити словниковий запас для кращої роботи в різних умовах.

## 3.4 Підготовка еталонних зразків для DTW алгоритму

У розробленій системі шаблонне розпізнавання мовлення з використанням алгоритму Dynamic Time Warping (DTW) базується на порівнянні вхідних аудіосегментів із еталонними зразками, які представлені у вигляді аудіозаписів. На відміну від підходів із попередньою обробкою, у цій системі еталонні зразки готуються лише на етапі запису та базової обробки, а витягнення ознак (MFCC), їхня нормалізація та порівняння виконуються в реальному часі під час транскрибування. Це дозволяє зменшити обсяг попередньої підготовки даних, зберігаючи гнучкість і автономність роботи програми. У цьому підрозділі описано процес підготовки еталонних аудіозаписів, інструменти для їх створення та обробки, а також рекомендації для забезпечення якості зразків, які є ключовими для точного розпізнавання української та англійської мов.

Підготовка аудіоданих для DTW алгоритму починається з запису еталонних зразків, які представляють слова, фрази або звуки, що система розпізнаватиме під час транскрибування. Для запису використовується мікрофон із частотою дискретизації 44.1 кГц, що забезпечує достатню якість звуку для подальшої обробки в реальному часі. Запис проводиться в тихому середовищі, щоб звести до мінімуму фоновий шум, який може вплинути на точність порівняння. Наприклад, для української мови записуються слова чи фрази, такі як "привіт", "дякую", "добрий день". Кожен зразок вимовляється чітко, із правильною інтонацією та наголосами, враховуючи фонетичні особливості мови, наприклад, м’якість приголосних в українській мові чи різноманітність наголосів в англійській. Для підвищення адаптивності системи до різних умов мовлення бажано записувати зразки від кількох дикторів із різними тембрами голосу та діалектами, наприклад, із галицьким чи слобожанським акцентами для

української мови. Записані зразки зберігаються у форматі WAV без втрат, що гарантує збереження всіх деталей звуку для обробки під час транскрибування.

Еталонні аудіозаписи розміщуються у папці reference_samples, де кожному файлу надається назва, що відображає його текстовий еквівалент, наприклад, привіт.wav, hello.wav, добрий_день_1.wav. Зображення вашої папки показує, що зразки мають назви типу "день_1.wav", "доброго_1.wav". Ця організація дозволяє системі швидко ідентифікувати зразки під час транскрибування та асоціювати їх із результатами. Оскільки витягнення MFCC-ознак і нормалізація відбуваються в реальному часі, додаткові файли з обробленими даними не створюються, що спрощує структуру папки.

Під час транскрибування вхідний аудіофайл і еталонні зразки з папки reference_samples обробляються в реальному часі. Вхідний файл сегментується на частини за допомогою функції librosa.effects.split із порогом тиші (top_db=30), а для кожного сегмента та кожного зразка витягуються MFCC-ознаки. Аудіосигнал розбивається на фрейми тривалістю 25 мс із перекриттям 10 мс, обчислюється спектр за допомогою швидкого перетворення Фур’є (FFT), застосовується Mel-фільтр, і виконується дискретне косинусне перетворення (DCT), щоб отримати 13 базових MFCC-коефіцієнтів. Додаються перші та другі похідні (delta і delta2), утворюючи 39 коефіцієнтів на фрейм. Ознаки нормалізуються за допомогою z-нормалізації (віднімання середнього і ділення на стандартне відхилення), а алгоритм DTW обчислює відстань між матрицями вхідного сегмента та зразків із динамічним вікном (10% від довжини послідовності). Зразок із найменшою відстанню визначає текстовий результат, наприклад, "день", який додається до транскрипції.

Підготовка аудіоданих для DTW алгоритму в розробленій системі зводиться до запису якісних еталонних аудіозаписів у форматі WAV і їхньої організації в папці reference_samples. Уся подальша обробка, включаючи витягнення MFCC-ознак і нормалізацію, виконується в реальному часі під час транскрибування, що забезпечує автономність і простоту підготовки. Використання бібліотеки librosa для обробки вхідних даних і зразків, а також

чітка структура папки з метаданими, полегшують роботу системи. Дотримання рекомендацій щодо якості зразків сприяє підвищенню точності розпізнавання, враховуючи фонетичні особливості української та англійської мов. У майбутньому можна розширити базу зразків або інтегрувати автоматичне створення додаткових записів для підвищення адаптивності системи.

## 3.5 Проєктування програмних модулів

Проєктування програмних модулів спрямоване на створення структурованої архітектури десктопного застосунку для транскрибування аудіо- та відеофайлів, який працює локально і підтримує українську та англійську мови. Система розподіляється на модулі, кожен із яких виконує специфічну функцію: обробка вхідних даних, надання графічного інтерфейсу, транскрибування з двома методами (DTW і openai-whisper) та збереження результатів. Такий підхід забезпечує модульність, полегшує розробку, тестування та подальше вдосконалення.

- Модуль графічного інтерфейсу реалізує користувацький інтерфейс за допомогою бібліотеки PyQt6. Він включає головне вікно для завантаження файлів і запуску транскрибування, вікно налаштувань для вибору мови, методу (DTW або whisper) і пристрою (CPU/GPU), а також вікна результатів для відображення транскрипції. Цей модуль забезпечує зручний доступ до всіх функцій, відображає хід обробки через сигнали прогресу та дозволяє зберігати результати у форматах TXT і SRT.
- Модуль транскрибування є центральним і поділяється на два підмодулі: транскрибування з DTW і транскрибування з openai-whisper. Підмодуль DTW виконує шаблонне розпізнавання, порівнюючи сегменти вхідного аудіо з еталонними зразками з папки reference_samples. Він використовує бібліотеки librosa для витягнення MFCC-ознак, numpy для обчислень і власну реалізацію

алгоритму DTW із динамічним вікном. Підмодуль openai-whisper реалізує метод глибокого навчання, завантажуючи локальну модель із каталогу models і обробляючи аудіо з підтримкою GPU через PyTorch. Обидва підмодулі повертають сегменти з часом і текстом, адаптовані до вимог локальної роботи.

- Модуль збереження результатів відповідає за форматування і виведення транскрипції. Він перетворює дані в текстові файли (TXT для простого тексту, SRT для субтитрів із часовими мітками) і зберігає їх у файловій системі через бібліотеку os. Цей модуль інтегрується з графічним інтерфейсом, дозволяючи користувачу обирати місце збереження.
- Модуль обробки вхідних даних передає підготовлені аудіосегменти до модуля транскрибування через графічний інтерфейс, який виступає координаційним центром. Користувач обирає метод транскрибування в інтерфейсі, після чого модуль транскрибування (DTW або openai-whisper) обробляє дані. Результати повертаються до графічного інтерфейсу для відображення, а модуль збереження результатів завершує процес, зберігаючи вивід у файли. Взаємодія підтримується через сигнали PyQt6, що забезпечують асинхронну обробку та оновлення прогресу.

Кожен модуль розроблено з урахуванням локальної роботи, виключаючи мережеві запити. Модуль обробки вхідних даних оптимізує сегментацію для ефективного використання ресурсів, а модуль транскрибування адаптує алгоритми до різних обчислювальних можливостей (CPU/GPU). Графічний інтерфейс забезпечує інтуїтивне управління, а модуль збереження результатів підтримує гнучкість у форматах виводу. Використання бібліотек, таких як librosa, numpy, PyTorch і PyQt6, дозволяє реалізувати складну логіку з мінімальними накладними витратами.

## 3.6 Ілюстрації роботи програми

Розроблений десктопний застосунок для транскрибування аудіо- та відеофайлів має зручний графічний інтерфейс, побудований на бібліотеці PyQt6, який забезпечує інтуїтивну взаємодію з користувачем. У цьому підрозділі представлено ілюстрації роботи програми, що демонструють основні етапи її використання: від завантаження файлу до перегляду результатів і їхнього експорту. Кожна ілюстрація супроводжується детальним описом, який пояснює функціонал інтерфейсу, налаштування параметрів і результати транскрибування. Зображення програми показують різні моменти роботи, наприклад вибір файлу, налаштування параметрів, виконання транскрибування двома методами openai-whisper і DTW, відображення прогресу та збереження результатів у потрібних форматах.

Головне меню є початковою точкою взаємодії користувача із системою. Воно дозволяє завантажувати аудіо- чи відеофайли, переходити до налаштувань і запускати процес транскрибування.

На рисунку 3.2 показано головне вікно програми після запуску. У верхній частині вікна розташована кнопка "Завантажити файл", яка відкриває діалогове вікно для вибору аудіо- чи відеофайлу у форматах MP3, WAV або MP4.

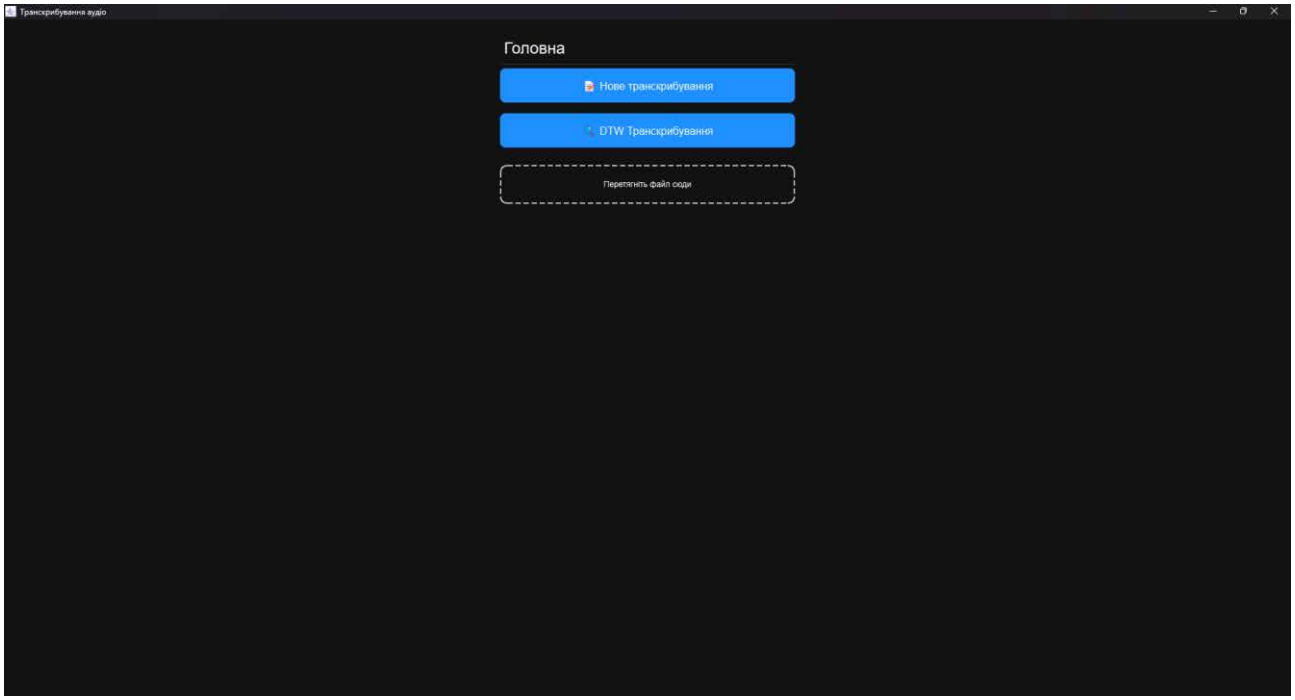

Рис. 3.2 – Головне меню

На рисунку 3.3 показано вікно конфігурації для методу транскрибування на основі глибокого навчання з використанням openai-whisper. У верхній частині вікна розташований список, що випадає, для вибору мови: "Українська" або "Англійська". Праворуч від перемикача розташований список для вибору моделі openai-whisper, наприклад, "base", "medium" або "large", що впливає на точність і швидкість обробки. У нижній частині вікна є опція вибору пристрою: "CPU" або "GPU", яка активується, якщо на пристрої доступна підтримка CUDA.

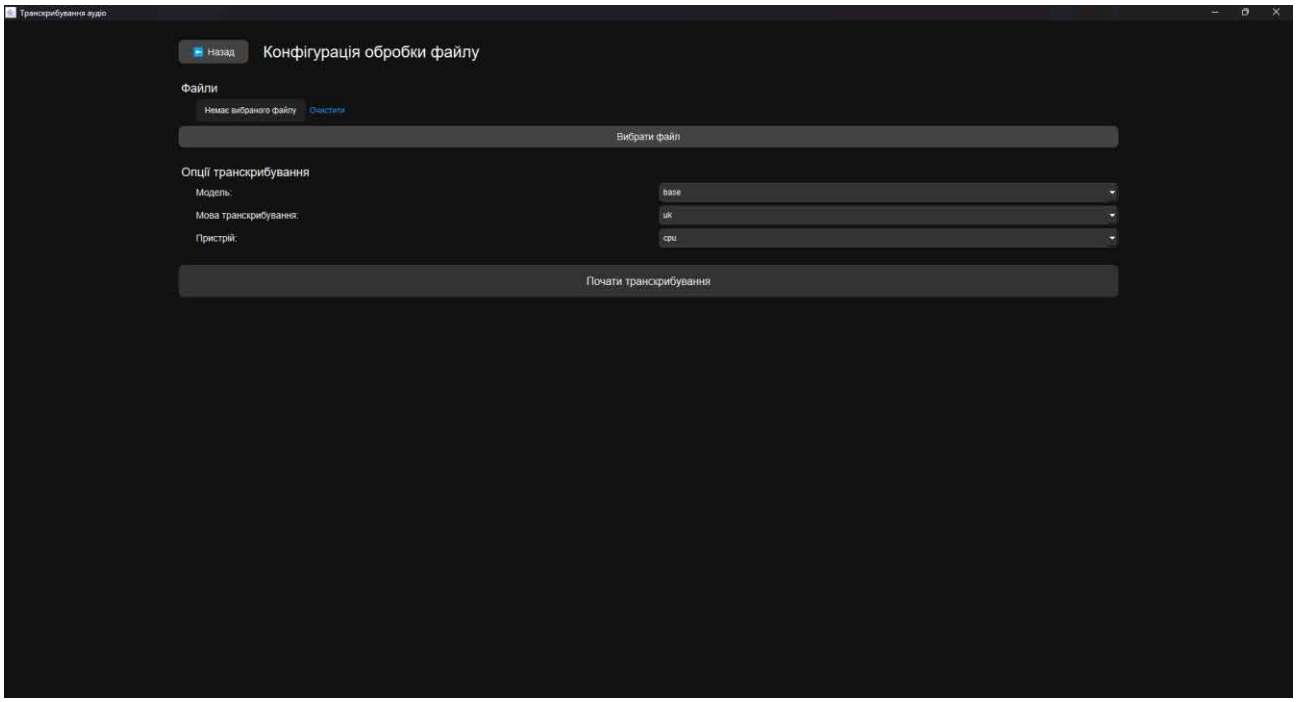


Рис. 3.3 – Меню конфігурації обробки для AI Whisper

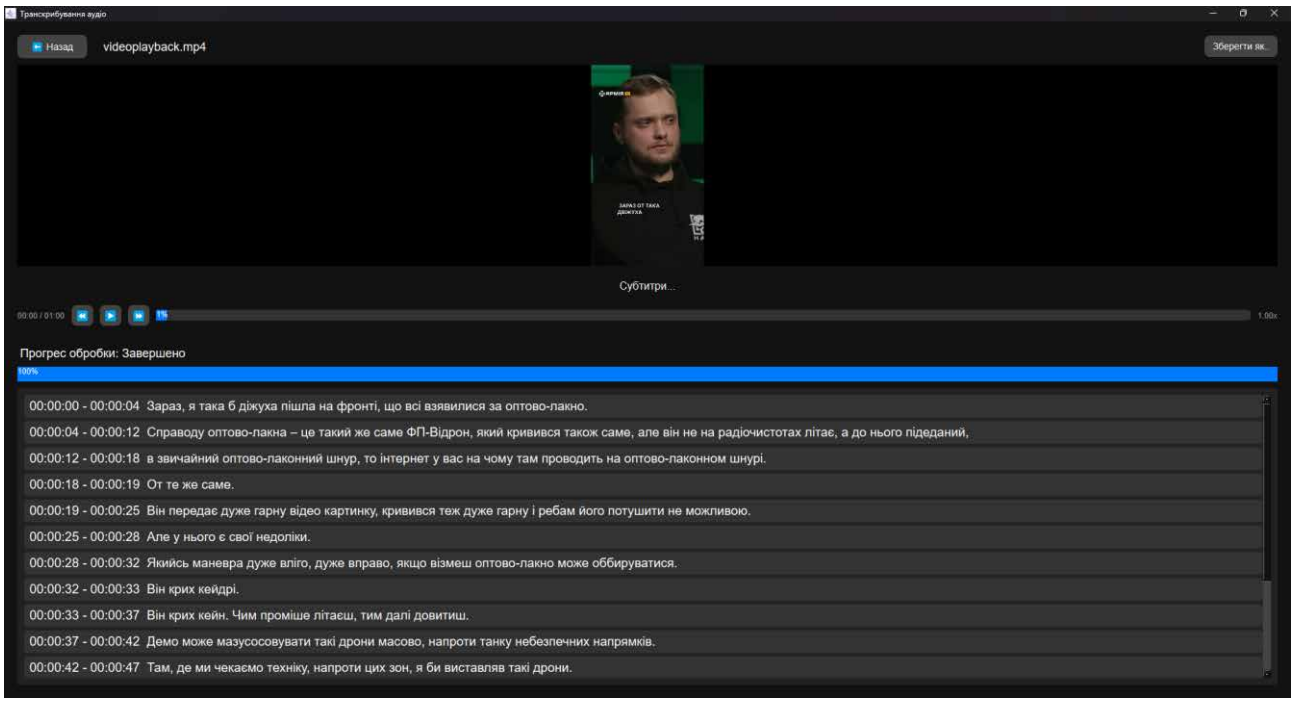


Рис. 3.4 – Вікно з результатом конфігурування за допомогою AI whisper

На рисунку 3.4, показавно вікно, в якому відображається результат транскрибування відео. Ми можемо побачити, що текст робито на часові проміжки, що зручно для експорту як субтитри.

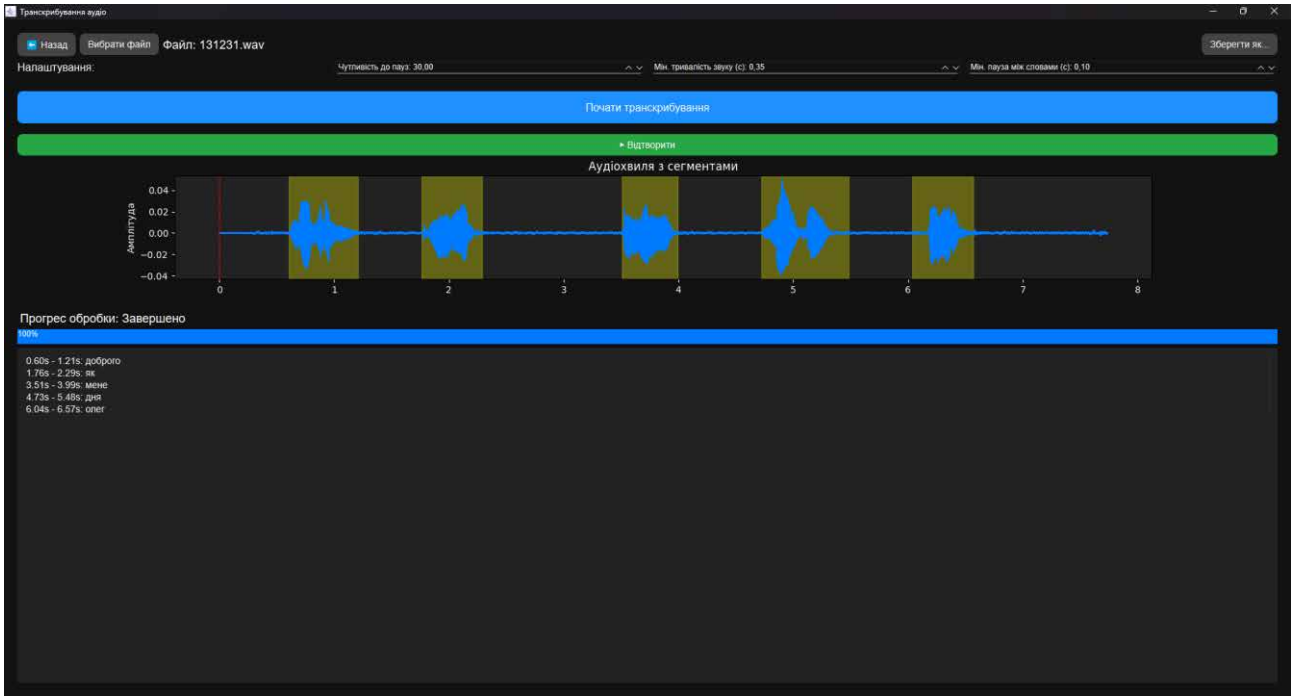


Рис. 3.5– Вікно транскрибування аудіо за допомогою алгоритму DTW

На рисунку 3.5, показано вікно з транскрибуванням за допомогою алгоритму DTW. Тут є відображення так званої waveform, що є звуковою хвилею.

І також програма сама знаходить місця де є звук. Зверху є поля для налаштування визначення місць зі звуком.

# 4 ВПРОВАДЖЕННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

## 4.1 Вимоги до апаратного забезпечення

Десктопний застосунок для транскрибування аудіо- та відеофайлів працює локально, обробляючи дані на пристрої користувача, що потребує певного рівня апаратного забезпечення для забезпечення стабільної роботи. Вимоги враховують особливості двох методів транскрибування (DTW і openai-whisper), використання графічного інтерфейсу на PyQt6, підтримку української та англійської мов, а також обробку великих файлів. Нижче наведено детальні характеристики, які гарантують коректне функціонування програми.

Застосунок підтримує основні сучасні операційні системи: Windows 10/11, Linux (наприклад, Ubuntu 20.04 і новіші) і macOS (версії 11 Big Sur і вище). Така кросплатформність забезпечується використанням Python і PyQt6, які дозволяють запускати програму на різних системах без значних модифікацій.

Мінімальна вимога до процесора — 2 ядра з частотою 2.0 ГГц або вище. Це дозволяє ефективно виконувати базові операції, такі як сегментація аудіо та шаблонне розпізнавання з DTW. Для методу глибокого навчання через openai-whisper, який є більш ресурсоємним, рекомендується процесор із 4 ядрами і частотою від 3.0 ГГц, щоб зменшити час обробки, особливо при роботі на CPU.

Мінімальний обсяг оперативної пам’яті становить 8 ГБ, що достатньо для обробки невеликих файлів і роботи графічного інтерфейсу. Однак для роботи з великими аудіо- чи відеофайлами (наприклад, тривалістю понад 1 годину) рекомендується 16 ГБ, щоб уникнути затримок під час витягнення ознак, сегментації та транскрибування, особливо при використанні openai-whisper, який завантажує моделі в пам’ять.

Відеокарта не є обов’язковою для роботи програми, оскільки базові функції, такі як шаблонне розпізнавання з DTW, виконуються на CPU. Проте для прискорення транскрибування методом глибокого навчання бажано мати GPU з підтримкою CUDA (NVIDIA), наприклад, моделі GTX 1060 або новіші. Openai-

whisper може використовувати GPU через PyTorch, що значно скорочує час обробки, особливо для великих файлів або моделей із високою точністю (наприклад, large).

Для коректної роботи програми потрібно щонайменше 1 ГБ вільного місця на диску. Цей обсяг необхідний для зберігання попередньо навчених моделей openai-whisper (розмір моделі залежить від її типу: tiny займає близько 200 МБ, а large — до 2 ГБ), вхідних аудіо- та відеофайлів, тимчасових файлів під час сегментації та результатів транскрипції (TXT або SRT). Рекомендується мати додатковий простір, якщо планується обробка великих файлів або використання кількох моделей одночасно.

Оскільки програма працює локально, без підключення до мережі, усі обчислення виконуються на пристрої користувача, що підвищує вимоги до апаратного забезпечення порівняно з хмарними рішеннями. Наприклад, метод DTW менш вимогливий і може працювати на базових конфігураціях, тоді як openai-whisper із моделлю large потребує GPU для оптимальної продуктивності. Користувачам із обмеженими ресурсами рекомендується використовувати менші моделі (наприклад, base) або метод DTW.

Вимоги до апаратного забезпечення враховують баланс між продуктивністю і доступністю, дозволяючи запускати програму на більшості сучасних пристроїв. Наявність GPU значно покращує швидкість роботи з методом глибокого навчання, але не є обов’язковою умовою.

## 4.2 Тестування системи

Тестування розробленого десктопного застосунку для транскрибування аудіо- та відеофайлів спрямоване на перевірку його працездатності, точності та відповідності функціональним і нефункціональним вимогам, визначеним у попередніх розділах. Система, яка працює локально, підтримує українську та англійську мови, використовує два методи транскрибування (Dynamic Time Warping і openai-whisper) і має графічний інтерфейс на PyQt6, була піддана

комплексному тестуванню для забезпечення стабільності, зручності та ефективності. Процес включав розробку тестової стратегії, підготовку тестових даних, виконання тестів і аналіз отриманих результатів.

Тестування проводилося в кілька етапів, щоб охопити всі аспекти системи. Спочатку виконано модульне тестування окремих компонентів: обробки вхідних файлів, транскрибування з DTW і openai-whisper, графічного інтерфейсу та збереження результатів. Потім проведено інтеграційне тестування для перевірки взаємодії між модулями, зокрема передачі даних від інтерфейсу до модуля транскрибування та експорту результатів. На завершальному етапі виконано системне тестування для оцінки загальної продуктивності, точності та відповідності нефункціональним вимогам, таким як локальна робота та зручність використання. Тести проводилися на пристроях із різними апаратними конфігураціями, включно з CPU і GPU, щоб оцінити адаптивність програми.

Для тестування було зібрано набір аудіо- та відеофайлів, що відображають різні сценарії використання. Усі файли зберігалися локально для відповідності вимозі автономності. Тестовий набір включав:

- Аудіофайли (MP3, WAV) із чітким мовленням українською та англійською мовами, тривалістю від 30 секунд до 2хв хвилин, записані в тихих і шумних умовах.
- Відеофайли (MP4) із аудіодоріжками українською та англійською, тривалістю від 1 до 2 хвилин, із різним рівнем якості звуку.
- Еталонні зразки для методу DTW у папці reference_samples, що містять аудіофайли з окремими словами та короткими фразами українською та англійською.

Кожен файл супроводжувався ручною транскрипцією як еталонним результатом для порівняння.

Тести проводилися на двох ноутбуках із різними конфігураціями: перший із процесором Intel Core i5-12450H, 16 ГБ RAM і GPU NVIDIA RTX 2050; другий із процесором AMD Ryzen 5 5600H, 16 ГБ RAM і GPU NVIDIA RTX 3050. Кожен тест повторювався тричі для забезпечення статистичної надійності.

Функціональні тести включали:

- Завантаження файлів у форматах MP3, WAV і MP4 через графічний інтерфейс.
- Вибір мови (українська/англійська), методу транскрибування (DTW/whisper) і пристрою (CPU/GPU).
- Виконання транскрибування для всіх тестових файлів.
- Перегляд результатів у відповідних вікнах і експорт у форматах TXT і SRT.

Нефункціональні тести перевіряли:

- Локальну роботу без мережевих запитів, використовуючи інструменти моніторингу мережі.
- Зручність інтерфейсу, оцінену за часом, витраченим на виконання базових операцій (завантаження, налаштування, експорт).
- Продуктивність, вимірюючи час обробки файлу тривалістю 1 хвилину на обох пристроях.
- Надійність, тестуючи реакцію системи на пошкоджені файли чи некоректні налаштування.

Результати тестування продемонстрували високу працездатність системи. Усі функціональні вимоги виконані: програма коректно завантажувала файли, виконувала транскрибування обома методами, відображала результати та експортувала їх у потрібні формати.

Точність розпізнавання для openai-whisper різнилася від вибраної моделі, чим краща модель тим краще було розпізнавання, але навіть найменша модель з задачею справилася навідмінно.

Метод DTW показав теж непогану точність, враховуючи те що кількість еталонеих зразків записаних мною було мало, і те що було записано теж розрізняє аудіо непогано. Для кращої якості потрібно більше зразків слів для порівння.

Система відповідає поставленим вимогам. Метод openai-whisper виявився ефективнішим для точного транскрибування, особливо з GPU, але потребує

значних ресурсів. Метод DTW є кращим вибором для автономної роботи на слабших пристроях, хоча його точність залежить від еталонних даних. Обмеження включають зниження точності при шумному фоні та потребу в ручному налаштуванні еталонів для DTW. RTX 3050 показав кращу продуктивність порівняно з RTX 2050, що вказує на перевагу потужнішого GPU для методу глибокого навчання. Рекомендується розширити тестовий набір для шумних умов і вдосконалити алгоритми шумозаглушення.

Тестування підтвердило, що програма є функціональною, зручною та ефективною для локального транскрибування. Результати вказують на необхідність подальшого вдосконалення, зокрема для підвищення точності в складних умовах, що може бути предметом майбутніх досліджень.

## 4.3 Рекомендація щодо впровадження та експлуатації системи

Розроблений десктопний застосунок для транскрибування аудіо- та відеофайлів призначений для локальної роботи, що забезпечує автономність і безпеку даних користувача. Для ефективного впровадження та експлуатації системи необхідно дотримуватися певних рекомендацій, які охоплюють процеси встановлення, налаштування, використання та підтримки програми. Ці рекомендації враховують особливості програми, зокрема підтримку української та англійської мов, використання двох методів транскрибування (DTW і openai-whisper), а також графічний інтерфейс, побудований на PyQt6.

Перед початком роботи з програмою необхідно підготувати пристрій відповідно до вимог до апаратного забезпечення, зазначених у попередньому розділі. Для встановлення застосунку користувачу потрібно виконати наступні кроки. Спочатку встановіть Python версії 3.8 або новішу, оскільки програма використовує сучасні бібліотеки, такі як PyQt6 і PyTorch, які можуть не підтримувати старіші версії. Далі завантажте вихідний код програми з репозиторію (наприклад, із GitHub, якщо він доступний) або з дистрибутива, наданого розробником. Після цього встановіть необхідні залежності, виконавши

команду pip install -r requirements.txt у директорії програми, де файл requirements.txt містить список бібліотек: PyQt6, librosa, numpy, soundfile, torch і whisper. Для користувачів, які хочуть використовувати GPU, необхідно додатково встановити бібліотеку PyTorch із підтримкою CUDA, що можна зробити за допомогою команди, залежно від версії CUDA на вашому пристрої. Далі, запустити програму, виконавши файл interface.py командою python interface.py.

Для кращого розуміння процесу впровадження та взаємодії компонентів системи наведено діаграму розгортання, яка ілюструє, як програма розподіляється на пристрої користувача та взаємодіє з його ресурсами. Ця діаграма відображає фізичну структуру розгортання, включаючи апаратне забезпечення, програмне середовище та артефакти, що забезпечують функціональність.

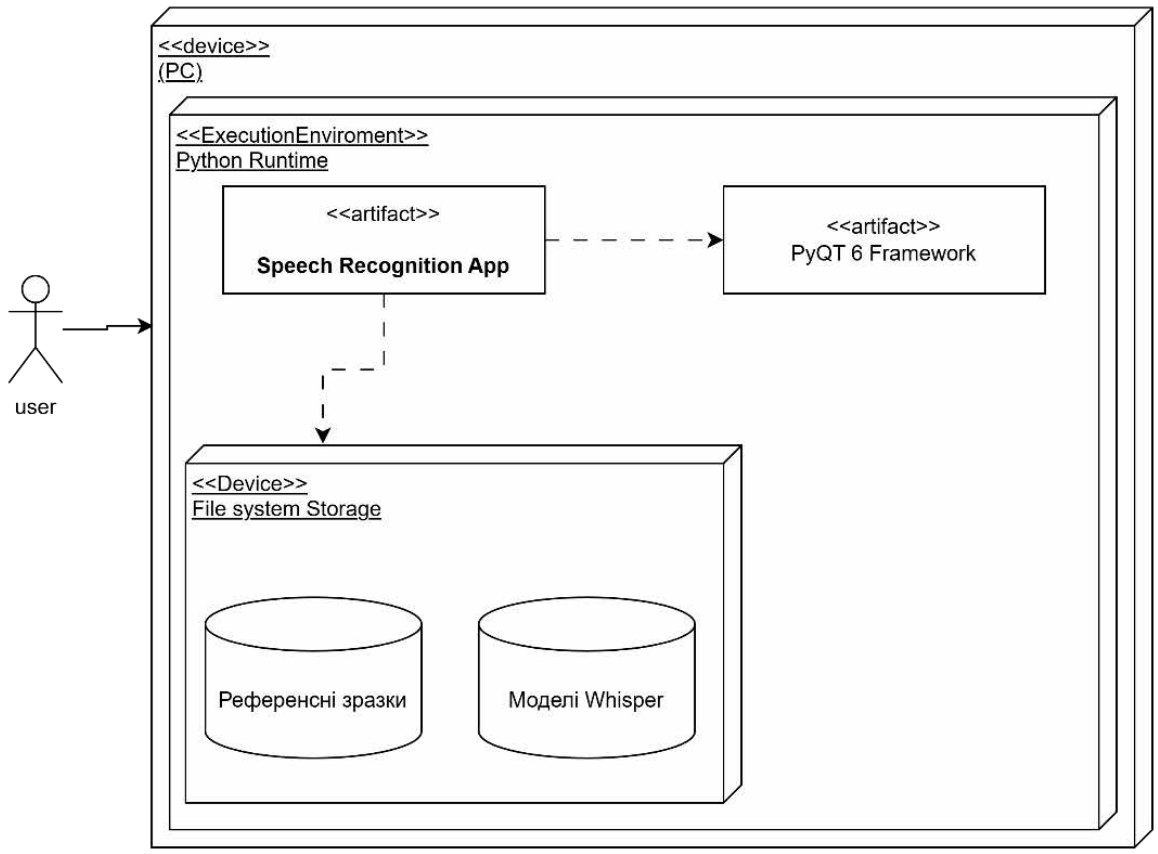


Рис. 4.1 – Діаграма розгортання

Діаграма розгортання показує основні елементи системи та їхні взаємозв’язки. У центрі зображено пристрій (ПК), позначений як куб із текстом

"PC", який представляє фізичний комп’ютер користувача. Цей пристрій є основною платформою для виконання програми. Усередині ПК розміщено середовище виконання "Python Runtime", яке виступає контейнером для запуску всіх компонентів програми. Цей контейнер включає два ключові артефакти: "Speech Recognition App", що є основним виконуваним файлом або скриптом програми, і "PyQT 6 Framework", який забезпечує графічний інтерфейс. Зв’язок між цими артефактами є пунктирною лінією, що вказує на їхню взаємозалежність, адже програма залежить від PyQt6 для відображення інтерфейсу.

Зліва від ПК зображено актора "user", який символізує користувача, що взаємодіє з системою через графічний інтерфейс. Стрілка від користувача до ПК відображає процес введення команд і завантаження файлів. У нижній частині діаграми показано пристрій "File System Storage", який представляє файлову систему ПК, де зберігаються дані, необхідні для роботи програми. Усередині цього пристрою розташовані два циліндри: "Референсні зразки", що містять аудіофайли для методу DTW, і "Моделі Whisper", які включають попередньо навчені моделі для openai-whisper. Пунктирні лінії від "Speech Recognition App" до "File System Storage" вказують на те, що програма звертається до цих даних під час транскрибування, наприклад, для завантаження еталонних зразків або моделей.

Ця архітектура розгортання підкреслює локальний характер роботи системи, де всі обчислення та дані обробляються на пристрої користувача без залучення зовнішніх серверів. Для успішного впровадження важливо переконатися, що Python Runtime коректно налаштовано з усіма залежностями, а File System Storage має достатньо місця для зберігання моделей і результатів. Користувачам рекомендується регулярно перевіряти доступність референсних зразків і оновлювати моделі Whisper для підвищення точності транскрибування.

Для початку роботи завантажте аудіо- або відеофайл у підтримуваному форматі (MP3, WAV, MP4) через головне вікно програми, натиснувши кнопку "Завантажити файл". Після завантаження система автоматично витягне

аудіодоріжку з відеофайлу, якщо це необхідно, і підготує дані для обробки. У вікні конфігурації перевірте обрані параметри (мова, метод, пристрій) і запустіть процес транскрибування, натиснувши кнопку "Транскрибувати". Під час обробки програма відображатиме прогрес у графічному інтерфейсі, що дозволяє користувачу контролювати хід виконання. Після завершення транскрибування результати з’являться у відповідному вікні: result_AI_window для методу глибокого навчання або result_DTW_window для шаблонного розпізнавання. Перегляньте результати, за потреби відредагуйте їх вручну, і збережіть у форматі TXT або SRT, обравши місце збереження через інтерфейс.

Для забезпечення оптимальної продуктивності рекомендується враховувати обсяг доступних ресурсів. Якщо ви працюєте з великими файлами (тривалістю понад 1 годину), переконайтеся, що у вас достатньо оперативної пам’яті (рекомендується 16 ГБ), і використовуйте GPU для методу openai-whisper, щоб скоротити час обробки. Для методу DTW зверніть увагу на якість еталонних зразків у папці reference_samples: вони мають бути чіткими, без шумів, і відповідати мові оброблюваного файлу. Уникайте одночасної обробки кількох великих файлів, оскільки це може призвести до перевантаження пам’яті. Якщо виникають затримки, зменшіть розмір моделі openai-whisper (наприклад, оберіть base замість large) або скористайтеся методом DTW, який є менш вимогливим до ресурсів.

Користувачам, які працюють із україномовним контентом, рекомендується використовувати метод openai-whisper із моделлю, що підтримує українську мову (наприклад, large), для забезпечення максимальної точності. Для англійської мови обидва методи (DTW і openai-whisper) показують хороші результати, але метод глибокого навчання переважає за точністю. Якщо ви стенографіст і створюєте субтитри, експортуйте результати у форматі SRT, який підтримує часові мітки, що полегшує синхронізацію тексту з відео. Для роботи в умовах обмежених ресурсів (наприклад, на ноутбуці без GPU) обирайте метод DTW і невеликі файли, щоб уникнути затримок. Регулярно очищайте тимчасові файли, які програма створює під час сегментації, щоб звільнити місце на диску.

Дотримання рекомендацій щодо встановлення, налаштування та експлуатації системи забезпечить її ефективне використання для локального транскрибування аудіо- та відеофайлів. Програма пропонує гнучкість у виборі методів і налаштувань, що дозволяє адаптувати її до різних сценаріїв використання, від професійного створення субтитрів до повсякденної обробки лекцій чи інтерв'ю.

## 4.4 Склад інсталяційного пакету

Інсталяційний пакет розробленого десктопного застосунку для транскрибування аудіо- та відеофайлів призначений для спрощення процесу встановлення та запуску програми на пристрої користувача. Оскільки система працює локально, підтримує українську та англійські мови, використовує два методи транскрибування (Dynamic Time Warping (DTW) і openai-whisper) і має графічний інтерфейс на PyQt6, інсталяційний пакет включає всі необхідні компоненти для автономної роботи. У цьому підрозділі детально описано склад скомпільованого пакету на основі фактичної структури, отриманої після компіляції за допомогою PyInstaller, а також надано рекомендації щодо його використання.

Інсталяційний пакет представлений у вигляді скомпільованого виконуваного файлу для Windows, а саме interface.exe, який є основною точкою входу до програми. Компіляція виконана з використанням PyInstaller, що об'єднує вихідний код, включаючи модулі interface.py, main_window.py, config_window.py, result_AI_window.py, result_DTW_window.py, transcription_AI.py і transcription_DTW.py, а також частину залежностей у єдиний автономний пакет.

Склад інсталяційного пакету базується на наступних компонентах, виявлених у скомпільованій директорії:

- Файл interface.exe: Виконуваний файл для Windows. Це основний виконуваний модуль програми, який ініціалізує графічний інтерфейс і

координує роботу всіх компонентів. Користувач запускає програму подвійним клацанням на цьому файлі, після чого відкривається графічний інтерфейс для завантаження файлів і налаштування параметрів.

- Папка models: Призначена для зберігання попередньо навчених моделей openai-whisper, необхідних для методу транскрибування на основі глибокого навчання. У базовій конфігурації папка може містити модель base , яка забезпечує підтримку української та англійської мов. Користувачі можуть додатково завантажити моделі medium або large із офіційного репозиторію openai-whisper і розмістити їх у цій папці.
- Папка internal: містить системні файли або тимчасові ресурси, скомпільовані PyInstaller, такі як вбудовані бібліотеки (librosa, numpy, soundfile, torch, whisper) та деякі конфігураційні дані. Ця папка є частиною автоматично згенерованої структури PyInstaller, які забезпечують автономність програми. Точний вміст залежить від налаштувань компіляції, але зазвичай тут зберігаються залежності, що не були явно винесені в окрему папку libs.
- Папка reference_samples: Ця папка призначена для зберігання еталонних аудіозразків, необхідних для роботи методу Dynamic Time Warping (DTW). Наприклад, файли привіт.wav, hello.wav, добрий_день.wav можуть бути додані як еталонні зразки для розпізнавання типових слів українською та англійською мовами. Кожен файл має бути записаний із частотою дискретизації 44.1 кГц і тривалістю не більше 2–3 секунд для оптимальної обробки. Назви файлів мають відповідати текстовому еквіваленту слова чи фрази, наприклад, дякую.wav для слова "дякую". Для ефективного розпізнавання рекомендується додати щонайменше 10–20 зразків для кожної мови, враховуючи різні інтонації та акценти. Загальний розмір папки залежить від кількості зразків, але для базового набору (20 файлів по 500 КБ) становить приблизно 10 МБ.

# ВИСНОВКИ

У рамках виконання бакалаврської кваліфікаційної роботи було розроблено десктопний застосунок для транскрибування аудіо- та відеофайлів у текстовий формат із підтримкою української та англійської мов. Програма працює локально, забезпечуючи автономність і безпеку даних, і пропонує користувачу два методи транскрибування: сучасний підхід на основі глибокого навчання з використанням бібліотеки openai-whisper і шаблонне розпізнавання через алгоритм Dynamic Time Warping (DTW). Розробка проводилася з урахуванням потреб цільової аудиторії — клієнтів і стенографістів, які потребують зручного інструменту для обробки мовних матеріалів.

Основна мета роботи досягнута: створено функціональний прототип, який дозволяє завантажувати аудіо- та відеофайли у форматах MP3, WAV і MP4, обирати мову транскрибування, метод обробки та пристрій (CPU або GPU), переглядати результати в графічному інтерфейсі та експортувати їх у формати TXT і SRT. Для реалізації поставленої мети виконано всі завдання, визначені на початку дослідження. Проведено аналіз предметної області, що дозволив визначити ключові вимоги до системи, включаючи локальну роботу, підтримку української мови та гнучкість у виборі методів транскрибування. Розроблено архітектуру програмного забезпечення з чітким розподілом на модулі: графічний інтерфейс, обробка вхідних даних, транскрибування та збереження результатів. Реалізовано графічний інтерфейс на PyQt6, який забезпечує інтуїтивне управління процесом транскрибування, включно з налаштуванням параметрів сегментації для DTW. Виконано тестування системи на двох пристроях (Intel Core i5-12450H із RTX 2050 і AMD Ryzen 5 5600H із RTX 3050), що підтвердило її працездатність, відповідність вимогам і зручність використання.

Практична цінність розробленого програмного продукту полягає в його універсальності та доступності. Застосунок може бути використаний у різних сферах: в освіті для транскрибування лекцій і семінарів, у медіа для створення субтитрів, у юридичній сфері для документування усних матеріалів. Локальна

робота забезпечує незалежність від мережевих сервісів і захист даних, що є важливим для користувачів, які працюють із конфіденційною інформацією. Скомпільована версія програми, представлена у вигляді інсталяційного пакету, спрощує її встановлення та використання, а гнучкість налаштувань дозволяє адаптувати систему до різних сценаріїв.

Результати тестування показали, що система відповідає функціональним і нефункціональним вимогам. Точність openai-whisper досягає високих показників, особливо з більшими моделями, тоді як DTW демонструє прийнятну якість за наявності достатньої кількості еталонних зразків. Проте виявлено обмеження, зокрема зниження точності при обробці шумного аудіо та залежність DTW від якості еталонних даних. Ці недоліки можуть бути усунуті в майбутніх дослідженнях шляхом інтеграції алгоритмів шумозаглушення та розширення бази еталонних зразків.

Перспективи подальшого розвитку включають кілька напрямків. По-перше, варто вдосконалити алгоритми обробки шумного аудіо, додавши методи шумозаглушення для підвищення точності транскрибування в складних умовах. По-друге, можна розширити підтримку мов, додавши інші мови, наприклад, німецьку чи французьку, за допомогою нових моделей openai-whisper. По-третє, доцільно розробити функцію автоматичного створення еталонних зразків для DTW, що спростить підготовку даних для шаблонного розпізнавання. Нарешті, створення портативної версії програми для роботи з USB-накопичувачів може ще більше підвищити її доступність.

Розроблений застосунок є ефективним рішенням для локального транскрибування аудіо- та відеофайлів, яке відповідає сучасним потребам користувачів. Отримані результати створюють основу для подальшого вдосконалення системи, що може призвести до створення повноцінного комерційного продукту для автоматизації транскрибування в різних сферах.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. VERBIT [Електронний ресурс] - Режим доступу до ресурсу: https://verbit.ai/ai-technology/from-audrey-to-siri-the-evolution-of-speech-recognition-technologies/
2. VERBIT [Електронний ресурс] - Режим доступу до ресурсу: https://evergreens.com.ua/ua/articles/uml-diagrams.html
3. From Audrey to Siri: The Evolution of Speech Recognition Technology [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/products/speech-to-text
4. AMAZON Transcribe [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/transcribe/medical/
5. Google Speech to text [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/speech-to-text
6. python-soundfile [Електронний ресурс] – Режим доступу до ресурсу: https://python-soundfile.readthedocs.io/en/0.13.1/
7. How DTW (Dynamic Time Warping) algorithm works [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=_K1OsqCicBY&ab_channel=ThalesSehnK%C3%B6rting
8. MFCC and DTW [Електронний ресурс] – Режим доступу до ресурсу: https://www.testdevlab.com/blog/audio-comparison-using-mfcc-and-dtw
9. Whisper [Електронний ресурс] – Режим доступу до ресурсу: https://openai.com/index/whisper/
10. An introduction to Dynamic Time Warping [Електронний ресурс] – Режим доступу до ресурсу: https://rtavenar.github.io/blog/dtw.html
11. Hidden Markov Models [Електронний ресурс] – Режим доступу до ресурсу: https://web.stanford.edu/~jurafsky/slp3/A.pdf
12. KNN [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/think/topics/knn

# Додаток А. Код обробки аудіо за допомогою AI Whisper

```
#Використання AI WHISPER
import sys
import whisper
import os
import logging

logging.basicConfig(filename="transcription.log", level=logging.INFO,
encoding="utf-8")

def format_time(seconds):
    hrs, rem = divmod(seconds, 3600)
    mins, secs = divmod(rem, 60)
    return f"{int(hrs):02}:{int(mins):02}:{int(secs):02}"

def transcribe_audio(
    file_path, model_name, language="uk", device="cpu", progress_callback=None
):
    try:
        os.environ["WHISPER_MODELS_DIR"] = os.path.abspath("models")
        if progress_callback:
            progress_callback("Завантаження моделі розпізнавання аудіо..")
        model = whisper.load_model(
            model_name, device=device,
download_root=os.environ["WHISPER_MODELS_DIR"]
```

```
        )

        if language == "auto":
            if progress_callback:
                progress_callback("Автоматичне розпізнавання мови..")
            result = model.transcribe(file_path, task="detect-language")
            language = result["language"]
            if progress_callback:
                progress_callback(f"Виявлена мова: {language}")

        if progress_callback:
            progress_callback("Транскрибування аудіо..")
        result = model.transcribe(file_path, language=language, fp16=True)

        transcription = []
        for segment in result["segments"]:
            transcription.append(
                {
                    "start": segment["start"],
                    "end": segment["end"],
                    "time": f"{format_time(segment['start'])} -
{format_time(segment['end'])}",
                    "text": segment["text"],
                }
            )
```

```
        if progress_callback:
            progress_callback("Завершено")
        return transcription
    except Exception as e:
        if progress_callback:
            progress_callback(f"Помилка: {str(e)}")
        return {"error": str(e)}


if __name__ == "__main__":
    if len(sys.argv) < 5:
        # print(json.dumps({"error": "Помилка: Не передано всі необхідні параметри."}), flush=True)
        sys.exit(1)

    audio_file_path = sys.argv[1]
    model_name = sys.argv[2]
    language = sys.argv[3]
    device = sys.argv[4]

    transcription = transcribe_audio(audio_file_path, model_name, language, device)
    # print(json.dumps(transcription), flush=True)
```

# Додаток Б. Код обробки аудіо за допомогоб алгоритму DTW

```
## Використання алгоритму DTW
import librosa
import numpy as np
import os
import soundfile as sf
from PyQt6.QtCore import QObject, pyqtSignal


class DTWTranscriptionWorker(QObject):
    progress = pyqtSignal(str)
    finished = pyqtSignal(list)
    error = pyqtSignal(str)


    def __init__(
        self,
        file_path,
        reference_folder="reference_samples",
        top_db=20,
        n_mfcc=13,
        min_segment_length=0.05,
        min_pause_length=0.2,
    ):
        super().__init__()
        self.file_path = file_path
```

```
    self.reference_folder = reference_folder
    self.top_db = top_db
    self.n_mfcc = n_mfcc
    self.min_segment_length = min_segment_length
    self.min_pause_length = min_pause_length
    self.sr = None

def pre_emphasis(self, signal, pre_emph=0.97):
    """Попереднє підсилення сигналу."""
    return np.append(signal[0], signal[1:] - pre_emph * signal[:-1])

def get_mfcc_sequence(self, file_path, n_mfcc):
    """Отримання послідовності MFCC з додатковими ознаками."""
    try:
        y, sr = librosa.load(file_path, sr=None)
        if self.sr is None:
            self.sr = sr
        y = self.pre_emphasis(y)
        mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc)
        delta_mfcc = librosa.feature.delta(mfcc)
        delta2_mfcc = librosa.feature.delta(mfcc, order=2)
        combined = np.vstack([mfcc, delta_mfcc, delta2_mfcc]).T
        combined = (combined - np.mean(combined, axis=0)) / (
            np.std(combined, axis=0) + 1e-8
        )
```

```
            return combined
        except Exception as e:
            self.error.emit(f"Помилка вилучення MFCC: {str(e)}")
            return None

    def split_audio(self, file_path, top_db, min_duration, merge_threshold):
        """Сегментація аудіо з можливістю злиття коротких сегментів."""
        try:
            y, sr = librosa.load(file_path, sr=None)
            if self.sr is None:
                self.sr = sr
            intervals = librosa.effects.split(y, top_db=top_db)
            merged_intervals = []

            prev_start, prev_end = intervals[0]
            for start, end in intervals[1:]:
                pause = (start - prev_end) / sr
                if pause < merge_threshold:
                    prev_end = end
                else:
                    merged_intervals.append((prev_start, prev_end))
                    prev_start, prev_end = start, end
            merged_intervals.append((prev_start, prev_end))

            segments = []
```

```
        for start, end in merged_intervals:
            duration = (end - start) / sr
            if duration >= min_duration:
                segments.append((y[start:end], start / sr, end / sr))
        return segments, sr
    except Exception as e:
        self.error.emit(f"Помилка сегментації аудіо: {str(e)}")
        return [], None

def custom_dtw(self, seq1, seq2, window=None):
    try:
        n, m = len(seq1), len(seq2)
        if window is None:
            window = (
                max(n, m) // 10
            )  # Динамічне вікно (10% від максимальної довжини)

        # Ініціалізація матриці вартостей
        cost_matrix = np.full((n + 1, m + 1), np.inf)
        cost_matrix[0, 0] = 0

        # Заповнення матриці з обмеженням вікна
        for i in range(1, n + 1):
            j_min = max(1, i - window)
            j_max = min(m + 1, i + window + 1)
```

```
            for j in range(j_min, j_max):
                # Евклідова відстань між MFCC векторами
                dist = np.sqrt(np.sum((seq1[i - 1] - seq2[j - 1]) ** 2))
                # Мінімум з сусідніх елементів
                cost_matrix[i, j] = dist + min(
                    cost_matrix[i - 1, j],  # вгору
                    cost_matrix[i, j - 1],  # вправо
                    cost_matrix[i - 1, j - 1],  # по діагоналі
                )

        return cost_matrix[n, m]
    except Exception as e:
        self.error.emit(f"Помилка в custom_dtw: {str(e)}")
        return np.inf

def compare_mfcc(self, seq1, seq2):
    """Порівняння двох MFCC-послідовностей за допомогою DTW."""
    try:
        distance = self.custom_dtw(seq1, seq2)
        return distance
    except Exception as e:
        self.error.emit(f"Помилка порівняння MFCC: {str(e)}")
        return float("inf")

def run(self):
```

```
try:
    # Сегментація аудіо
    self.progress.emit("Сегментуємо аудіо...")
    segments, sr = self.split_audio(
        self.file_path,
        top_db=self.top_db,
        min_duration=self.min_segment_length,
        merge_threshold=self.min_pause_length,
    )
    if not segments:
        self.error.emit("Не вдалося сегментувати аудіо")
        return

    self.progress.emit(f"Знайдено {len(segments)} сегментів")

    results = []
    temp_files = []

    # Аналіз кожного сегмента
    for i, (segment, start_time, end_time) in enumerate(segments):
        self.progress.emit(f"Аналізуємо сегмент {i+1}/{len(segments)}")
        segment_path = f"temp_segment_{i}.wav"
        sf.write(segment_path, segment, sr)
        temp_files.append(segment_path)
```

```
input_mfcc_seq = self.get_mfcc_sequence(
    segment_path, n_mfcc=self.n_mfcc
)
if input_mfcc_seq is None:
    results.append(
        {"start": start_time, "end": end_time, "text": "[unknown]"}
    )
    continue

min_distance = float("inf")
best_match = None

# Порівняння з референсами
for ref_file in os.listdir(self.reference_folder):
    if ref_file.endswith((".wav", ".mp3")):
        ref_path = os.path.join(self.reference_folder, ref_file)
        ref_mfcc_seq = self.get_mfcc_sequence(
            ref_path, n_mfcc=self.n_mfcc
        )
        if ref_mfcc_seq is None:
            continue
        distance = self.compare_mfcc(input_mfcc_seq, ref_mfcc_seq)
        self.progress.emit(f"{ref_file}: DTW Distance = {distance:.2f}")
        if distance < min_distance:
            min_distance = distance
```

```
                best_match = ref_file

        # Форматування результату
        if best_match:
            word = best_match.split(".")[0].split("_")[0]
            results.append({"start": start_time, "end": end_time, "text": word})
            self.progress.emit(
                f"Найкращий збіг: {word} (DTW = {min_distance:.2f})"
            )
        else:
            results.append(
                {"start": start_time, "end": end_time, "text": "[unknown]"}
            )
            self.progress.emit("Не вдалося знайти збіг")

    # Видалення тимчасових файлів
    for temp_file in temp_files:
        if os.path.exists(temp_file):
            os.remove(temp_file)

    self.finished.emit(results)
    self.progress.emit("Завершено")
except Exception as e:
    self.error.emit(f"Помилка транскрибування: {str(e)}")
```

# Додаток В. Код який об’єднує всі вікна інтерфейсу

```
# interface.py
import sys
from PyQt6.QtWidgets import QApplication, QMainWindow, QStackedWidget
from PyQt6.QtGui import QIcon
from main_window import MainWindow
from config_window import ConfigWindow
from result_window import ResultWindow
from dtw_result import DTWResultWindow

class Interface(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Транскрибування аудіо")
        self.setWindowIcon(QIcon("icon.ico"))
        self.setStyleSheet(
            "background-color: #121212; color: white; font-family: Arial, sans-serif;"
        )

        self.stack = QStackedWidget()
        self.setCentralWidget(self.stack)

        self.main_window = MainWindow(self)
        self.config_window = None
        self.result_window = None
```

```
        self.hmm_result_window = None  # Додаємо нове вікно

        self.stack.addWidget(self.main_window)
        self.showMaximized()

    def switch_to_config(self, file_path=None):
        if not self.config_window:
            self.config_window = ConfigWindow(self, file_path)
            self.stack.addWidget(self.config_window)
        else:
            self.config_window.set_file_path(file_path)
        self.stack.setCurrentWidget(self.config_window)

    def switch_to_result(self, file_path, model_name, language, device):
        if not self.result_window:
            self.result_window = ResultWindow(
                self, file_path, model_name, language, device
            )
            self.stack.addWidget(self.result_window)
        else:
            self.result_window.update_content(file_path, model_name, language, device)
        self.stack.setCurrentWidget(self.result_window)

    def switch_to_hmm_result(self):
        if not self.hmm_result_window:
```

```
            self.hmm_result_window = DTWResultWindow(self)
            self.stack.addWidget(self.hmm_result_window)
        self.stack.setCurrentWidget(self.hmm_result_window)

    def switch_to_main(self):
        self.stack.setCurrentWidget(self.main_window)

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