ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмної платформи для розпізнавання тексту на базі моделей машинного навчання»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми 126 Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

	Богдан ВОВКОТРУБ
(підпис)	*Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. ІСДМ-63
Богдан ВОВКОТРУБ
Ім'я, ПРІЗВИЩЕ

Керівник: Андрій БОНДАРЧУК
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Рецензент:
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інформаційних систем та технологій

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ

_____ Каміла СТОРЧАК
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Вовкотруб Богдан Володимирович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Розробка програмної платформи для розпізнавання тексту на базі моделей машинного навчання

керівник кваліфікаційної роботи Андрій БОНДАРЧУК, д.т.н., професор ,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» 10.2024р. №320

2. Строк подання кваліфікаційної роботи «27» грудня 2024р.

3. Вихідні дані до кваліфікаційної роботи:
1. Науково-технічна література про технології розпізнавання мовлення.
2. Технічна документація сучасних систем розпізнавання мовлення.
3. Характеристики програмно-апаратного забезпечення (NVIDIA 3070, Intel i7, 32 ГБ оперативної пам'яті).
4. Вимоги до сучасних моделей розпізнавання мовлення.
5. Тестові набори даних для оцінки точності й швидкості перетворення аудіо в текст.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз принципів розпізнавання природнього мовлення за допомогою машинного навчання у текстовому форматі
2. Дослідження технологій розпізнавання мовлення на основі моделей машинного навчання
3. Розробка програмної платформи для розпізнавання мовлення на основі сучасних технологій, побудованих на основі моделей машинного навчання
4. Дослідження ефективності роботи платформи під час одночасного оброблення декількох аудіофайлів різної тривалості.

5. Перелік графічного матеріалу: *презентація*

6. Дата видачі завдання «15» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури.	15.10-17.10.24	виконав
2	Дослідження предметної області. Огляд основних підходів та технологій розпізнавання мовлення	18.10-24.10.24	виконав
3	Обґрунтування вибору сучасних підходів і технологій розпізнавання мовлення.	25.10-29.10.24	виконав
4	Вибірковий аналіз моделей розпізнавання мовлення, на основі яких буде розроблена програмна платформа	30.10-05.11.24	виконав
5	Обґрунтування архітектурних рішень.	06.11-08.11.24	виконав
6	Розроблення алгоритму роботи програмного продукту. Обґрунтування вибору інструментальних засобів розроблення.	09.11-11.11.24	виконав
7	Розробка програмного продукту, в основі якого є обрана система розпізнавання мовлення. Опис компонентів програмного продукту.	12.11-04.12.24	виконав
8	Аналіз результатів. Опис роботи програмного продукту	06.12-13.12.24	виконав
9	Підготовка доповіді, презентації, роздаткового матеріалу, реферату	20.12-26.12.24	виконав

Здобувач вищої освіти _____ Богдан ВОВКОТРУБ

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник

кваліфікаційної роботи _____ Андрій БОНДАРЧУК

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 94 стор., 15 рис., 4 табл., 27 джерел.

Мета роботи – розробка програмної платформи для автоматичного розпізнавання мовлення на базі моделей машинного навчання, зокрема системи WhisperX, для забезпечення високої точності та швидкості перетворення аудіо в текст.

Об'єкт дослідження – технологія автоматичного розпізнавання мовлення.

Предмет дослідження – методологія та технічні рішення для автоматизації розпізнавання мовлення з аудіо файлів з використанням сучасних архітектур і програмних інтерфейсів.

Короткий зміст роботи. Досліджено розробку програмної платформи для автоматичного розпізнавання мовлення на базі системи WhisperX, модель якої ґрунтується на архітектурі Transformer. Окреслено основні компоненти: API для обміну даними, базу даних MongoDB, Redis для кешування, а також WhisperX для розпізнавання мовлення. API обробляє запити, а WhisperX перетворює аудіофайли у текст.

Тестування показало, що оптимально розпізнавати один або два аудіофайли одночасно, оскільки обробка трьох і більше знижує швидкість. Виявлено, що для підвищення продуктивності важливі потужні графічні карти, достатня оперативна пам'ять та ефективний процесор. Рекомендується перейти від монолітної до мікросервісної архітектури для горизонтальної масштабованості, що підвищить продуктивність та адаптивність платформи.

КЛЮЧОВІ СЛОВА: ASR, АВТОМАТИЧНЕ РОЗПІЗНАВАННЯ МОВЛЕННЯ, TRANSFORMER, SPEECH-TO-TEXT, ДІАРИЗАЦІЯ, WHISPER, WHISPERX, ПРОГРАМНА ПЛАТФОРМА, API.

ABSTRACT

The textual part of the qualification work for obtaining a master's degree: 94 pages, 15 figures, 4 tables, 27 sources.

The aim of the work – the development of a software platform for automatic speech recognition based on machine learning models, specifically the WhisperX system, to ensure high accuracy and speed of audio-to-text conversion.

The object of the research – the technology of automatic speech recognition.

The subject of the research – methodology and technical solutions for automating speech recognition from audio files using modern architectures and software interfaces.

Summary of the work: The development of a software platform for automatic speech recognition based on the WhisperX system, which is built on the Transformer architecture, has been investigated. The main components are outlined: an API for data exchange, a MongoDB database, Redis for caching, and WhisperX for speech recognition. The API processes requests, while WhisperX converts audio files into text. Testing has shown that it is optimal to recognize one or two audio files simultaneously, as processing three or more reduces speed. It has been found that powerful graphics cards, sufficient RAM, and an efficient processor are important for improving performance. It is recommended to transition from a monolithic to a microservices architecture for horizontal scalability, which will enhance the performance and adaptability of the platform.

KEYWORDS: ASR, AUTOMATIC SPEECH RECOGNITION, TRANSFORMER, SPEECH-TO-TEXT, DIARIZATION, WHISPER, WHISPERX, SOFTWARE PLATFORM, API.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
NLP	Natural Language Processing
ASR	Automatic Speech Recognition
STT	Speech-to-Text
HMM	Hidden Markov Model
GMM	Gaussian Mixture Model
DNN	Deep Neural Networks
RNN	Recurrent neural network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
VAD	Voice Activity Detection
MSA	Microservices Architecture
ПЗ	Програмне Забезпечення
HTTP	HyperText Transfer Protocol
AMQP	Advanced Message Queuing Protocol
gRPC	Google Remote Procedure Calls
API	Application Programming Interface
ОС	Операційна система
БД	База даних
СУБД	Система управління базою даних

ВСТУП

Актуальність. У сучасному світі автоматичне розпізнавання мовлення стає невід'ємною частиною багатьох технологічних рішень. Попит на ефективні системи, які обробляють мовленнєву інформацію в реальному часі, стрімко зростає в галузях, таких як медицина, юриспруденція, освіта та бізнес. Системи на основі сучасних моделей, які забезпечують високу точність і швидкість обробки, стають актуальними для організацій, що прагнуть оптимізувати процеси.

Об’єктом дослідження є технології автоматичного розпізнавання мовлення, які включають методи для перетворення аудіо сигналів в текстову інформацію та інтеграцію з різними програмними продуктами.

Предметом дослідження є методологія та технічні рішення, що стосуються автоматизації процесів розпізнавання мовлення. Особливу увагу приділено системам, побудованим на архітектурах, таких як Transformer, які здатні забезпечити високу точність і продуктивність.

Мета дослідження: полягає в розробці програмної платформи для автоматичного розпізнавання мовлення на основі системи WhisperX, що забезпечує інтеграцію з іншими програмними продуктами та ефективну обробку аудіофайлів. Це передбачає вивчення існуючих рішень у сфері ASR, вибір оптимальних моделей та розробку архітектури платформи, яка відповідає сучасним вимогам.

Методи дослідження: включають аналіз наукових робіт у сфері ASR, експериментальні дослідження продуктивності різних моделей, а також принципи розробки програмного забезпечення, зокрема, рекомендації щодо переходу на мікросервісну архітектуру.

Структура роботи, Дана робота складається зі вступу, трьох розділів, висновків, додатків та переліку посилань. Перший розділ розглядає актуальність та еволюцію технологій розпізнавання мовлення, другий – вибір оптимальної моделі ASR, а третій – архітектуру розробленої програмної платформи з акцентом на її ключові компоненти та рекомендації щодо її вдосконалення.

1 ТЕОРЕТИЧНІ ОСНОВИ ТА МЕТОДОЛОГІЧНІ ПІДХОДИ ДО РОЗПІЗНАВАННЯ ТЕКСТУ ІЗ ЗВУКУ

1.1 Поняття та особливості технологій розпізнавання мовлення

Великий скачок у розвитку штучного інтелекту (AI), який стрімко прогресує, суттєво вплинув і на розвиток технологій розпізнавання мовлення з аудіофайлів або аудіопотоків у реальному часі у текст. Наприклад, коли людина говорить у мікрофон і відразу її слова перетворюються у текст. Технологією розпізнавання мовлення називається ASR, що з англійської так і розшифровується – автоматичне розпізнавання мовлення (Automatic Speech Recognition). ASR – це технологія машинного навчання (ML) або AI, яка за допомогою акустичних, мовних моделей, та методів обробки сигналів, витягує слова та граматичні конструкції з аудіо, обробляє їх і забезпечує певний тип реакції системи.

ASR, які перетворюють мовлення у текстовий формат, називають загальновживаним терміном Speech-to-Text (STT). Наприклад:

- Google Speech-to-Text API;
- Microsoft Azure Speech-to-Text;
- Whisper Speech-to-Text та інші.

Назва «Speech-to-Text» чітко описує та відображає суть технології: перетворення мовлення *(speech)* в текст *(text)*. Вона є прямим перекладом суті процесу і має загальний описовий характер. Цей термін почав використовуватись ще з ранніх часів розвитку технологій для розпізнавання мовлення, коли основна мета полягала у автоматичному перетворенні аудіо інформації в текстовий формат.

Назви ASR та STT часто використовують як взаємозамінними але, насправді, між назвами ASR та STT є деякі відмінності. STT – це більше про конкретну реалізацю технології ASR, де основна мета полягає саме в перетворенні мовлення в текст. Це вузько направлена задача в межах загального процесу ASR. Отже, STT є частиною ASR, або, скажімо так, STT – це ASR, результатом якого є

розшифрований текст із звукового сигналу. Якщо говорити про перетворення мовлення в текст, STT можна розглядати як окремий підрозділ ASR, тоді як ASR може включати більше можливостей, таких як розпізнавання мовця або контекстне розуміння мовлення.

В свою чергу варто додати, що ASR є частиною гілки обробки природної мови (NLP) у AI – більш глобальної технології машинного навчання, яка дає комп'ютерам можливість інтерпретувати, маніпулювати та розуміти людську мову.

1.2 Місце технологій ASR у сучасному світі

Сучасні технології ASR забезпечують високу точність розпізнавання мовлення, підтримку багатьох мов і здатність працювати в реальному часі, що робить їх незамінними для широкого кола додатків. Інтеграція з іншими технологіями AI, адаптивність до різних умов та користувачів, а також можливість роботи з різними джерелами аудіоданих роблять ASR потужним інструментом для автоматизації обробки мовлення. Технології ASR за останні роки стали одним із ключових елементів у розвитку інноваційних рішень, які змінюють підхід до автоматизації, взаємодії з цифровими пристроями, медіа, освіти та багатьох інших галузей. Їх широке застосування допомагає значно підвищити продуктивність, забезпечити інклюзивність та спростити доступ до інформації. У сучасному світі ASR відіграє роль не тільки як технологія для автоматичного перетворення мовлення на текст, але й як фундамент для більш глибоких змін у тому, як люди взаємодіють із технологіями, обробляють дані та використовують інформацію.

1.2.1 Автоматизація бізнес-процесів та оптимізація робочих операцій за допомогою ASR

ASR стають незамінним інструментом для оптимізації бізнес-процесів у багатьох сферах. У корпоративному середовищі вони використовуються для

автоматичного транскрибування нарад, інтерв'ю та робочих дзвінків, що дозволяє компаніям швидше обробляти і аналізувати інформацію. Це також сприяє документуванню, полегшуючи створення звітів та зберігання важливих даних. Наприклад, у фінансовій сфері, розпізнавання голосових записів може автоматизувати процеси аудиту, де кожна розмова повинна бути зафіксована й оброблена для відповідності регуляційним нормам.

1.2.2 Телекомунікації та обслуговування клієнтів

ASR революціонізують сферу телекомунікацій та обслуговування клієнтів. Голосові боти на базі ASR можуть автоматично розпізнавати мовлення клієнтів та відповідати на їхні запити, що підвищує ефективність роботи колцентрів і зменшує потребу у втручанні операторів. Це забезпечує більш швидке та точне обслуговування клієнтів, покращуючи їхній досвід взаємодії з компаніями.

Автоматизація взаємодії з клієнтами через голосові команди дозволяє компаніям обробляти більшу кількість запитів одночасно, знижуючи навантаження на персонал та підвищуючи рівень задоволення клієнтів. Це також допомагає бізнесу оперативно реагувати на питання та вирішувати проблеми клієнтів у реальному часі, що є важливим аспектом сучасної бізнес-стратегії.

1.2.3 Освітні технології та доступ до знань

ASR мають важливе місце в освітній сфері, де вони використовуються для автоматичного створення субтитрів для онлайн-лекцій, семінарів або вебінарів. Це значно спрощує процес навчання, особливо для людей, які мають обмежений доступ до традиційних освітніх матеріалів або мають особливі освітні потреби. ASR дозволяє студентам отримувати доступ до лекцій у текстовому форматі, що спрощує вивчення матеріалів та створює більш гнучкі можливості для навчання.

Окрім того, ця технологія використовується для створення доступних навчальних матеріалів для людей із порушеннями слуху, що робить освітній процес

більш інклюзивним. Автоматичні транскрипції також використовуються для записів наукових конференцій і дискусій, забезпечуючи зручний доступ до знань у письмовому вигляді.

1.2.4 Інтеграція технологій ASR з голосовими помічниками

ASR стали основою для голосових помічників, таких як Apple Siri, Amazon Alexa, Google Assistant та Microsoft Cortana, які значно спрощують взаємодію користувачів з пристроями. Вони дозволяють виконувати низку завдань, просто промовляючи команди, що підвищує ефективність і зручність використання технологій. За допомогою таких помічників користувачі можуть створювати нагадування, відправляти повідомлення, здійснювати пошук в інтернеті, керувати домашніми пристроями та багато іншого без використання клавіатури або дисплею.

Інтеграція ASR з іншими технологіями відкриває нові можливості для автоматизації повсякденних задач і роботи з розумними пристроями, що робить технології голосового управління більш доступними та ефективними. Голосові помічники поступово стають важливою частиною розумних будинків та офісів, дозволяючи людям легше управляти своїм середовищем за допомогою голосових команд.

1.2.5 Вплив ASR для людей з обмеженими можливостями

ASR має величезний вплив на інклюзивність, особливо для людей з обмеженими можливостями, таких як люди з порушеннями слуху або мовлення. Наприклад, автоматичне створення субтитрів у реальному часі під час лекцій, відеоконференцій або телевізійних передач дозволяє таким людям бути повноцінними учасниками інформаційного процесу. ASR також можуть використовуватися у спеціалізованих додатках для створення доступного контенту, таких як інтерфейси, що підтримують голосові команди для людей, які не можуть користуватися традиційними засобами введення, наприклад, клавіатурою. У

поєднанні з системами NLP, такі рішення забезпечують глибоку інтеграцію з іншими технологіями та допомагають цей світ більш цифровізованим.

1.2.6 Роль STT під час кризових ситуацій у світі

Такі кризові ситуації, як пандемія у 2020 році та війни, особливо інформаційні, у сучасному світі довели на скільки важливими є швидка обробка великого обсягу інформації, збір даних у реальному часі, швидка документація, інформаційна підтримка людей та висвітлення правди, адаптація до різних мов тощо. Адже під час кризових ситуацій кількість запитів від людей дуже значною мірою зростає, і все це потрібно оброблювати. Запити, які надходять від людей, можна обробити за допомогою таких сервісів як генеративний штучний інтелект, що здатний підтримувати запити у текстовому вигляді природними людськими мовами. А для того, щоб такі запити обробляти відразу з голосу людини, на допомогу приходять ASR технології. Так як медична та медіа сфери є критично важливими під час кризи, то розглянемо як застосовується ASR у цих сферах:

- *Застосування ASR у медичній галузі.* У медичній галузі ASR застосовується для автоматичної транскрипції консультацій, медичних обстежень або диктувань лікарів. Це допомагає значно скоротити час на обробку медичних записів і покращує точність документування. Робота [1], в якій досліджувалися методи та способи виявлення осіб, які можуть носити вірус COVID-19, на основі списку симптомів, які вони надають, підкреслює на скільки важливою і актуальною є технологія розпізнавання мовлення разом із машинним навчанням у контексті медицини під час пандемій або кризових ситуацій, коли велика кількість медичних даних має бути оперативно обробленою, демонструючи потенціал для підвищення оперативності у сферах охорони здоров'я та поширення інформації.

- *Застосування ASR у медіа та журналістиці.* У медіа індустрії ASR є потужним інструментом для автоматизації процесів, пов'язаних з обробкою великої кількості аудіо- та відеоконтенту. Це особливо актуально для журналістів, які працюють з інтерв'ю, прес-конференціями або відеозаписами. Технології

розпізнавання мовлення дозволяють швидко транскрибувати інтерв'ю або події, що дає можливість оперативно публікувати інформацію та забезпечувати своєчасний доступ до новин. Завдяки автоматизації процесу розпізнавання мовлення, журналісти можуть сконцентруватися на аналізі даних та створенні контенту, тоді як ASR забезпечує швидке і точне перетворення аудіо в текстовий формат.

ASR суттєво покращують доступність інформації для людей, забезпечуючи їм можливість отримувати новини та медичні рекомендації в зручному для них форматі. У часи кризи, коли важливо швидко реагувати на запити, ASR також може бути використано для аналізу соціальних мереж, виявляючи ключові тренди та емоції, що дозволяє організаціям краще розуміти потреби суспільства. Це не лише підвищує ефективність роботи медіа та медичних установ, але й сприяє формуванню більш інформованого та відповідального суспільства.

1.3 Розбір технології ASR

Будь-яке рішення для розпізнавання мовлення базується на технології ASR. Розпізнавання мовлення – це дуже складний процес, тому що аудіодані залежать від фонового шуму, якості аудіо, акцентів і промислового жаргону, що, як відомо, ускладнює сприйняття машинами. Тому десятиліттями ASR не такими точними як сьогодні.

1.3.1 Статистичні моделі ASR

Раніше ASR розпізнавали мовлення за допомогою статистичних моделей, які були обмежені в своїй здатності точно розпізнавати різні акценти, діалекти та стилі мовлення. Існують такі статистичні моделі автоматичного розпізнавання мовлення: Приховані Марківські моделі (HMM) та модель Гаусової суміші (GMM), яка доповнює першу.

1.3.1.1 Приховані Марківські моделі (HMM)

HMM були стандартом в ASR протягом багатьох років. Вони використовують статистичні методи для моделювання послідовностей звуків. HMM відмінно працюють при наявності обмеженої кількості даних, але можуть мати проблеми з масштабуванням. В ASR HMM використовується для моделювання звукових сигналів у часі, де кожен стан представляє певний звук або фонему.

Основна ідея HMM полягає в тому, що мовлення можна розглядати як послідовність спостережуваних акустичних сигналів, які генеруються з прихованої послідовності фонем або станів мовного процесу. Математично це можна описати за допомогою таких компонентів:

- множина станів:

$$S = \{S_1, S_2, \dots, S_N\} \tag{1.1}$$

де N – кількість можливих станів. Кожен стан відповідає певному фізичному чи акустичному стану мови (наприклад, фонеми). Стани є прихованими, тобто ми не знаємо точно, у якому стані система перебуває в певний момент часу;

- множина спостережень:

$$O = \{O_1, O_2, \dots, O_T\} \tag{1.2}$$

де T – кількість акустичних векторів (фреймів) у мовному сигналі. Спостереження представляють собою вимірювані характеристики сигналу (частоти, енергії тощо) на кожному часовому кроці.

- ймовірність переходів між станами:

$$A = \{a_{ij}\}, a_{ij} = P(S_j|S_i) \tag{1.3}$$

де A – матриця розміром $M \times N$, яка описує ймовірність переходів між прихованими станами;

a_{ij} – ймовірність переходу із стану S_i у стан S_j ;

- ймовірність спостережень для кожного стану:

$$B = \{b_j(O_T)\}, b_j(O_T) = P(O_T|S_j) \quad (1.4)$$

де $b_j(O_T)$ – ймовірність спостереження O_T за умови, що система перебуває у стані S_j;

- ймовірність початкових станів:

$$\pi = \{\pi_i\},\ \pi_i = P(S_1 = S_T) \quad (1.5)$$

де π_i – ймовірність того, що перший стан моделі S_1 дорівнює S_T

У задачі розпізнавання мовлення нам потрібно знайти найімовірнішу послідовність прихованих станів (послідовність фонем або слів), які спричинили спостережувану послідовність акустичних ознак описаних формулою 1.1.

Основними задачами HMM у розпізнаванні мовлення є:

- оцінка ймовірності послідовності спостережень: обчислюється ймовірність того, що модель згенерувала спостережувану послідовність O, використовуючи алгоритм прямого проходу:

$$P(O|\lambda) = \sum_S P(O|S,\lambda)P(S,\lambda),\ \lambda = (A,B,\pi) \quad (1.6)$$

де λ – параметри моделі;

S – приховані стани.

- пошук найбільш ймовірної послідовності станів: для знаходження послідовності станів, яка з найбільшою ймовірністю відповідає спостереженням, використовують алгоритм Вітербі. Його суть полягає в тому, щоб максимізувати ймовірність прихованої послідовності станів для заданих спостережень:

$$Q^* = arg\ maxP(Q|O,|\lambda) \quad (1.7)$$

де Q – послідовність станів S.

- оцінка параметрів моделі: Використовуючи алгоритм Баум-Велша (очікування-максимізація), можна оцінити параметри моделі (A, B, π) на основі навчальних даних, що дозволяє покращити точність розпізнавання мовлення:

$$P(\lambda|O) = arg \max_{\lambda} P(O|\lambda) \quad (1.8)$$

Розглянемо приклад застосування. Нехай ми маємо набір звукових сигналів, розділених на короткі фрагменти (вектори ознак). Ці фрагменти відповідають акустичним спостереженням O, і ми використовуємо HMM для того, щоб знайти найімовірнішу послідовність фонем (прихованих станів) S, яка відповідає цьому сигналу. Кроки такі:

1) Моделювання акустичних векторів (спостереження) для кожної фонеми за допомогою функцій ймовірностей за формулою 1.4;

2) Використання ймовірностей переходів за формулою 1.3, щоб відобразити ймовірність переходу між фонемами (станами)

3) За допомогою алгоритму Вітербі знаходимо найімовірнішу послідовність фонем на основі послідовності спостережень.

Приховані Марківські моделі дозволяють будувати математично обґрунтовані рішення для розпізнавання мовлення, що дає можливість відтворювати послідовність фонем або слів, яка відповідає записаному мовленню.

1.3.1.2 Модель Гаусової суміші (GMM)

Для кращого моделювання акустичних ознак у межах кожного стану використовується модель Гаусової суміші (GMM*),* яка використовуються разом з HMM для моделювання акустичних ознак. Гаусові суміші дозволяють враховувати

складніші розподіли даних, представляючи кожен стан як суміш декількох нормальних розподілів. Ймовірність спостережуваних акустичних ознак O_T у стані S_i описується формулою:

$$P(O_T|S_i) = \sum_{m=1}^{M} c_m \mathcal{N}(O_T|\mu_m, \Sigma_m) \quad (1.9)$$

де $\mathcal{N}(O_T|\mu_m, \Sigma_m)$ – це нормальний розподіл з середнім значенням μ_m та коваріаційною матрицею Σ_m;

c_m – ваговий коефіцієнт для кожної компоненти суміші.

Використання GMM дозволяє більш точно моделювати розподіл акустичних ознак у кожному стані HMM, підвищуючи точність розпізнавання мовлення. Але їх ефективність використання прихованих Марківських моделей у поєднанні з Гаусовою сумішшю обмежена в порівнянні з новішими методами.

1.3.2 Моделі ASR побудовані на алгоритмах машинного навчання

З появою ML і глибокого навчання (DL) у 2010-х роках ASR дуже змінилися. Статистичні моделі поступово були замінені алгоритмами ML, які здатні фіксувати ідіоматичні вирази та інші нюанси, які раніше було важко виявити. Існують такі ASR ML: глибокі нейронні мережі (DNN) та рекурентні нейронні мережі (RNN) [3].

1.3.2.1 Глибокі нейронні мережі (DNN)

DNN використовуються для покращення акустичного моделювання. Вони значно перевершують традиційні статистичні моделі у точності, однак можуть бути складними для тренування. Для математичного опису використання глибоких нейронних мереж у задачі розпізнавання мовлення, можна виділити кілька

ключових аспектів, які охоплюють структуру моделі, процес навчання та функцію втрат.

Розглянемо структуру моделі DNN, що складається з множини шарів:

- *вхідний шар:* представлення вхідних даних, яке може бути у вигляді векторів ознак, що характеризують аудіосигнал. Наприклад, це можуть бути вектори спектрограм, що отримані з аудіосигналу за допомогою перетворення Фур'є.

$$x = [x_1, x_2, \dots, x_n] \tag{1.10}$$

- *приховані шари:* кожен прихований шар l має n_l нейронів. Вихід нейронів j на l-му шарі обчислюється за формулою:

$$h_j^{(l)} = \sigma\left(\sum_{i=1}^{n_{l-1}} w_{ij}^{(l)}\, h_i^{(l-1)} + b_j^{(l)}\right) \tag{1.11}$$

де $h_j^{(l)}$ – вихід нейрона j на l-му шарі;

$w_{ij}^{(l)}$ – вага між нейроном i в $(l-1)$-му шарі та нейроном j в l-му шарі;

$b_j^{(l)}$ – зсув для нейрону j в l-му шарі;

σ – активізаційна функція (наприклад, ReLU, Sigmoid [2]).

- *вихідний шар:* формує ймовірності для кожного класу (наприклад, для розпізнавання слів). Якщо маємо C класів, то:

$$y_j = softmax\left(h_j^{(L)}\right) = \frac{e^{h_j^{(L)}}}{\sum_{k=1}^{C} e^{h_k^{(L)}}} \tag{1.12}$$

де $h_j^{(L)}$ – вихід нейрона j на останньому шарі;

y_j – ймовірність, що вхідний сигнал належить до класу j.

Щодо процесу навчання, то навчання DNN виконується за допомогою алгоритму зворотного розповсюдження помилки (back propagation), який мінімізує функцію втрат, що вимірює відхилення передбачених значень від реальних. Функція втрат для задачі класифікації часто обирається як крос-ентропія:

$$L(y, \hat{y}) = -\sum_{j=1}^{C} y_j \log(\hat{y}_j) \tag{1.13}$$

де y – реальний вектор міток (один гарячий вектор);

$\hat{y}$ – передбачений вектор ймовірностей.

Після процесу навчання наступним кроком йде оптимізація ваг та зсувів, яка виконується за допомогою алгоритму градієнтного спуску (Gradient Descent):

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta \frac{\partial L}{\partial w_{ij}^{(l)}} \tag{1.14}$$

де η – швидкість навчання.

Об'єднуючи все разом, можна описати процес розпізнавання мовлення за допомогою DNN так:

$$\hat{y} = f(x; W, b) \tag{1.15}$$

де f – функція, яка описує структуру нейронної мережі;

W – матриця ваг;

b – вектор зсувів;

$\hat{y}$ – передбачення ймовірностей класів для вхідного сигналу x.

Глибокі нейронні мережі забезпечують потужний інструмент для автоматичного розпізнавання мовлення, оскільки можуть навчатися на великих

обсягах даних та адаптуватися до різних акцентів і умов. Формалізація цих процесів за допомогою математичних моделей дозволяє краще розуміти механізми, які лежать в основі DNN, і сприяє їх подальшому розвитку та оптимізації.

1.3.2.2 Рекурентні нейронні мережі (RNN)

Оскільки мовлення є часовою послідовністю, важливо враховувати залежності між різними моментами часу. RNN забезпечують можливість моделювати такі залежності, оскільки кожен стан мережі залежить від попередніх вхідних даних.

Основна відмінність RNN від глибоких нейронних мереж полягає у наявності рекурентних зв'язків, що дозволяють мережі зберігати стан на кожному часовому кроці. Для вхідного сигналу x_t на часі t вихідний стан h_t обчислюється як:

$$h_t = \sigma(W_x x_t + W_h h_{t-1} + b) \tag{1.16}$$

де x_t – вектор вхідних ознак на кроці t;

h_t – стан мережі на кроці t,; який залежить як від поточного вхідного вектора x_t, так і від попереднього стану h_{t-1};

W_x – матриця ваг для входу;

W_h – матриця ваг для рекурентного зв'язку (попереднього стану);

b – вектор зсувів;

σ – нелінійна активаційна функція (наприклад, $tanh$ або ReLU).

Цей підхід дозволяє RNN зберігати інформацію про попередні стани, що робить їх придатними для обробки часових даних, таких як мовлення.

В задачах розпізнавання мовлення RNN передбачають ймовірність вихідної послідовності символів або слів. На кожному часовому кроці передбачена

ймовірність для відповідного класу (фонеми, слова тощо) обчислюється за допомогою softmax-функції:

$$\widehat{y_t} = softmax(W_y h_t + b_y) \quad (1.17)$$

де W_y – матриця ваг для перетворення прихованого стану в ймовірності класів;

b_y – вектор зсувів.

Як і в інших моделях класифікації, для навчання мережі використовується функція втрат крос-ентропії:

$$L = -\sum_{t=1}^{T}\sum_{k=1}^{C} y_t^{(k)} log\left(\hat{y}_t^{(k)}\right) \quad (1.18)$$

де T – довжина вхідної послідовності;

C – кількість класів;

$y_t^{(k)}$ – реальне значення на часі t для класу k;

$\hat{y}_t^{(k)}$ – передбачена ймовірність для класу k на часі t.

У задачах розпізнавання мовлення RNN та DNN не є окремими конкуруючими технологіями, а, навпаки, вони часто доповнюють одна одну. У процесі обробки мовлення кожна з цих мереж виконує свою окрему роль, і поєднання цих мереж дає кращий результат.

DNN добре працюють для екстракції високорівневих ознак із фіксованих вхідних даних, таких як аудіо-фрагменти або спектрограми. Вони здатні навчати складні нелінійні відображення між вхідними даними та вивченими ознаками. DNN використовуються для створення представлення мовного сигналу на більш високому рівні абстракції, наприклад, для моделювання фонем або інших одиниць мовлення.

1.3.3 Поліпшені версії моделей ASR побудованих на алгоритмах машинного навчання

Однією з проблем RNN є згасання градієнта [4], що обмежує їх здатність зберігати інформацію на довгих часових відрізках. Для подолання цієї проблеми були розроблені модифікації RNN, такі як LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit).

1.3.3.1 Модифікація RNN – LSTM

LSTM додає механізми керування інформацією через комірки пам'яті. У кожного нейрону є три «ворота»:

- забуття (Forget Gate): вирішує, яку частину інформації зберегти або забути;
- вхід (Input Gate): вирішує, яку нову інформацію додати до пам’яті;
- вихід (Output Gate): вирішує, яку інформацію використовувати для передбачення на поточному кроці.

Кожен з цих воріт є окремим нейронним шаром, який приймає вхід, визначає, яку частину інформації слід передати далі або зберегти в комірках пам'яті, і забезпечує баланс між збереженням старої інформації і додаванням нової.

Математично це можна описати так:

1) Визначаємо забуттєві ворота за формулою:

$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big),\ \sigma(x) = \frac{1}{1+e^{-x}} \tag{1.19}$$

де f_t – забуттєві ворота на часовому кроці t. Це вектор значень між 0 і 1, який визначає, яку частину попередньої інформації слід зберегти, а яку "забути". Якщо значення близьке до 1 — інформацію слід зберегти, якщо до 0 — забути;

σ – сигмоїдна функція активації. Це нелінійна функція, яка перетворює вхід в значення в діапазоні від 0 до 1;

W_f – ваги забуттєвих воріт. Це матриця ваг, яка визначає вплив попереднього прихованого стану та поточного входу на значення забуттєвих воріт;

h_{t-1} – прихований стан на попередньому часовому кроці. Це вектор, що зберігає інформацію про минулий контекст. Він є виходом нейронної мережі на попередньому кроці часу t-1;

x_t – поточний вхід на кроці часу t. Це вхідні дані, що поступають на нейронну мережу на поточному кроці (наприклад, частина звукового сигналу);

$[h_{t-1}, x_t]$ – конкатенація попереднього прихованого стану h_{t-1} і поточного входу x_t. Це поєднання векторів в один вектор, який буде використаний як вхід до LSTM;

b_f – зміщення забуттєвих воріт. Це вектор зміщення, що додається до результату матричного множення.

2) Визначаємо вхідні ворота на часовому кроці t за формулою:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{1.20}$$

де i_t – вхідні ворота на часовому кроці t. Це вектор значень між 0 і 1, який визначає, скільки нової інформації слід додати до стану комірки;

σ – сигмоїдна функція активації, яка перетворює вхід в значення в діапазоні від 0 до 1;

W_i – ваги вхідних воріт. Це матриця, яка визначає вплив попереднього прихованого стану і поточного входу на вхідні ворота;

b_i – зміщення вхідних воріт. Це вектор зміщення, що додається до результату матричного множення для вхідних воріт;

3) Підбираємо кандидата на новий стан комірки на часовому кроці t. Це потенційно нова інформація, яку можна додати для вхідних воріт. Обчислюємо це за допомогою формули:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C),\ tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (1.21)$$

де $\tilde{C}_t$ – кандидат на новий стан комірки на часовому кроці t;

$tanh$ – гіперболічний тангенс, функція активації, яка перетворює значення у діапазон від -1 до 1;

W_C – ваги для обчислення нового кандидата стану комірки. Це матриця ваг, яка застосовується до векторів прихованого стану та вхідного сигналу;

b_C – зміщення для обчислення нового кандидата стану комірки. Це вектор зміщення, який додається до результату матричного множення.

4) Оновлюємо стан комірки за формулою:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \widetilde{C}_t \quad (1.22)$$

де C_t – оновлений стан комірки на часовому кроці *t*. Це новий стан пам'яті після оновлення. Він містить важливу інформацію для наступних часових кроків;

f_t – забуттєві ворота, які визначають, скільки старої інформації з C_{t-1} потрібно зберегти;

C_{t-1} – попередній стан комірки. Це інформація, що зберігалася на попередньому кроці часу;

i_t – вхідні ворота, які визначають, скільки нової інформації потрібно додати до стану комірки;

$\widetilde{C}_t$ – новий кандидат на стан комірки, що представляє нову інформацію для додавання.

5) Визначаємо, яка частина інформації з оновленого стану комірки повинна бути передана на вихід. Іншими словами, формуємо «фільтр» (вектор o_t) для інформації, що виходить зі стану комірки за формулою:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{1.23}$$

де o_t – це вектор значень вихідних воріт на часовому кроці t, який визначає, скільки інформації з оновленого стану комірки C_t передається на вихід;

σ – сигмоїдна функція, яка перетворює результат множення та додавання в значення між 0 і 1, що дозволяє «контролювати» рівень передачі інформації.

6) Застосовуємо «фільтр» до нормалізованого значення стану комірки за формулою:

$$h_t = o_t \cdot tanh(C_t) \tag{1.24}$$

де h_t – це прихований стан на кроці часу t. Він є фінальним виходом LSTM на поточному кроці і служить як вхід для наступного кроку;

C_t – це оновлений стан комірки після всіх операцій на кроці часу t;

$tanh(C_t)$ – функція, яка перетворює стан комірки в значення між -1 та 1, тобто нормалізує інформацію для прихованого стану.

Вектор вихідних воріт o_t служить ваговим коефіцієнтом для інформації, що виходить із стану комірки. Якщо o_t близький до 1, значення з комірки передається майже повністю. Якщо o_t близький до 0 – значення майже не передається.

Прихований стан h_t на поточному кроці часу t є кінцевим результатом, який буде використовуватися як:

- вихід моделі на цьому часовому кроці;

- вхід для наступного кроку часу, що дозволяє LSTM зберігати контекст і залежності між тимчасовими кроками.

Прихований стан h_t залежить від:

- стану комірки C_t, який акумулює пам'ять (інформацію) за попередні кроки;

- вихідних воріт o_t, які визначають, яка частина цього акумульованого стану буде передана як вихід.

Отже, на кожному кроці часу ми отримуємо h_t, який містить зважену та нормалізовану інформацію зі стану комірки, що слугує результатом роботи LSTM для поточного моменту часу.

1.3.3.2 Модифікація RNN – GRU

GRU (Gated Recurrent Unit) — це модифікація RNN, яка була розроблена для зменшення проблеми «згасаючого градієнта» і покращення здатності моделі вивчати довготривалі залежності в послідовностях. Вона подібна до LSTM, але менш складна, оскільки має менше параметрів, що робить її швидшою для навчання і менш вимогливою до ресурсів.

GRU використовує два основні типи воріт:

- оновлювальні ворота (update gate) z_t;

- скидувальні ворота (reset gate) r_t.

Ці ворота керують тим, скільки інформації з минулих станів передавати далі і як оновлювати поточний стан.

GRU використовує такі основні рівняння:

1) Оновлювальні ворота:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \tag{1.25}$$

де z_t – оновлювальні ворота, які контролюють, скільки попереднього прихованого стану h_{t-1};

W_z – матриця ваг для оновлювальних воріт;

$[h_{t-1}, x_t]$ – конкатенований вектор прихованого стану з попереднього кроку h_{t-1} і поточного вхідного вектора x_t;

b_z – вектор зміщень для оновлювальних воріт;

σ – сигмоїдна функція, яка обмежує значення воріт в діапазоні від 0 до 1.

Оновлювальні ворота визначають, яка частина минулого прихованого стану буде використана для обчислення нового прихованого стану. Якщо $z_t \approx 1$, прихований стан майже не змінюється. Якщо $z_t \approx 0$, тоді прихований стан значно оновлюється.

2) Скидувальні ворота:

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \tag{1.26}$$

де r_t – скидувальні ворота, які визначають, скільки минулої інформації з попереднього прихованого стану h_{t-1} потрібно «забути»;

W_r – матриця ваг для скидувальних воріт;

b_r – вектор зміщень для скидувальних воріт.

Якщо $r_t \approx 0$, це означає, що модель «забуде» багато інформації з попереднього прихованого стану. Це особливо корисно, коли потрібно оновити прихований стан на основі поточного входу без сильного впливу минулого.

3) Кандидат на новий прихований стан:

$$\tilde{h}_t = tanh(W_h \cdot [r_t \cdot h_{t-1}, x_t] + b_h) \tag{1.27}$$

де $\tilde{h}_t$ – кандидат на новий прихований стан;

$r_t \cdot h_{t-1}$ – результат елементного множення скидувальних воріт r_t та прихованого стану h_{t-1}. Це означає, що скидувальні ворота можуть «вимкнути» частину інформації з попереднього прихованого стану.

W_h – матриця ваг для кандидата на новий прихований стан;

b_h – вектор зміщень для кандидата на новий прихований стан;

$tanh$ – гіперболічний тангенс, який обмежує значення в діапазоні від -1 до 1, нормалізуючи вихід.

Кандидат на новий прихований стан є оновленою версією прихованого стану, яка враховує скидувальні ворота і поточний вхід x_t.

4) Остаточне обчислення нового прихованого стану:

$$h_t = z_t \cdot h_{t-1} + (1 - z_t) \cdot \tilde{h}_t \tag{1.28}$$

де h_t – це новий прихований стан на часовому кроці t;

$z_t \cdot h_{t-1}$ – частина прихованого стану, яка переноситься з попереднього кроку;

$(1 - z_t) \cdot \tilde{h}_t$ – частина оновленого прихованого стану, яка визначається через кандидата $\tilde{h}_t$.

Ця формула показує, як оновлювальні ворота z_t регулюють баланс між збереженням старої інформації h_{t-1} і додаванням нової інформації $\tilde{h}_t$. Коли z_t близьке до 0, модель більше покладається на новий кандидат $\tilde{h}_t$.

Загалом GRU можна описати так:

1) Оновлювальні ворота z_t керують тим, скільки попереднього прихованого стану h_{t-1} ми хочемо зберегти в новому стані h_t.

2) Скидувальні ворота r_t контролюють, скільки інформації з попереднього прихованого стану повинно бути «скинуто» при обчисленні кандидата на новий прихований стан $\tilde{h}_t$.

3) Кандидат на новий прихований стан $\tilde{h}_t$ враховує поточний вхід і оновлює прихований стан на основі цього входу і частини попереднього стану.

4) Фінальне значення прихованого стану h_t поєднує стару інформацію h_{t-1} і оновлену інформацію $\tilde{h}_t$, збалансовану через оновлювальні ворота z_t.

GRU на кожному часовому кроці генерує новий прихований стан $\tilde{h}_t$, який є балансом між старою інформацією h_{t-1} і новою інформацією $\tilde{h}_t$. Оновлювальні та скидувальні ворота дозволяють моделі вирішувати, яку інформацію зберігати, а яку скинути, що допомагає GRU вчитися на довготривалих залежностях у даних.

GRU є більш простою архітектурою порівняно з LSTM, оскільки має лише два типи воріт (замість трьох, як у LSTM), що робить її швидшою та менш ресурсоємною.

1.3.4 Сучасна модель ASR – Transformer

У 2017 році з’явилася нова модель Transformer [5], що безумовно стало одним із найважливіших винаходів десятиліття. Це найсучасніша модель, яка використовує механізми самоуваги (self-attention) або самоконтролю для обробки послідовних даних. Вона працює з усіма елементами входу одночасно, що дозволяє досягти високої точності. На відміну від усіх попередніх моделей, цій моделі вдалося вловити віддалені залежності між різними частинами мови, що дозволяє їм враховувати ширший контекст кожного транскрибованого речення. Завдяки цьому Transformer є потужнішою моделлю в багатьох задачах розпізнавання мовлення, зокрема у відношенні точності та швидкості.

Примітка. Процес перетворення аудіофайлу в письмовий формат за допомогою технологій STT називають транскрибуванням (transcribing), а текст, який отримали в результаті – транскрибований текст. Іншими словами, «транскрибований» можна замінити на «розшифрований».

Архітектура Transformer для транскрибування аудіо в текст базується на механізмі уваги (attention) та принципах енкодера-декодера. Розглянемо математичні основи процесу покроково.

1) Вхідний сигнал:

Нехай маємо $x_1, x_2, \dots, x_T$ – це вхідні ознаки аудіо-сигналу (наприклад, спектральні або мел-кепстральні коефіцієнти), де T – кількість кадрів, де кожен x_i – це матриця, що має розмір d_{model} або вектор ознак або ембедінг для кожного кроку часу i. Це може бути ознака аудіо чи будь-якого іншого сигналу.

Вхідні ознаки послідовності можна записати у такому вигляді:

$$X = [x_1, x_2, \dots, x_T] \tag{1.29}$$

Ознаки подаються на вхід енкодера, який перетворює цю послідовність в згладжене внутрішнє подання для подальшої обробки.

2) Енкодер:

Енкодер складається з N шарів, кожен з яких містить дві основні складові:

- Нормалізація та обробка позиційної інформації (Positional Encoding).
- Мультиголовий механізм уваги (Multi-head Attention);

2.1) Нормалізація та початкова обробка кожної послідовності:

Перед тим як обчислювати Multi-Head Attention для кожної послідовності X, потрібно провести початкову обробку вхідних даних, щоб включити в них інформацію про позицію кожного елемента в послідовності. Це забезпечується через додавання позиційного кодування до вхідних ознак (векторів), а потім вже ці «оброблені» послідовності передаються на обчислення Multi-Head Attention.

Оскільки Transformer не враховує природну послідовність елементів через свою паралельну архітектуру, ми додаємо до кожного вектора ознак позиційне кодування. Це дозволяє моделі «знати», який елемент в якому місці послідовності знаходиться.

Позиційне кодування PE для кожної позиції обчислюється на основі синусоїдальних функцій і додається до вхідних ознак:

$$PE_{(pos,2i)} = sin\left(\frac{pos}{1000^{\frac{2i}{d_{model}}}}\right) \quad (1.30)$$

$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{1000^{\frac{2i}{d_{model}}}}\right) \quad (1.31)$$

$$Z_{input} = X + PE \quad (1.32)$$

де pos – позиція елемента в послідовності;

i – індекс елемента у векторі;

d_{model} – розмірність вхідних ознак кожної послідовності або кількість вимірів векторів ембедінгу;

$PE_{(pos,2i)}$ – значення синусоїдального позиційного кодування для парних індексів вимірів векторів (для компонент 0, 2, 4, ...);

$PE_{(pos,2i+1)}$ – значення косинусоїдального позиційного кодування для непарних індексів вимірів векторів (для компонент 1, 3, 5, ...);

Z_{input} – «оброблена» послідовність за допомогою позиційного кодування.

Це допомагає моделі враховувати позиції кадрів у часі.

2.2) Мультиголовий механізм уваги (Multi-head Attention):

Transformer використовує Механізм уваги, який дозволяє кожному елементу послідовності звертати увагу на інші елементи. Для кожної голови уваги, вхідні ознаки X проектуються у простір запитів Q (query), ключів K (key) та значень V (value):

$$Q = Z_{input}W_Q \quad (1.33)$$

$$K = Z_{input}W_K \quad (1.34)$$

$$V = Z_{input}W_V \quad (1.35)$$

де W_Q, W_K, W_V – це навчальні матриці перетворення, які перетворюють вектори послідовності в запити, ключі та значення.

Далі для кожного елемента вхідної послідовності обчислюється увага (Attention) як:

$$\mathrm{Attention}(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1.36}$$

d_k – розмірність простору ключів (і запитів);

$\frac{QK^T}{\sqrt{d_k}}$ – скалярний добуток запитів і ключів, нормалізований на корінь квадратний з розмірності простору ключів (і запитів), щоб уникнути занадто великих значень при обчисленнях, де K^T – це транспонована матриця ключів, яка має розмірність $d_k \times n$ (це означає, що стовпці стають рядками і навпаки);

$softmax$ – функція активації, яка переводить скалярні значення в імовірності.

Так як у кожній послідовності на кожному часовому кроці є різні характеристики, які описують аудіо-сигнал, то тоді потрібно на кожному часовому кроці одночасно (паралельно) враховувати кожну характеристику. Тобто потрібно паралельно обчислювати увагу для кожної з характеристик. Таке обчислення умовно ділять на «голови» які ніби слідкують за кожною з характеристик у кожній послідовності на кожному часовому кроці. Таке одночасне обчислення уваг називають Multi-Head Attention. Multi-Head Attention дозволяє моделі адаптивно вивчати більше інформації з вхідних даних. Це особливо важливо для обробки складних та різноманітних послідовностей, таких як аудіо-сигнали.

Тепер, коли послідовність поділена на «голови», які отримують окремі запити, ключі та значення, тоді обчислення уваги для кожної з голов виглядає так:

$$\mathrm{Attention}(Q_i, K_i, V_i) = softmax\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i \quad (1.37)$$

де Q_i, K_i, V_i – запити, ключі та значення для i-ої голови

Результат обчислення уваги для кожної «голови» називають «виходом голови». Для обчислення уваги для кожної «голови» вектор запиту Q, ключа K, та значення V проектуються через відповідні матриці ваг W_Q, W_K, W_V, тому вихід голови можна позначити як:

$$head_i = Attention(QW_Q^{(i)}, KW_K^{(i)}, VW_V^{(i)}) \quad (1.38)$$

Усі «виходи голови» потрібно об'єднати до одного спільного результату, тобто потрібно зробити конкатенацію уваг кожної з «голів». Конкатенація (Concat) означає, що ми беремо результати уваги з усіх голів і просто поєднуємо їх у одну велику матрицю розміром $T \times (h \cdot d_v)$, де T – це кількість елементів послідовності, h – кількість голів, а d_v – розмір векторів значень кожної голови.

Конкатенація «голів» виглядає так:

$$Concat(head_1, head_2, \dots, head_h) \quad (1.39)$$

Для того, щоб повернутися до початкового вигляду матриці розміром $T \times d_{model}$, що відповідає розміру початкової послідовності ознак, потрібно помножити результат конкатенації на матрицю ваг W_O, яка є частиною параметрів моделі і навчається під час тренування нейронної мережі. Матриця W_O не є фіксованою або заданою заздалегідь — вона навчається разом з іншими параметрами нейромережі під час тренування моделі. Це є частиною механізму

трансформації інформації, щоб результати з різних голов були узгодженими для подальших шарів мережі.

Результат множення результату конкатенації на матрицю ваг W_O і є результатом обчислення мультиголової уваги. Тоді для Multi-Head Attention отримуємо таку загальну формулу для обчислення мультиголової уваги:

$$MultiHead(Q, K, V) = Concat(head_1, head_2 \dots, head_h)W_O \quad (1.40)$$

Multi-Head Attention в Енкодері дозволяє кожному елементу вхідної послідовності (наприклад, ознакам аудіо) фокусуватися на різних частинах тієї ж послідовності, зважаючи на взаємозв'язки між різними кадрами. Простими словами можна сказати, що мультиголова увага дозволяє моделі одночасно звертати увагу на різні частини вхідної послідовності, забезпечуючи вивчення більш комплексних залежностей між кадрами.

3) Декодер:

Декодер – це механізм, який дозволяє вихід енкодера перетворити у текст. Він також використовує Multi-Head Attention, але з додатковим механізмом уваги для зв'язку з виходом енкодера. Він отримує попередні виходи і намагається передбачити наступний символ (або токен), одночасно враховуючи інформацію від енкодера.

Декодер у Transformer складається з кількох шарів, подібно до енкодера, але з деякими відмінностями. Основні компоненти декодера:

- мультиголова увага з маскуванням (Masked Multi-Head Self-Attention);
- Multi-Head Attention – для взаємодії з виходом енкодера;
- Feed Forward Layer – для нелінійних перетворень;
- Позиційне кодування – для того, щоб декодер «знав», у якому порядку генерується вихід.

Математичний опис роботи декодера виглядає так:

3.1) Вхідна послідовність (частково згенерований текст):

Нехай маємо вже згенеровану частину цільової послідовності на кроці t:

$$Y = [y_1, y_2, \dots, y_{t-1}] \tag{1.41}$$

Спочатку Y може бути порожнім, або починатися з токена початку речення.

До кожного елемента y_i додається позиційне кодування *PE*, як це робилось у енкодері:

$$Z_{input}^{target} = Y + PE \tag{1.42}$$

3.2) Masked Multi-Head Self-Attention:

На цьому етапі обчислюється увага, так само, як і в енкодері, але з додатковим маскуванням. Маскування забороняє моделі звертатись до майбутніх елементів послідовності. Це забезпечує те, що на кроці t модель не має доступу до елементів $y_{t+1}, y_{t+2}, \dots$.

Використовуються ті ж формули для обчислення:

$$Q = Z_{input}^{target} W_Q \tag{1.43}$$

$$K = Z_{input}^{target} W_K \tag{1.44}$$

$$V = Z_{input}^{target} W_V \tag{1.45}$$

Потім обчислюється Multi-Head Attention аналогічно як у Енкодері, тільки з маскуванням. Для цього додається маска – це матриця, яка містить значення 0 або мінус нескінченність ($-\infty$) залежно від того, чи дозволено звертатися до певного токена:

$$MaskedAttention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V \tag{1.46}$$

де M це маска. Якщо маска забороняє доступ до майбутніх токенів, то елементи заборонених позицій у матриці M мають значення $-\infty$. Коли такі значення додаються до результату обчислення QK^T, це змушує *softmax* перетворювати їх на нульову ймовірність.

Матриця маски може виглядати так:

$$M = \begin{pmatrix} 0 & -\infty & -\infty & \dots \\ 0 & 0 & -\infty & \dots \\ 0 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{1.47}$$

Це означає, що для кожного наступного кроку увага націлюється тільки на поточні або попередні токени, а майбутні залишаються заблокованими.

Після додавання маски процес обчислення уваги продовжується як звичайний Multi-Head Attention за формулою 1.40, де кожна «голова» $head_i$ обчислюється як:

$$head_i = MaskedAttention(QW_Q^{(i)}, KW_K^{(i)}, VW_V^{(i)}) \tag{1.48}$$

Результати всіх «голів» h конкатенуються і після цього проектуються через матрицю W_O, так само, як і в Енкодері, за формулою 1.40.

Маска забороняє увагу звертатися до майбутніх елементів у послідовності, що важливо для автогресивних моделей генерації тексту, де на кожному кроці потрібно генерувати новий токен на основі вже згенерованих.

Softmax з маскою ефективно зануляє ймовірності для майбутніх елементів, забезпечуючи правильну послідовність генерації.

3.3) Обчислення Multi-Head Attention з виходом Енкодера:

На цьому етапі вихід декодера взаємодіє з виходом енкодера. Тобто, вхід до другого шару декодера містить як інформацію про згенеровані частини цільової

послідовності, так і інформацію з енкодера (який обробляв аудіо або інші вхідні дані).

Обчислюються запити Q з виходу після self-attention декодера, але ключі K і значення V беруться з виходу енкодера:

$$\begin{aligned} Q &= output\ from\ self attention \\ K &= encoder\ output\ W_K \\ V &= encoder\ output\ W_v \end{aligned} \tag{1.49}$$

Після цього знову виконується Multi-Head Attention, але тепер на основі обох послідовностей

3.4) Feed Forward Layer:

Після двох блоків уваги вихід передається через feed forward мережу:

$$FFN(x) = ReLU(xW_1 + b_1)W_2 + b_2 \tag{1.49}$$

Це дозволяє моделі створювати складніші взаємозв'язки і навчати нелінійні залежності.

3.5) Нормалізація та додавання залишкового зв'язку:

Як і в енкодері, після кожного блоку уваги і feed forward, додається залишковий зв'язок (residual connection) та нормалізація:

$$output = LayerNorm(x + \frac{Attention}{FFN(x)}) \tag{1.50}$$

Загальний вигляд роботи декодера:

- Y_{input} проходить через masked self-attention для обробки вже згенерованої частини;

- Потім обчислюється Multi-Head Attention між виходом декодера і виходом енкодера;

- Після цього проходить через feed forward шар.

- Цей процес повторюється для кожного шару декодера.

4) Функція втрат (Loss Function):

Зазвичай для ASR на базі Transformer використовується функція втрат, така як крос-ентропійна втрата (cross-entropy loss) для порівняння передбачених символів з еталонними (транскрибованими) символами:

$$L = -\sum_{t=1}^{T} y_t log \hat{y}_t \quad (1.51)$$

де y_t – справжній символ на кроці;

$\hat{y}_t$ – передбачений символ.

5) Пошук променів (Beam Search):

Після передбачення моделі використовується алгоритм пошуку променів для знаходження найбільш імовірної послідовності слів:

$$BeamSearch(y_1, y_2, \dots, y_T) \quad (1.52)$$

Цей алгоритм обирає декілька найімовірніших шляхів на кожному кроці і дозволяє знаходити найкращу послідовність слів.

Transformer використовує багатошарову архітектуру уваги для перетворення послідовностей ознак аудіо у послідовності символів тексту. Математично це досягається через обчислення подібності між ознаками, позиційним кодуванням та механізмом уваги, що дозволяє фокусуватися на ключових моментах послідовності.

Transformer представляє собою найбільш передовий підхід у розвитку технологій автоматичного розпізнавання мовлення. Завдяки своїм перевагам у точності, паралелізації, гнучкості та контекстуальній обробці, ця модель стає все

більш популярною і затребуваною у сучасних ASR системах, забезпечуючи високу ефективність та продуктивність. Ця модель стала основою для багатьох передових ASR систем, таких як Google Speech-to-Text і Whisper, які демонструють високу продуктивність і точність.

1.4 Особливості сучасних ASR

У попередньому пункті було розглянуто еволюцію моделей ASR, яку проілюстровано на рис. 1.1. і було визначено, що модель Transformer для розпізнавання мовлення є найсучаснішою.

Рис. 1.1. Еволюція ASR

Розглянемо особливості сучасних ASR. Особливості сучасних систем ASR полягають у наступному:

- *точність розпізнавання мовлення:* сучасні системи ASR можуть досягати високого рівня точності, особливо завдяки використанню глибокого навчання, нейронних мереж та великих мовних моделей. Однак точність може залежати від якості запису, фонових шумів, акценту мовця та складності мовлення.

- *підтримка різних мов:* багато систем підтримують багатомовність, що дозволяє використовувати ASR для розпізнавання мовлення на різних мовах. Деякі системи можуть працювати з десятками мов, забезпечуючи універсальність застосування.

- *розпізнавання в реальному часі:* технології ASR дозволяють конвертувати мовлення у текст у реальному часі, що є важливим для голосових помічників, систем відеоконференцій та транскрибування розмов в режимі реального часу.

- *інтеграція з іншими технологіями:* ASR часто використовується у поєднанні з іншими технологіями штучного інтелекту, такими як обробка природної мови (NLP), для подальшого аналізу або взаємодії з користувачем. Це дозволяє створювати комплексні рішення для аналізу голосових даних, розпізнавання намірів користувача або автоматичного перекладу.

- *вимоги до ресурсів:* високоякісні моделі розпізнавання мовлення потребують значних обчислювальних ресурсів для навчання та обробки. Водночас сучасні хмарні сервіси, як-от Google або Microsoft, дозволяють використовувати ці потужності дистанційно, що спрощує інтеграцію технології в різні додатки.

- *підтримка різних форматів даних:* системи ASR можуть працювати з різними джерелами даних, такими як аудіофайли або потоки мовлення. Деякі з них навіть дозволяють обробляти зашумлені записи, використовувати попередню обробку аудіо для покращення якості розпізнавання.

- *адаптивність*: нові системи ASR мають можливість адаптуватися до особливостей голосу конкретного користувача або певної галузі (медичної, юридичної, технічної термінології), що підвищує точність розпізнавання у спеціалізованих сценаріях.

1.5 Проблеми та виклики STT технологій

Незважаючи на значний прогрес, технології Speech-to-Text все ще мають певні обмеження. Основні виклики включають:

- *низьку точність розпізнавання в умовах шуму:* фонові шуми або низька якість запису можуть суттєво знижувати точність перетворення мовлення на текст;

- *різноманітність акцентів та діалектів:* незважаючи на багатомовність, технології STT іноді стикаються з труднощами при розпізнаванні регіональних акцентів або діалектів;

- *великі вимоги до обчислювальних ресурсів:* сучасні моделі потребують значних обсягів даних для тренування та значних обчислювальних ресурсів до виконання;

- *необхідність у масштабованості*: оскільки сучасні моделі потребують багато обчислювальних ресурсів, то для розпізнавання великих обсягів даних зростає потреба у горизонтальній масштабованості на декілька машин, які повинні вміти працювати одночасно як одна система

Проте сучасні рішення постійно вдосконалюються, і ці проблеми поступово зменшуються з розвитком технологій машинного навчання та обробки природної мови.

1.6 Постановка задачі

Потрібно розробити таку програмну платформу, набір компонентів якої можна буде інтегрувати у інші програмні забезпечення. Для того, щоб інші програми могли легко взаємодіяти з розробленою програмною платформою, має бути розроблений інтерфейс для такої взаємодії. Програмна платформа може бути побудована на основі існуючих ASR чи використовувати їх.

Програмна платформа має виконувати таку роботу:

- отримувати від користувачів аудіодані, та вхідні параметри, що необхідні для системи ASR, яка бути транскрибувати ці дані. Такими параметрами можуть

бути, наприклад: мова, кількість спікерів, формат отриманого результату тощо.

- отримані аудіодані від користувача будуть транскрибуватися згідно вхідних параметрів, які надав користувач;

- користувач отримує прогрес виконання його завдання, бачить його статус і може відмінити його у будь-який момент;

- у кінці виконання завдання користувач отримує результат – транскрибований текст із аудіо або помилку у разі неуспішності виконання завдання;

- для можливості ведення звітності дані про завдання мають зберігатися у базі даних.

Для цього потрібно виконати наступне:

1) Провести аналіз існуючих сучасних систем ASR і зробити вибір тієї системи, що буде задовольняти таким вимогам:

- відкритість для використання та розробки;

- наявність зрозумілої документації;

- підтримка та оновлення системи;

- підтримка необхідних мов розпізнавання;

- висока точність і швидкість транскрибування;

- можливість включення контексту в процес розпізнавання;

- стійкість до шуму;

- розпізнавання спікерів.

Для порівняння систем ASR потрібно підготувати аудіофайл, з якого необхідно розпізнати мовлення та порівняти результати.

2) Визначити архітектуру програмної платформи. Має бути зовнішній інтерфейс для взаємодії з клієнтами. Далі запит користувача має передаватися у певний об’єкт, який бути виконувати завдання. Клієнт повинен мати змогу спостерігати та управляти повним циклом процесу.

3) Визначитися з мовою програмування та середовищем, у якому буде працювати майбутня платформа. Для досягнення оптимальної продуктивності необхідно обрати мову програмування, що відповідає сучасним вимогам у сфері

обробки мовлення та розподілених систем. Перевагу можна віддати мовам, що підтримують багатозадачність та ефективну роботу з великими даними. Вибір операційної системи залежатиме від вимог до середовища розгортання та використання контейнерів.

4) Обрати базу даних. Залежно від вимог до збереження даних і їх обробки можна обрати реляційну базу даних (наприклад, PostgreSQL) або нереляційну (наприклад, MongoDB).

1.7 Висновки до розділу

У межах першого розділу даної роботи розглянуто актуальність систем ASR, зокрема STT, як частини ASR. У теперішній час є великий попит на такі системи в багатьох галузях: від медичної до аналітичної, юридичної чи технологічної. Система, яка може масштабуватися і обробляти задачі через розподілені клієнти на різних ПК з використанням сучасних протоколів взаємодії, є не тільки актуальною, але й необхідною для багатьох організацій та підприємств.

Було приділено увагу на еволюцію моделей розпізнавання мовлення, а також розглянуто з математичної точки зору як вони працюють. Очевидно, що для розробки сучасних рішень потрібно обирати більш сучасні системи ASR, а саме ті, які працюють на основі найсучаснішої та найрозвинутішої моделі Transformer. Це найсучасніша модель, яка працює з усіма елементами входу одночасно, що дозволило досягти високої точності розпізнавання мовлення із аудіо. І, на відміну від усіх попередніх моделей, цій моделі вдалося вловити віддалені залежності між різними частинами мови, що дозволяє їм враховувати ширший контекст кожного транскрибованого речення. Завдяки цьому Transformer є потужнішою моделлю в багатьох задачах розпізнавання мовлення, зокрема у відношенні точності та швидкості.

Так як сучасні системи розпізнавання мовлення набули значного розвитку, і точність їх результатів наблизилася до еталонної, то попит на розпізнавання мовлення у теперішній час значно збільшився у багатьох галузях. І таким чином

актуальним завданням є розробка програмної платформи, яку можна буде інтегрувати у інші програмні продукти. Таке рішення може бути корисним для багатьох організацій та підприємств. Таку розробку можна зробити на основі існуючих моделей розпізнавання мовлення, яку можна обгорнути інтерфейсом для зовнішнього користування та можливості інтеграції з іншими сервісами на підприємствах різних галузей. Тому перш за все необхідно розібратися з необхідними критеріями вибору для найбільш оптимальної системи та провести вибірковий аналіз сучасних ASR на основі визначених критеріїв. Необхідно буде врахувати всі залежності для повноцінної роботи обраної системи розпізнавання мовлення та визначитися з платформою для розробки і запуску продукту.

2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ПРАКТИЧНИХ АСПЕКТІВ РОЗПІЗНАВАННЯ ТЕКСТУ

2.1 Визначення критеріїв для оцінки існуючих ASR систем на основі архітектури Transformer

У першому розділі було визначено що найсучаснішою архітектурою ASR моделей є Transformer. Сучасні ASR моделі можна використовувати в командному рядку як окремі повноцінні інструменти, що додають їм функціональність самостійних систем ASR. Будемо розглядати саме такі моделі як самостійні інструменти, тому будемо називати їх системами розпізнавання мовлення. Необхідно визначитися з критеріями вибору існуючих рішень для розпізнавання мовлення на основі архітектури Transformer для того, щоб обрана система, яка буде чи не найбільш основною частиною майбутнього продукту, була якомога ефективнішою. Далі розглянемо основні критерії, які допоможуть оцінити існуючі ASR системи на основі Transformer, забезпечуючи оптимальний вибір платформи для побудови ефективної програмної платформи.

2.1.1 Відкритість для використання та розробки

Даний критерій визначає, наскільки легко ASR система може бути інтегрована та адаптована під конкретні потреби. Відкритий код дозволяє модифікувати систему, робити її кастомізацію та інтегрувати у сторонні рішення, що є важливим для гнучкості та довгострокової підтримки проекту. Для цього потрібно проаналізувати ліцензії, під якими поширюються ASR системи. Відкриті ліцензії, такі як Apache, MIT, або GPL, дозволяють модифікацію та інтеграцію. Також важливо оцінити доступність коду на платформах типу GitHub та можливість інтеграції до власного проекту.

Перелік показників даного критерію:

- ліцензія (відкрита чи закрита);

- доступність репозиторію на GitHub;
- легкість інтеграції та налаштування.

2.1.2 Документація та підтримка

Якісна документація і активна підтримка розробників та спільноти забезпечують ефективне використання системи. Це допомагає швидко вирішувати проблеми, зрозуміти архітектуру та можливості системи, а також підтримувати її актуальність завдяки регулярним оновленням.

Оцінка документації проводиться через перегляд офіційної документації системи (вона має бути зрозумілою та повною). Також важливо перевірити наявність спільноти розробників та частоту оновлень системи.

Перелік показників даного критерію:

- наявність офіційної документації;
- активність на форумах (наприклад, Stack Overflow, Reddit та ін.);
- частота оновлень у GitHub;
- наявність підтримки (як безкоштовної, так і платної).

2.1.3 Підтримка мов та багатомовність

Для глобальних проектів, або тих, які потребують обробки мовлення на різних мовах, важливою є можливість роботи з багатьма мовами. Цей критерій дозволяє обрати систему, яка найкраще відповідає вимогам щодо підтримки різноманітних мов та діалектів.

Оцінити системи можна через перелік підтримуваних мов в офіційній документації або на веб-сайті проєкту. Додатково, можна протестувати систему на різних мовах для перевірки якості розпізнавання кожної з них.

Перелік показників даного критерію:

- кількість підтримуваних мов;

- підтримка основних мов (українська, англійська, іспанська тощо), або мови, які точно необхідні для використання;

- наявність можливості додавати нові мови;

- адаптація системи під нові діалекти.

2.1.4 Точність та швидкість

Точність та швидкість є ключовими показниками ефективності ASR системи. Ці параметри впливають на якість розпізнавання мовлення, особливо у випадках з великими обсягами аудіоданих або в режимі реального часу. Точна та швидка система дозволить користувачам отримувати результати без значних затримок та з мінімальними помилками.

Точність можна оцінити через тестування системи на стандартизованих тестових наборах аудіофайлів і порівняння результатів з реальними транскрипціями. Для швидкості важливо вимірювати час, необхідний для обробки однієї хвилини аудіо. Оцінку точності часто проводять через метрики, такі як Word Error Rate (WER).

WER — це метрика, що використовується для оцінки точності ASR. Вона показує, наскільки транскрибований текст відрізняється від реального, шляхом підрахунку кількості помилок. WER розраховується за формулою:

$$WER = \frac{S+D+I}{N} \tag{2.1}$$

де S *(Substitutions)* – кількість замін неправильних слів (коли одне слово помилково замінене іншим);

D *(Deletions)* – кількість пропущених слів у транскрипції;

I *(Insertions)* – кількість зайвих доданих слів;

N – загальна кількість слів у правильній транскрипції (референтний текст).

Перелік показників даного критерію:

- WER (відсоток помилок у словах);
- час обробки аудіо на одиницю тривалості;
- порівняння результатів з іншими системами на тих самих тестових наборах.

2.1.5 Стійкість до шуму

Багато реальних аудіозаписів містять шум або інші перешкоди. Системи з високою стійкістю до шуму здатні забезпечувати точне розпізнавання мовлення навіть у складних акустичних умовах, що є важливим для застосувань у реальних сценаріях (наприклад, дзвінки, інтерв'ю).

Проведення тестів на аудіозаписах з різними рівнями шуму. Для цього можна використовувати стандартизовані аудіо датасети або створити штучні приклади, додаючи фоновий шум. Оцінюється точність розпізнавання мовлення у таких умовах.

Перелік показників даного критерію:

- WER на аудіо з фоновими шумами;
- порівняння результатів з чистими аудіофайлами;
- рівень допустимого шуму, при якому система все ще функціонує на прийнятному рівні точності.

2.1.6 Розпізнавання спікерів

Ця характеристика є важливою для систем, які працюють з багатоспікерськими даними або потребують індивідуальної адаптації до голосів користувачів. Це забезпечує високу точність у контексті багаторазового використання системи конкретними спікерами.

Система повинна бути протестована на аудіо з кількома спікерами. Оцінка розпізнавання спікерів може здійснюватися через тестування на аудіозаписах з

різними голосами та перевірку здатності системи коректно ідентифікувати спікерів. Адаптація оцінюється шляхом аналізу навчання системи на нові голоси.

Перелік показників даного критерію:

- точність розпізнавання окремих спікерів у діалогах;
- здатність до адаптації до нового голосу після кількох аудіофрагментів;
- можливість автоматичного визначення кількості спікерів.

2.1.7 Включення контексту

Ефективне врахування контексту під час розпізнавання мовлення дозволяє уникнути помилок у випадках, коли зміст фрази залежить від попередніх речень. Це покращує точність результатів, особливо при обробці довгих текстів або діалогів.

Для оцінки цього критерію можна тестувати систему на складних фразах або довгих діалогах, де контекст має значення для правильного розпізнавання. Оцінюється здатність системи правильно інтерпретувати слова або фрази, використовуючи попередню інформацію з потоку мовлення.

Перелік показників даного критерію:

- точність розпізнавання складних структурованих речень;
- результативність у випадках, де розпізнавання залежить від попереднього контексту.

2.2 Найвідоміші існуючі системи розпізнавання мовлення, що базуються на архітектурі Transformer

Серед найвідоміших Open-Source систем розпізнавання мовлення, які базуються на архітектурі Transformer є дві моделі: Whisper [7] від OpenAI та Wav2Vec 2.0 [9] від Meta AI. Обидві системи є розробками від всесвітньо відомих провідних компаній, які мають значний вплив у сфері штучного інтелекту, що значно пришвидшує прогрес у сфері обробки мови, оскільки великі ресурси, знання

та досвід цих організацій забезпечують високу точність, функціональність та масштабованість цих систем у реальному світі.

Обидві системи є найвідомішими на найсучаснішими, окрім їх розробників, через кілька ключових факторів:

- *велика кількість даних для навчання:* Whisper була навчена на 680,000 годин багатомовного та багатозадачного навчального датасету [8]. Wav2Vec 2.0 була навчена на понад 50 000 годин нерозміченого мовлення. Велика кількість даних дозволяє цим системам краще розпізнавати різні акценти, фоновий шум та технічну мову;

- *Висока точність:* Обидві системи досягли високої точності розпізнавання мовлення. Whisper показує високу стійкість до акцентів, фонового шуму та технічної мови. На рис. 2.1 показано, що у системі Whisper більшість мов мають низький рівень WER (<50%), а основні мови – взагалі менше 10%, і це є дуже гарним показником. Wav2Vec 2.0 також досягає низького рівня помилок у словах (WER) навіть на чистих даних;

- *Відкрита доступність:* Обидві системи є Open-Source, що дозволяє розробникам легко інтегрувати їх у свої проекти та додавати нові функції [10][11]. Це сприяє їх популярності серед спільноти розробників;

- *Підтримка багатомовності:* і Whisper, і Wav2Vec 2.0 підтримують багатомовність (понад 100 мов), що робить ці системи корисними для різних географічних регіонів та показує їх важливість для міжнародних застосувань [12]. На рисунку 2.1 видно як багато мов підтримує Whisper, та який гарний показник WER щодо цих мов він має.

Обидві системи використовують передові технології, такі як контрастивне навчання у випадку Wav2Vec 2.0 та сильно навчені моделі у випадку Whisper. Але вони мають різні підходи до їх архітектури:

- Whisper орієнтована на обробку аудіо з текстовим супроводом і пропонує широкий набір можливостей, таких як багатомовна підтримка та транскрипція. Вона також має спеціальну архітектуру для роботи з різними мовами та умовами.

- Wav2Vec 2.0 використовує метод навчання без нагляду, де модель спершу вчиться розпізнавати звукові характеристики без розмітки тексту, що дозволяє їй краще узагальнювати результати на різних мовах та ситуаціях.

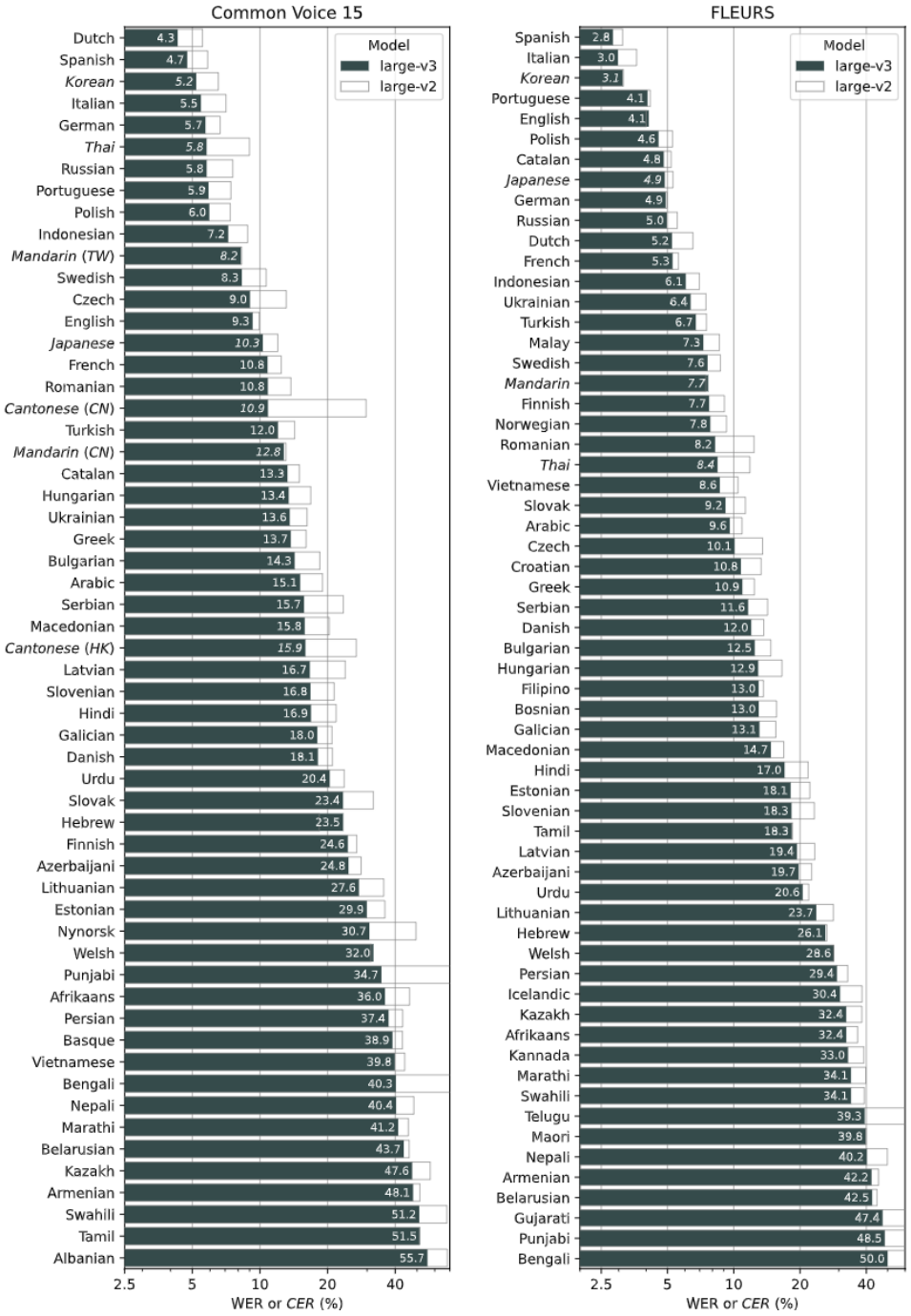


Рис. 2.1. Порівняння показників WER щодо різних мов у Whisper [10]

І Whisper, і Wav2Vec 2.0 є лідерами в галузі автоматичного розпізнавання мовлення. Тому їх можна розглядати як кандидати для вибору системи ASR,як за основу для інтеграції для розробки згідно поставленої задачі.

2.2 Вибірковий аналіз найбільш оптимальної ASR системи між Whisper та Wav2Vec 2.0 на основі визначених критеріїв оцінювання таких систем

У роботі [13] проведено порівняння трьох найвідоміших ASR систем: Kaldi, Wav2Vec 2.0 та Whisper. Kaldi побудована на основі моделей HMM+GMM для моделювання акустичних систем і для покращення точності також може використовувати DNN та RNN. Ця система менш зручна у використанні, і побудована більш академічним кодом, який не дуже оптимізований під enterprise рішення. Ще варто зазначити, що у роботі [13] Kaldi тестували на основі навченої на наборі даних Gigaspeech XL з використанням RNN і при цьому всьому, забігаючи наперед, у роботі [13] Kaldi показала хоч і гарні, але найгірші результати. Тому будемо вважити що Kaldi є застарілою у порівнянні з Whisper та Wav2Vec 2.0, і тому ми її пропустимо у даній роботі.

Щоб результати були більш справедливими, тестування у роботі [13] проводилося на різних підготовлених наборах аудіо файлів, що мають різну якість записаного звуку та різний рівень шуму. Для збільшення точності використовувалося по декілька аудіо файлів на кожну із типів розмов. Для кожного з аудіо файлів, над якими проводився дослід, була підготовлена правильна та якісна транскрипція, яка порівнювалась з отриманими транскрипціями від Whisper та Wav2Vec 2.0, згідно яких за формулою 2.1 обчислювався показник WER. Загальна характеристика аудіо файлів наведена у таблиці 2.1.

Таблиця 2.1

Характеристики підготовлених аудіо файлів у роботі [13] для порівняння ASR систем Whisper та Wav2Vec 2.0

Назва набору аудіо файлів	**Середня тривалість аудіо файлів, (сек)**	**Опис**
Діалог людини з штучним інтелектом	39	Спонтанна розмова людини з голосовим чат-ботом. Аудіо у складних умовах з шумом та фоновою мовою. Багато коротких файлів.

Запис телефонної розмови	132	Записи різних спонтанних телефонних розмов. Включно з випадками колцентру. Загалом аудіо низької якості 8 кГц із фоновою музикою
Зустрічі та конференції	2166	Спонтанна розмова під час зустрічей із кількома особами, включаючи особисті зустрічі та відеоконференції. Файли, як правило, довгі з великою кількістю мовців.
Відео	174	Відео з Інтернету та інших джерел із різноманітним вмістом. Переважно спонтанна мова. Загалом високоякісний звук із високою частотою дискретизації.
Телефонна або відео конференція	2370	Виразне мовлення учасників. Багато фінансового жаргону та розмовних числівників. Довга тривалість аудіо.

У таблиці 2.2 відображений загальний показник WER для кожного набору аудіо файлів, який дає «макроскопічне» уявлення про ефективність моделі для кожної вибірки. Даного результату достатньо, щоб зрозуміти яка з систем точніше справляється із задачею розпізнаванні мовлення.

Таблиця 2.2

Загальний WER транскрипцій отриманих у роботі [13] за допомогою ASR систем Whisper та Wav2Vec 2.0

Назва набору аудіо файлів	**Whisper, (WER)**	**Wav2Vec 2.0, (WER)**
Діалог людини з штучним інтелектом	19.9	36.3
Запис телефонної розмови	16.6	31.0
Зустрічі та конференції	13.9	27.4
Відео	9.7	28.1
Телефонна або відео конференція	8.9	23.3

Згідно результатів обчислення WER для кожної з моделей ASR у роботі [13] виходить такий простий висновок, що Whisper є більш точною системою, ніж

Wav2Vec 2.0, адже по всім параметрам результати у Whisper кращі, ніж у Wav2Vec 2.0. Саме тому будувати свою платформу на основі ASR Whisper.

2.3 Перехід до більш швидкої системи WhisperX з додатковою можливістю розпізнавання спікерів

У деяких галузях швидкість обробки мовлення важливіша, ніж абсолютна точність. Це відбувається тому, що:

- *оперативність прийняття рішень* – у певних сценаріях, швидкість реакції на аудіо чи мовні команди може бути критичною. Навіть якщо є деякі неточності, важливо швидко отримати загальну картину для дії;

- *висока частота подій* – коли аудіо обробляється в реальному часі з високим потоком даних, намагатися досягти ідеальної точності може бути нераціонально з точки зору ресурсів і часу.

Такими галузями, де швидкість переважає точність розпізнавання мовлення, можуть бути: колцентри, новинні агентства, торгівельні платформи тощо. Але й у деяких випадках має вирішальне значення саме точність розпізнавання мовлення, оскільки навіть найменші помилки можуть призвести до серйозних наслідків, особливо у юридичній, медичній чи фінансовій сферах.

Іншою важливою проблемою є проблема «вирівнювання» результату транскрипції. «Вирівнювання» (alignment) у контексті розпізнавання мовлення – це процес, під час якого транскрибований текст (вихідна форма) узгоджується з відповідними сегментами аудіо (вхідними даними). Простими словами, «вирівнюванням» можна назвати процес, який допомагає точно визначити часові мітки кожного сегменту тексту – речення або слова. «Вирівнювання» важливе для:

- *створення субтитрів:* для того, щоб текст на екрані з'являвся максимально синхронно з мовленням у фільмах або відео;

- *аналізу даних:* для того, щоб краще розуміти, коли відбуваються певні події в аудіозаписі;

- *покращення моделей:* потрібно для тренування ASR моделей для того, щоб вони могли точніше розпізнавати і узгоджувати слова з відповідними моментами в аудіо.

Обрана у попередньому пункті ASR модель Whisper демонструє надзвичайно приголомшливі результати у різних областях і різними мовами. Але у обраної моделі є певний недолік – часові мітки є прогнозованими і вони є не для кожного окремого слова, а для фрази, речення. У статті [14] вирішили даний недолік у представленій системі WhisperX [15]. WhisperX – це модифікація моделі Whisper від дослідників з Оксфордського Університету. Дана модифікація включає попередню сегментацію звука за допомогою запропонованої стратегії «VAD Cut & Merge», яка не тільки покращує «вирівнювання», а ще дозволила прискорити процес розпізнавання мовлення у 12 разів.

Розпізнавання часових міток на рівні окремих слів у WhisperX відбувається у наступні три етапи:

1) Попередня сегментація вхідних даних з аудіо файлу за допомогою зовнішньої моделі виявлення голосової активності (VAD);

2) Розділення аудіо файлу на частини по виявленій голосовій активності, і потім об’єднування отриманих сегментів аудіо у вхідні фрагменти тривалістю приблизно і не більше, ніж 30 секунд з границями, що лежать у найменш активних голосових мовленнєвих регіонах. Дані 30-секундні фрагменти можна транскрибувати паралельно, без втрат контексту, що дозволило прискорити процес розпізнавання мовлення;

3) Примусове вирівнювання за допомогою використання зовнішньої моделі розпізнавання фонем, яка навчена розпізнавати найменшу одиницю мовлення відрізняючи одне слово від іншого. Дана модель запозичена із системи Wav2Vec 2.0, про яку вже відомо із попередніх пунктів.

Використання сегментації аудіо файлу за допомогою VAD також дозволило виконувати діаризацію – виявлення спікерів. Отримані сегменти ми можемо передавати у окрему модель, яка вміє аналізувати їх і виявляти різних мовців. І це є ще однією перевагою WhisperX над стандартним Whisper.

Загальний принцип роботи WhisperX продемонстрований на рис. 2.2 [14]:

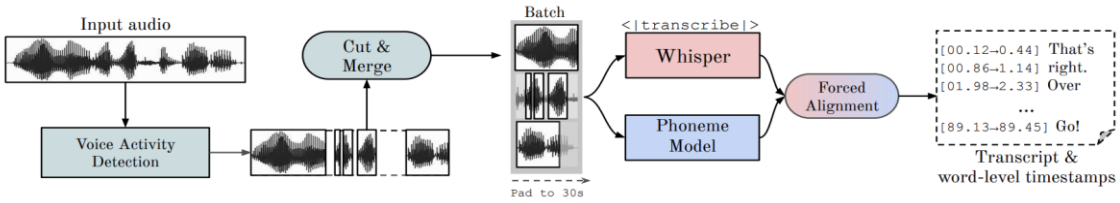


Рис. 2.2. Загальний принцип роботи WhisperX

Згідно статті [14] у таблиці 2.3 наведено порівняння показника WER та швидкості транскрибування між моделями Whisper, Wav2Vec 2.0 та WhisperX на одному і тому ж наборі аудіо.

Таблиця 2.3

Порівняння Whisper, Wav2Vec 2.0 та WhisperX

Показник	**ASR Моделі**		
	Wav2Vec 2.0	**Whisper**	**WhisperX**
WER	19.8	9.7	10.5
Коефіцієнт швидкості	10.3	1.0	11.8

Таблиця 2.3 не охоплює різні набори аудіо набори, а також не містить порівняння якості транскрибування з використанням та без використання VAD. Все це детально є описано у статті [14]. Але навіть виходячи з результатів із цієї таблиці можемо зробити висновок, що, простими словами, WhisperX – так само точний, як Whisper, і так само швидкий, як Wav2Vec 2.0. Тому було прийнято рішення, все-таки, за основу для задачі розпізнавання мовлення у розробці програмної платформи обрати саме ASR модель WhisperX.

2.4 Відповідність WhisperX до поставлених критеріїв вибору

У першому пункті цього розділу були визначені критерії щодо вибору оптимального інструменту для автоматичного розпізнавання мовлення із аудіо у текст. Основними критеріями вибору були:

- відкритість для використання та розробки;

- якісна документація і активна підтримка розробників та спільноти;

- можливість роботи з багатьма мовами;

- точність та швидкість розпізнавання мовлення;

- стійкість до шуму;

- можливість розпізнавання спікерів та адаптація;

- включення контексту.

Обраний інструмент WhisperX задовольняє всім визначеним критеріям:

- побудований на відкритих моделях штучного інтелекту та доступний через платформи, такі як GitHub та Hugging Face, що робить цю систему відкритою для використання і модифікації;

- хоч документація WhisperX може бути менш розвиненою порівняно з основною моделлю Whisper, але в самому репозиторії є приклади використання з поясненнями і цього достатньо для розробки;

- цей інструмент є достатньо відомим, має високі показники активності на платформі GitHub;

- останнє глобальне оновлення на платформі GitHub на момент написання цієї роботи було 13.05.2023 року але при цьому періодично вносяться правки та доповнення – останній commit відбувся 19.08.2024 автором на запит в pull-request від спільноти репозиторію, що є досить близькою датою до написання цієї роботи, що свідчить про гарну підтримку та прогнозує на подальший розвиток;

- підтримує всі мови, що й оригінальний Whisper від OpenAI (понад 100 мов);

- працює так само точно, як і Whisper, і так само швидко, як і Wav2Vec 2.0;

- стійкий до шуму так само, як і найпередовіша та найсучасніша модель Whisper;

- використання сегментації вхідних даних з аудіо файлу за допомогою VAD дозволяє використовувати інструмент pyannote.audio speaker diarization toolkit, який достатньо точно вміє розпізнавати спікерів;

- розпізнавання мовлення у WhisperX побудоване на архітектурі Transformer, що саме собою вміє працювати з контекстом розмови. WhiperX, як і оригінальний Whisper чудово справляється з цією задачею.

Все наведене вище свідчить про те, що обрана ASR система WhisperX є найбільш оптимальним інструментом для розпізнавання мовлення, на основі якого буде розроблено програмну платформу.

2.5 Висновки до розділу

У другому розділі даної роботи потрібно було підібрати найоптимальнішу сучасну ASR, яка побудована на сучасній архітектурі Transformer, яка була визначена у першому розділі даної роботи.

Для цього були визначені основні критерії вибору існуючих рішень для розпізнавання мовлення на основі моделі Transformer для того, щоб обрана система була якомога ефективнішою. Були надані оцінки до ASR на основі архітектури Transformer, які розглядалися щодо їх вибору, таким чином, щоб вибір був оптимальним для побудови масштабованої та ефективної програмної платформи.

Були розглянуті такі ASR, що основані на архітектурі Transformer: Whisper від OpenAI, Wav2Vec 2.0 від MetaAI, та WhisperX – модифікація Whisper.

Вибір впав саме на останню – WhisperX. Було підтверджено, що дана модель автоматичного розпізнавання мовлення задовольняє всім визначеним критеріям.

3 КОНЦЕПЦІЯ ТА РЕАЛІЗАЦІЯ РОЗРОБКИ ПРОГРАМНОЇ ПЛАТФОРМИ ДЛЯ РОЗПІЗНАВАННЯ ТЕКСТУ

3.1 Визначення архітектури програмної платформи

Система розпізнавання мовлення у даній розробці є «серцем» даної платформи, тому що саме цей елемент буде виконувати найбільш основну функцію – розпізнавати мовлення із аудіо у текст. Тому для визначення архітектури потрібно відштовхуватися саме від обраної ASR системи. У якості ASR системи було обрано WhisperX, яка написана на Python і є кросплатформою, що дозволяє їй працювати на таких ОС, як Linux, Windows, MacOS. Тому, відповідно, розроблена програмна платформа теж може і має бути кросплатформовою. Хоч і потрібно під час розробки враховувати всі нюанси до кожної з ОС, на яких планується запускати розроблений продукт, та загальна архітектура буде однаковою для всіх них. Для розробки необхідно визначити ключові компоненти платформи, визначити як та за допомогою чого їх розробити, оскільки важливо чітко розуміти, як окремі частини системи будуть взаємодіяти між собою, щоб забезпечити стабільність, надійність і продуктивність роботи системи.

3.2 Опис ключових компонентів платформи

Для визначення архітектури програмної платформи необхідно розібратися з її ключовими компонентами. Для того, щоб зрозуміти, які компоненти необхідні, для цього потрібно відповісти на такі запитання:

1) «Що ми хочемо отримати?»;
2) «Що для цього потрібно зробити?»;
3) «Що ми маємо для цього?».

У контексті даної роботи ми можемо відповісти на поставлені запитання так:

1) «Ми хочемо отримати якомога точну транскрипцію мовлення у текстовому форматі із аудіо файлу, та ще й так, щоб розміти у який момент й хто говорив»;

2) «Для цього ми повинні передати системі розпізнавання мовлення аудіо файл та надати початкові дані, необхідні для процесу і потім отримати результат у вигляді тексту, і при цьому ми повинні мати змогу відстежувати даний процес з можливістю керувати ним»;

3) «Ми маємо: аудіо файл з якого потрібно отримати транскрипцію та завантажений репозиторій системи розпізнавання мовлення WhisperX, у якого ми передаватимемо аудіо файл, а точніше повний шлях до нього, і також ми передаватимемо, разом з повним шляхом до аудіо файлу, вхідні параметри, необхідні для ініціалізації та виконання процесу».

Отже першим і найголовнішим ключовим компонентом програмної платформи є безпосередньо сам WhisperX.

Командного рядку, який є присутній у WhisperX недостатньо для більш гнучкого використання WhisperX. Тому навколо WhisperX має бути інтерфейс, який приймає від зовнішнього середовища необхідні вхідні дані та налаштування. Це буде скрипт, який запускає процес розпізнавання мовлення на основі отриманих вхідних даних та налаштувань і в результаті отриману транскрипцію мовлення записує у вихідний файл. Так, як ще однією задачею у WhisperX є розпізнавання спікерів (діаризація), то для цього процесу також у даному скрипті має бути врахована підтримка даної задачі.

Інтерфейсу у вигляді командного рядку не достатньо для інтеграції у інші програмні продукти. Більшість сучасних рішень використовують для поєднання з іншими програмними продуктами набір чітко визначених методів для взаємодії різних компонентів – API. Це спосіб завдяки якому дві або більше комп’ютерні програми можуть спілкуватися між собою [19]. У нашому випадку програмна платформа за допомогою API буде отримувати вхідні дані від клієнтів, а потім ці дані буде передавати у інтерфейс WhisperX.

Більшість сучасних рішень мають API інтерфейс на основі HTTP-запитів. Тобто використовують WEB API для того, щоб ці рішення могли бути

інтегрованими у інші рішення. Саме тому API інтерфейс для нашої платформи будемо будувати на основі WEB технологій. Для цього будемо використовувати платформу NodeJS з бібліотекою Express.

Використання Node.js та Express для створення API інтерфейсу для платформи розпізнавання мовлення на основі WhisperX виправдане з кількох причин:

- *легка інтеграція та швидка розробка:* Node.js, завдяки асинхронній обробці та неблокуючій моделі введення-виведення, забезпечує швидке виконання запитів, що є важливим для API, який повинен обробляти значну кількість запитів на розпізнавання мовлення. Це дозволяє швидше масштабувати систему і забезпечує стабільність при високих навантаженнях;

- *ефективна робота з мережею та підключенням до баз даних:* У платформах, що залежать від даних, Node.js та Express ідеально підходять для роботи з різними базами даних (як, наприклад, MongoDB) завдяки своїй природній здатності обробляти численні одночасні з’єднання;

- *широка екосистема та наявність пакетів:* Express має потужну екосистему пакетів та модулів, які спрощують розробку API, забезпечуючи стандартизацію, безпеку та легкість реалізації, а також надає доступ до численних інструментів. Це особливо важливо для побудови API, який надає доступ до розпізнавання мовлення через інтерфейс для клієнтів;

- *універсальність та сумісність:* Express легко інтегрується з іншими мовами та технологіями, що дозволяє вам використовувати Python у компоненті для виконання розпізнавання мовлення з WhisperX. Node.js забезпечує гнучкість, тому що ми можемо легко запускати Python-скрипти через Node.js для завдань розпізнавання;

- *кросплатформеність та легкість розгортання:* Node.js підтримує кросплатформове середовище, що важливо для забезпечення сумісності системи з різними ОС, такими як Linux, Windows чи MacOS, що робить її універсальною для використання різними клієнтами в різних середовищах;

- *простота у налаштуванні:* Express дозволяє швидко та ефективно створити API, який забезпечує безперервну інтеграцію з іншими сервісами, необхідними для платформи розпізнавання мовлення, і підтримує високий рівень продуктивності та масштабованості.

За допомогою WEB API з'являється можливість відокремлення даної платформи на окрему машину. Це є дуже важливо, тому що ASR потребують багато обчислювальної потужності, а при цьому не всі рішення, у яких буде потреба в інтеграції даної програмної платформи, можуть забезпечити це.

Так як для розпізнавання мовлення із аудіо файлів необхідно на машині, де це буде відбуватися, зберігати ці аудіо файли. Не обов'язково збереження аудіо файлів має бути саме на машині, яка розпізнає мовлення, наприклад це може бути мережеве файлове сховище, але обов'язково ця машина повинна мати доступ до цих файлів у мережі. WEB API забезпечить отримання цих аудіо файлів від клієнтів до файлового сховища.

У нашій платформі API буде відповідати за такі задачі:

1) створення нових записів про нові завдання в базі даних та передавання їх до черги для обробки Воркерами;

2) прийом аудіо файлів від клієнтів для їх збереження на файловому сховищі;

3) передачу команди на відміну виконання завдання від клієнта до черги «відміни виконання завдань»;

4) моніторинг оновлення прогресу виконання завдання в реальному часі за запитом клієнта;

5) віддачу результату виконання завдання клієнту.

Так як визначено, що NodeJS вміє легко запускати Python-скрипти (за допомогою модуля «child_process»), так само, як у командному рядку, то тоді іншим ключовим компонентом, який знаходиться всередині компоненту API є сервіс, який запускає Python-скрипт для розпізнавання мовлення і діаризації і передає в нього вхідні дані від API.

Розберемося, які вхідні дані необхідні для розпізнавання мовлення із аудіо файлу у WhisperX. Вхідними параметрами для процесу розпізнавання аудіо файлу у системі WhisperX можуть бути такі:

- повний шлях до аудіофайлу (audio_file) – це обов'язковий параметр, без нього система не буде розуміти який файл необхідно обробляти. Так як аудіо файли зберігаються у сховищі, то клієнт може не знати у якому саме місці на сховищі знаходиться аудіо файл. Тому клієнт відправляє просто назву аудіо файлу, а повний шлях до цього файлу має забезпечити сама програмна платформа;

- мова (language) – за замовчуванням WhisperX сам автоматично розпізнає мову спікера;

- мінімальна кількість спікерів (min_speakers);

- максимальна кількість спікерів (max_speakers);

- точна кількість спікерів (num_speakers);

- мовленнєва модель Whisper (model_name) – на даний момент найсучаснішою та найбільшою моделлю, яка підтримує дуже багато мов і за допомогою якої WhisperX розпізнає текст з найбільш високою точністю є «large-v3»;

- точність обчислювань (compute_type) – необхідно підбирати відповідно до технічних параметрів апаратної частини машини, на якій відбуваються математичні процеси розпізнавання мовлення.

- назва файлу для вихідних результатів (captions_file);

Набір вхідних параметрів, що надається системі ASR, назвемо завданням (Task), яке необхідно виконати системі ASR, з якого необхідно отримати результат у вигляді транскрибованого тексту мовлення із аудіо файлу.

Звісно, для можливості відстеження статусу виконання, зберігання результатів та забезпечення збереження інформації про завдання необхідною також є база даних (БД). У БД зберігатимуться метадані завдань розпізнавання мовлення. У якості БД будемо використовувати MongoDB з кількох причин:

- *документо-орієнтована структура:* MongoDB використовує JSON-подібні документи для зберігання даних, що дозволяє гнучко зберігати різні типи завдань і параметрів без необхідності створювати складні схеми.

- *легке масштабування:* завдяки вбудованим механізмам шардінгу, MongoDB дозволяє масштабувати базу горизонтально, що відповідає потребам платформи.

- *висока швидкість операцій з великими обсягами даних:* MongoDB оптимізована для роботи з великими наборами даних, що є важливим для обробки великої кількості аудіо та текстових файлів;

- входить до стеку MERN [21].

Крім вхідних параметрів для ініціалізації процесу розпізнавання мовлення до завдання мають входити ще такі параметри, як:

- дата та час створення завдання;

- дата та час останнього оновлення виконання завдання;

- статус виконання завдання;

- прогрес виконання завдання у процентах.

Прогрес виконання завдання у більшості випадків необхідний лише під час самого процесу виконання. Прогрес оновлюється багато разів і його значенням може буде числове значення від 0 до 100. Таке постійне оновлення запису завдання у базі даних може збільшити навантаження на саму базу даних. Тому слід подбати про оперативний високо продуктивний кеш бази даних. Тоді прогрес виконання буде постійно записуватися спочатку у кеш, а до БД лише тоді, коли зміниться статус виконання, тобто під час якихось глобальних змін виконання завдання. Кеш завдань здійснюватиметься за допомогою Redis – оперативна короткочасна база даних, завданням якої є зменшити навантаження на основну базу даних за допомогою зберігання тимчасових даних, такі як процент прогресу виконання завдання, який оновлюється багато раз і який корисний лише під час виконання завдання (кешування). Здійснюється за допомогою Redis. Додатковою опцією Redis є відправка повідомлень між компонентами системи про оновлення інформації виконання завдань у реальному часі;

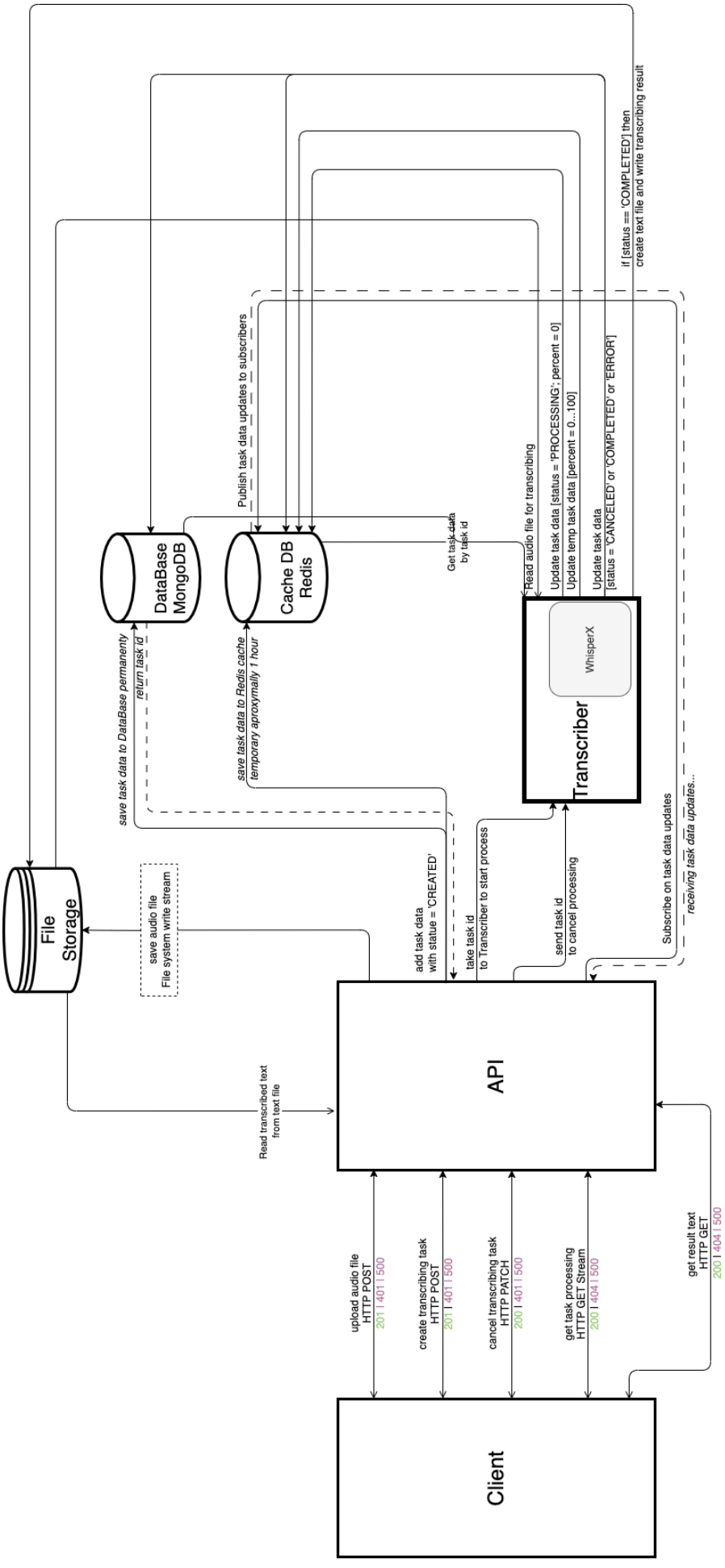


Рис.3.1 Схема програмної платформи з її ключовими компонентам

Отже, підсумовуючи з визначенням ключових компонентів програмної платформи, в результаті маємо такий список компонентів:

- WhisperX та скрипт для його запуску;
- API, яке забезпечує обмін даних та аудіо файлів між клієнтом та платформою;
- NodeJS сервіс (частина API) який зв’язує API та WhisperX. Також, за допомогою нього відбувається запис даних у БД;
- Файлове сховище, на якому не тільки аудіо файли, а й результати розпізнавання мовлення у текстовому форматі;
- База даних MongoDB;
- Кеш Redis.

Основну схему програмної платформи, на якій зображені всі ключові компоненти платформи зображено на рис. 3.1.

3.3 Підготовка програмно-апаратного середовища для роботи WhisperX

Перш за все, для роботи WhisperX необхідно підготувати відповідне програмно-апаратне середовище [16]. Було підготовлено наступне програмно-апаратне середовище, яке відповідає мінімальним вимогам і має бути оптимальним як для розробки, так і для подальшого використання розробки:

- Графічна карта NVIDIA RTX 3070, з пам’яттю, що відповідає мінімальним вимогам, у розмірі 8 Гб. Для використання GPU, встановлено бібліотеки NVIDIA, зокрема cuBLAS 11.8 і cuDNN 8.9.7;
- Процесор Intel Core i7-8700 разом з оперативною пам’яттю об’ємом 32 Гб, що дозволить, окрім WhisperX, запустити інші компоненти платформи на цій же машині;
- Python 3.10 та PyTorch 2.0.0 – саме на цих версіях Python та PyTorch буде гарантовано працювати WhisperX. Запуск та робота Python відбувається через віртуальне середовище за допомогою Conda [17].

Для розпізнавання спікерів (діаризація) було додатково встановлено бібліотеку pyannote.audio для виявлення голосової активності (VAD) і діаризації мовлення, яка також призначена і для інших завдань обробки аудіо [18].

3.4 Офлайн діаризація

За замовчуванням діаризація у WhisperX потребує доступ до Інтернету, і для цього ще й потрібно отримати доступ за токеном на платформі Hugging Face. Хоч це й безкоштовно, але за різними причинами деякі підприємства не хочуть бути залежними від зовнішніх факторів з Інтернету, а також не хочуть аби їх дані потрапляли до третіх рук у мережі Інтернет, тому слід подбати про те, щоб розпізнавання спікерів відбувалося у офлайн режимі, – тобто без використання Інтернету.

Для цього необхідно завантажити моделі необхідні для pipeline діаризації з офіційного репозиторію pyannote.audio на платформі Hugging Face. Pipeline – це комбінація кількох моделей або оброблювальних етапів, що об'єднані в послідовний процес для автоматизації завдання. У нашому випадку, pipeline для діаризації в pyannote.audio складається з кількох моделей, які працюють разом. Pipeline спочатку використовує модель для VAD, потім – для сегментації спікерів та подальшого маркування кожного фрагменту відповідним спікером. Детально описано у інструкції [20].

Таким чином у нас є директорія models, у якій дві моделі: «pyannote_embedding.bin» та «pyannote_model_segmentation-3.0.bin». А також файл конфігурації необхідний для pipline «pyannote_diarization_config.yaml» з конфігурацією зображеною на рис.3.2.

Така конструкція дозволяє використовувати WhisperX повністю у офлайн режимі без використання доступу в Інтернет.

```yaml
version: 3.1.0

pipeline:
  name: pyannote.audio.pipelines.SpeakerDiarization
  params:
    clustering: AgglomerativeClustering
    embedding: models/pyannote_embedding.bin
    embedding_batch_size: 32
    embedding_exclude_overlap: true
    segmentation: models/pyannote_model_segmentation-3.0.bin
    segmentation_batch_size: 32

params:
  clustering:
    method: centroid
    min_cluster_size: 12
    threshold: 0.7045654963945799
  segmentation:
    min_duration_off: 0.0
```

Рис. 3.2. Конфігурація «pyannote_diarization_config.yaml»,
що необхідна для pipline

3.5 Інтерфейс запуску для WhisperX – Transcriber

Для зручності та більш зрозумілої структури компоненту whisperx-transcriber розробленої платформи розділимо на декілька частин. Вхідною точкою буде скрипт main.py, який отримує вхідні дані та запускає весь процес розпізнавання мовлення. Цей скрипт імпортуватиме інші модулі, які розмістимо у директорії «myFunctions»:

- audio_utils.py – скрипт, у якому знаходяться функції в основному для зчитування аудіо файлів;

- diarization_utils.py – скрипт, у якому знаходяться функції для виконання процесу розпізнавання спікерів;

- transcriber.py – скрипт, у якому знаходяться функції для процесу розпізнавання мовлення.

Загалом у компоненті whisperx-transcriber отримано таку структуру, яку зображено на рис. 3.3.

```
whisperx-transcriber /
|- models /
|-|- pyannote_embedding.bin
|-|- pyannote_model_segmentation-3.0.bin
|-myFunctions /
|-|- audio_utils.py
|-|- diarization_utils.py
|-|- transcriber.py
|-pyannote_diarization_config.yaml
|-main.py
```

Рис. 3.3 Структура WhisperX-Transcriber

3.5.1 Огляд «transcriber.py»

Головною функцією у «transcriber.py» є функція transcriber, яка отримує вхідні дані для обробки. Спершу скрипт здійснює сегментацію та діаризацію аудіофайлу, застосовуючи функцію diarize з модуля «diarization_utils.py». Функція diarize повертає масив сегментів, де кожен сегмент є об’єктом, що містить часову мітку початку (timecode_in), часову мітку кінця (timecode_out) і номер спікера (speaker). Таким чином, для кожного спікера в аудіо можна виділити окремий сегмент для подальшого розпізнавання мовлення.

Після діаризації скрипт завантажує модель розпізнавання WhisperX, вказану як large-v3, із заданою конфігурацією compute_type. Ця конфігурація дозволяє ефективно використовувати апаратні ресурси, обираючи між точністю та швидкодією. Модель завантажується з локального каталогу models, що оптимізує процес розпізнавання для великих або тривалих аудіофайлів, де необхідна стабільність та гнучкість у налаштуванні.

Далі скрипт застосовує функцію load_audio_segment, передаючи часові мітки сегмента, щоб завантажити саме той фрагмент аудіо, де говорить відповідний спікер. Після отримання тексту, транскрибованого з аудіо сегмента, результати зберігаються у форматі WebVTT. Це дозволяє відображати результати як субтитри, забезпечуючи синхронізацію тексту з аудіо та зручне відтворення у сумісних медіа плеєрах.

Скрипт також включає обробку винятків через try-except, щоб уникнути збоїв у роботі під час тривалого процесу обробки аудіофайлу. Такий підхід забезпечує безперервну роботу навіть у випадках, коли виникають проблеми з окремими сегментами або файлом.

$$progress = progressDiarized + transcribeIteration \frac{100 - progressDiarized}{len(diarizeArray)} \quad (3.1)$$

де $progressDiarized$ – загальний показник прогресу виконання включно з діаризацією у відсотках, якого ми умовно позначили, що він дорівнює 15%;

$transcribeIteration$ – ітерація розпізнавання, що відповідає порядковому номеру сегмента, з якого розпізнається мовлення;

$len(diarizeArray)$ – розмір масиву сегментів, отриманий з функції diarize.

Після обробки всіх сегментів результати зберігаються у вихідний файл у форматі WebVTT.

3.5.2 Огляд «diarization_utils.py»

Скрипт diarization_utils.py створено для обробки аудіофайлів із застосуванням функцій діаризації, реалізованих у бібліотеці pyannote.audio. Він виконує низку завдань, таких як:

- обробка аудіо форматів, запуск діаризації за допомогою попередньо навчених офлайн моделей pipeline,

- обробка результатів та перетворення їх у зручний формат для подальшого використання.

Розглянемо ключові функції цього скрипту. Основними компонентами diarization_utils.py є:

- *Імпорт необхідних бібліотек:* Скрипт починається з імпорту бібліотек torch, pandas, pydub, Pipeline з pyannote.audio, а також декількох допоміжних бібліотек

(Path і re). Ці бібліотеки забезпечують основну функціональність скрипта: зокрема, pandas використовується для обробки даних, а pydub — для конвертації аудіофайлів у формат WAV.

- *Функція parseTimestring:* Ця функція відповідає за розпізнавання та обробку тимчасових міток у текстовому форматі [00:00:00.000 --> 00:00:00.000]. Вона використовує регулярні вирази для вилучення початкових та кінцевих часових міток, які пізніше використовуються у звітах про діаризацію.

- *Функція diarize_df_to_array:* diarize_df_to_array перетворює DataFrame з результатами діаризації на масив словників, де кожен словник представляє окремий сегмент аудіо. У цьому масиві зберігається інформація про початковий і кінцевий час сегменту, а також індекс спікера, що дозволяє зручно працювати з результатами діаризації.

- *Функція convert_to_wav:* Оскільки моделі pyannote.audio потребують формат WAV з певними характеристиками (16 кГц, моно), функція convert_to_wav конвертує аудіофайли у цей формат. Це гарантує, що будь-який аудіофайл, незалежно від початкового формату, може бути оброблений системою.

- *Функція diarize:* Основна функція diarize виконує основну обробку аудіофайлів:

1) завантажує попередньо навчену модель для діаризації за шляхом, вказаним у diarization_config_path.

2) Після завантаження моделі (pipeline) аудіофайл обробляється, і якщо він не у форматі WAV, виконується конвертація

3) Використовуючи клас ProgressHook, функція отримує результати діаризації та формує DataFrame, що містить часові сегменти з визначеними спікерами

4) Далі функція використовує diarize_df_to_array, щоб перетворити результати в масив, придатний для подальшого використання.

- *Вихідні дані:* Функція diarize повертає два результати: DataFrame з інформацією про кожен сегмент та масив словників, де кожен запис містить

детальну інформацію про часові межі та спікера. Це забезпечує зручність роботи з результатами діаризації, дозволяючи подальшу обробку та інтеграцію в платформу.

Скрипт diarization_utils.py є ключовим інструментом для запуску діаризації та обробки її результатів у платформі. Завдяки чіткому розподілу функцій на окремі етапи, скрипт забезпечує ефективну обробку аудіофайлів, формує структуровані дані про спікерів та спрощує інтеграцію процесу діаризації в загальну архітектуру платформи розпізнавання мовлення.

3.5.3 Огляд «main.py»

Скрипт main.py виконує роль основної точки запуску процесу розпізнавання мовлення та діаризації за допомогою інструмента whisperx-transcriber. Він приймає вхідні параметри, такі як тип обчислень (compute_type), шлях до вхідного аудіофайлу (input_file), файл із субтитрами (captions_file), а також мінімальну, максимальну та точну кількість спікерів для діаризації, після чого передає ці параметри до основної функції розпізнавання transcriber.

Використовуючи бібліотеку argparse, скрипт забезпечує зручне введення параметрів через командний рядок. Це дає змогу користувачеві легко налаштовувати процес обробки, контролюючи ключові параметри без необхідності модифікувати код. Ключові аргументи включають:

- compute_type – вказує тип обчислень, наприклад, float32, float16 або інші режими, що впливають на продуктивність і точність обробки.

- input_file та captions_file – шляхи до аудіофайлу для обробки та до файлу для збереження текстових результатів, що забезпечує зручність керування вхідними й вихідними даними.

- min_speakers, max_speakers та num_speakers – визначають кількість спікерів, що дозволяє адаптувати діаризацію до відомої кількості учасників.

Отримані параметри передаються в transcriber, яка виконує процес діаризації та розпізнавання мовлення, інтегруючи їх у загальний процес whisperx-transcriber.

Після обробки main.py виводить результати в консоль, що дозволяє відстежувати прогрес і отримати миттєвий доступ до кінцевих результатів роботи платформи.

Такий підхід організовує скрипт як централізований інтерфейс для ініціювання процесу розпізнавання мовлення й діаризації, забезпечуючи простоту у використанні й налаштуванні платформи на основі WhisperX.

Скрипт main.py відображено на рис. 3.4.

```
import time
import argparse
from myFunctions.transcriber import transcriber
from pathlib import Path

def main():
  parser = argparse.ArgumentParser()
  parser.add_argument('-compute_type', type=str, help='compute type')
  parser.add_argument('-input_file', type=str, help='input file')
  parser.add_argument('-captions_file', type=str, help='captions file')
  parser.add_argument('-min_speakers', default=None, type=int, help='min_speakers')
  parser.add_argument('-max_speakers', default=None, type=int, help='max_speakers')
  parser.add_argument('-num_speakers', default=None, type=int, help='num_speakers')
  parser.add_argument('-languages', nargs='+', default=None, type=str, help='num_speakers')

  args = parser.parse_args()

  print("Value for -compute_type:", args.compute_type)
  print("Value for -input_file:", args.input_file)
  print("Value for -captions_file:", args.captions_file)
  print("Value for -min_speakers:", args.min_speakers)
  print("Value for -max_speakers:", args.max_speakers)
  print("Value for -num_speakers:", args.num_speakers)
  print("Value for -languages:", args.languages)

  transcriber(
    compute_type=args.compute_type,
    audio_file=Path(args.input_file),
    captions_file=Path(args.captions_file),
    min_speakers=args.min_speakers,
    max_speakers=args.max_speakers,
    num_speakers=args.num_speakers,
    language=None
  )

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

Рис. 3.4. Скрипт «main.py»

3.6 API

Розробка частини платформи API відбувається на NodeJS за допомогою бібліотеки ExpressJS. Це дозволяє ефективно обробляти HTTP-запити від клієнтів

та передавати їх для обробки у WhisperX-Transcriber. API складається з таких основних частин, які відповідають MVC архітектурі [23]:

- *маршрутизація (routes)*, яка визначає шляхи, які відповідають архітектурному стилю REST [22], за якими обробляються HTTP-запити, які надходять до API. Основні маршрути наведені на рис. 3.5;

```
this.router.post('/', this.transcribe);
this.router.get('/', this.getAllTasks);
this.router.get('/:taskId', this.getTaskProcessing);
this.router.patch('/:taskId/cancel', this.cancelTask);
this.router.delete('/:taskId', this.removeTask);
```

Рис. 3.5. Основні маршрути для обробки HTTP-запитів для роботи з процесами розпізнавання мовлення

- *контролери (controllers)* – функції або методи, які обробляють HTTP-запити для певних маршрутів. Саме у них відбувається основна логіка роботи всіх процесів, що стосуються API. Найбільш важливим контролером є метод «transcribe» робота якого описана у блок-схемі, яку зображено на рис. 3.6. Дана блок-схема є спрощеною, тому що не враховує перевірки конфліктів та помилок;

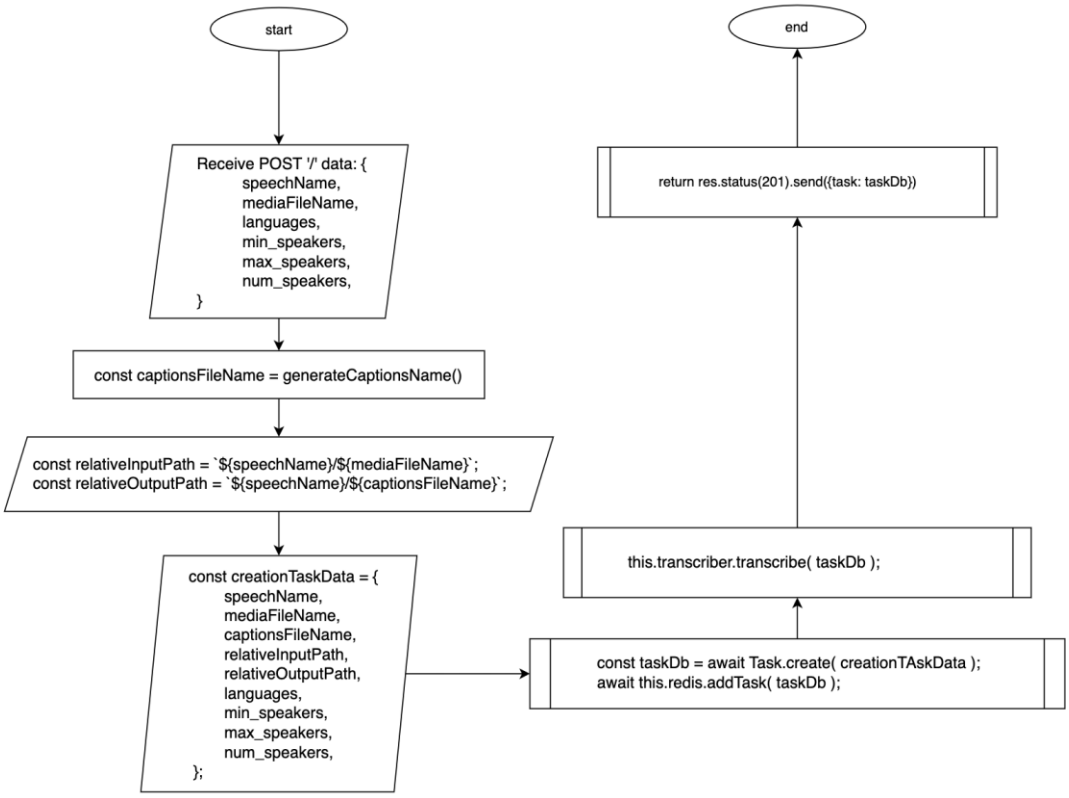


Рис. 3.6. Спрощена блок-схема контролера «transcribe»

- *моделі (models)*, які визначають структуру, правила та поведінку даних, які зберігаються в базі даних. Основною моделлю є модель Task, що побудована за допомогою бібліотеки об'єктно-документного моделювання mongoose. Task, яка відображена на рис. 3.7, описує як відбуваються запис, та редагування завдань у БД.

```
const mongoose = require('mongoose');
const openAILanguagesTable = require('../core/OpenAILanguagesTable.js');
const validLanguages = openAILanguagesTable.getAllList().map(({code}) => code);

const taskSchema = new mongoose.Schema({
  speechName: { type: String, match: /^[a-zA-Z0-9\s_\-\.]{1,64}$/, required: true, },
  mediaFileName: { type: String, required: true, },
  captionsFileName: { type: String, required: true, },
  relativeInputPath: { type: String, required: true, },
  relativeOutputPath: { type: String, required: true, },
  languages: [{
    type: String, enum: validLanguages,
    message: 'Language {VALUE} is not supported. It can be a one of these: ' + validLanguages.join(', '),
  }],
  min_speakers: {
    type: Number,
    validate: { validator: Number.isInteger, message: '{VALUE} is not an integer value', },
  },
  max_speakers: {
    type: Number,
    validate: { validator: Number.isInteger, message: '{VALUE} is not an integer value', },
  },
  num_speakers: {
    type: Number, required: false,
    validate: { validator: Number.isInteger, message: '{VALUE} is not an integer value', },
  },
  progress: { type: Number, min: 0, max: 100, default: 0, },
  status: {
    type: String, default: 'CREATED',
    enum: ['CREATED', 'WAIT_WORKER', 'PROCESS', 'COMPLETED', 'ERROR', 'CANCELLED'],
    message: '{VALUE} is not supported. It can be a one of these: "CREATED", "WAIT_WORKER", "PROCESS", "COMPLETED", "ERROR", "CANCELLED"',
  },
  description: { type: String, match: /^.{0,1024}$/, default: "", },
  createdAt: { type: Date, default: Date.now, },
  updatedAt: { type: Date, default: Date.now, },
});

const preSave = async function() {
  this.set({ updatedAt: Date.now() });
};
const preFindOneAndUpdate = async function() {
  this.set({ updatedAt: Date.now() });
};
taskSchema.pre('save', preSave);
taskSchema.pre('findOneAndUpdate', preFindOneAndUpdate);

const Task = mongoose.model('Task', taskSchema);
module.exports = Task;
```

Рис. 3.7. Модель Task

3.7 Виклик та моніторинг процесу транскрибування

Як було наведено у попередньому пункті, у API найбільш важливим контролером був метод «transcribe». У даному методі після того, як створився запис у БД та у кеші Redis, відбувається виклик методу «transcribe» об’єкта transcriber.

Метод «transcribe» за допомогою методу spawn вбудованого у NodeJS модуля child_process генерує на основі вхідних даних та запускає WhisperX-Transcriber (його точку входу main.py) у вигляді консольної команди, що зображено на рис. 3.8:

```
const args = [
      'main.py',
      '-compute_type', `${computeType}`,
      '-input_file', `${input_file}`,
      '-captions_file', `${captions_file}`,
      typeof min_speakers !== 'undefined' && Number.isInteger(min_speakers) === true ? ['-min_speakers', min_speakers] : null,
      typeof max_speakers !== 'undefined' && Number.isInteger(max_speakers) === true ? ['-max_speakers', max_speakers] : null,
      typeof num_speakers !== 'undefined' && Number.isInteger(num_speakers) === true ? ['-num_speakers', num_speakers] : null,
      Array.isArray(languages) === true && languages.length > 0 ? ['-languages', ...languages] : null,
].filter(el => el !== null).flat();

const transcribing = spawn('python', args, {
      cwd: transcriberDir,
      detached: false,
});
```

Рис. 3.8. Генерування команди та запуск main.py

Результатами виклику метода «transcribe» є: об’єкт transcribing, який містить методи для процесу spawn; функція cancel, що дозволяє зупинити процес розпізнавання мовлення; об’єкт myEmitter, який повідомляє про різні події, що сталися під час процесу розпізнавання, та дані цих подій. Так як main.py виводить результати у консоль, а метод «transcribe» об’єкта transcriber запускає при цьому main.py як звичайну консольну команду, то у об’єкт transcribing має метод для зчитування консольного виводу stdout. Дані отримані із stdout розбираються функціями: parseStdout, isProgress та parcePercent, які відображені на рис. 3.9.

```
const parsePercent = (inputString = '') => {
  const regex = /(\d+)%/;
  const match = inputString.match(regex);
  if (match) {
    const percent = match[0];
    return percent;
  } else {
    return null;
  }
};
const isProgress = (inputString = '') => {
  const regex = /PROGRESS.*?(\d+)%/;
  const match = inputString.match(regex);
  return match;
};
const parseStdout = (data, myEmitter) => {
  const lines = String(data).split('\n').map(line => line.trim());
  lines.forEach(line => {
    if (isProgress(line) !== null) {
      const percent = parsePercent(line);
      if (percent !== null) {
        myEmitter.emit('progress', {percent});
      }
    };
  })
};
transcribing.stdout.on('data', (data) => {
  parseStdout(data, myEmitter);
});
```

Рис. 3.9. Розбір даних отриманих із stdout об’єкта transcribing

Об’єкт myEmitter, отриманий в результаті виклику метода transcribe, повідомляє про наступні події:

- cancelled – відбувається тоді, коли процес transcribing був відмінений за допомогою функції cancel, отримана в результаті виклику метода transcribe;

- completed – відбувається тоді, коли процес transcribing був успішно завершений з нульовим кодом;

- not_completed – відбувається тоді, коли процес transcribing був завершений не з нульовим кодом або з критичними помилками;

- progress – відбувається тоді, коли змінився показник прогресу у відсотках від 0 до 100.

Дані подій, отриманих із об’єкта myEmitter записуються у кеш та у БД. Кеш Redis налаштований таким чином, що крім кешування, ще й виконує опцію публікації оновлень записів підписникам, що підписалися на певні записи. Коли клієнт посилає HTTP-запит з методом GET за шляхом «/:taskId», тоді контролер, який обробляє даний маршрут підписується за допомогою отриманого taskId на завдання Task, яке має id відповідне до taskId. Тоді результатом обробки даного HTTP-запиту буде потокова відповідь, з даними виконання процесу розпізнавання мовлення у реальному часі за допомогою HTTP-заголовку «Content-Type», який дорівнює значенню «text/event-stream; charset=utf-8».

Для того, щоб відмінити виконання завдання Task, клієнт посилає HTTP-запит, який містить taskId з HTTP-методом PATCH за шляхом «/:taskId/cancel». Тоді контролер за допомогою отриманого taskId викликає функцію cancel того завдання Task, id якого відповідає отриманому taskId. У результаті клієнт отримує відповідь з оновленнями запису Task із БД за taskId, наданого від клієнта.

3.8 Апробація та результати розробки

Для того, щоб скористатися розробкою, необхідно передусім мати аудіо файл, з якого необхідно розпізнати мовлення у текстовому форматі, який має відповідати вимогам до WhisperX. Для цього був підготовлений такий аудіофайл, у якому

відбувається діалог українською мовою між двома спікерами, де один спікер з жіночим голосом, інший – з чоловічим. Підготовлений аудіо файл має наступні технічні характеристики:

- хронометраж: 57 хв 39 сек;
- кодек: PCM;
- частота: 16 кГц;
- формат: wav;
- канали: моно.

У якості клієнта для надсилання API-серверу HTTP-запити будемо використовувати Postman – одна з найкращих утиліт для тестування та автоматизації API, також це інструмент, який вміє автоматично генерувати документацію для API [24].

Для порівняльного аналізу продуктивності системи проведемо три тести:

1) тільки один файл;

2) два абсолютно однакові файли (дві копії одного файлу);

3) три абсолютно однакові файли (три копії одного файлу).

У тестах упродовж процесу розпізнавання на однакових проміжках часу будуть вимірюватися наступні показники:

- Завантаження процесора (CPU, %);
- Використання оперативної пам’яті (RAM, %);
- Завантаження графічної карти (GPU Utilization, %);
- Використання пам’яті графічної карти (GPU dedicated memory, %);
- Використання виділеної пам’яті графічної карти (GPU shared memory, %);
- Температура графічної карти (GPU Temperature, °C).

Перший тест зайняв 8 хв 7 сек часу і впродовж тесту на однакових проміжках часу були зняті показники, наведені у таблиці 3.1. Враховуючи, що хронометраж аудіо файлу має 57 хв 39 сек, то процес розпізнавання включно з діаризацією спікерів виконався зі швидкістю у 7,1 разів швидше ніж хронометраж.

Таблиця 3.1

Результати вимірювань першого тесту з розпізнавання мовлення з одного аудіо файлу за допомогою розробленої програмної платформи

Time	**CPU %**	**RAM %**	**GPU Utilization %**	**GPU dedicated memory %**	**GPU shared memory %**	**GPU temperature °C**
00:00:00	3,0	39,2	4,0	12,8	0,6	37,0
00:00:41	57,0	42,6	2,0	24,3	1,3	41,0
00:01:21	53,0	42,6	2,0	24,3	1,3	42,0
00:02:02	40,0	42,9	2,0	66,4	0,6	53,0
00:02:42	28,0	42,9	2,0	67,7	0,6	55,0
00:03:23	34,0	43,9	2,0	62,6	0,6	55,0
00:04:04	29,0	43,3	4,0	63,9	1,3	56,0
00:04:44	33,0	42,6	3,0	63,9	2,5	55,0
00:05:25	36,0	42,3	2,0	63,9	1,3	56,0
00:06:05	25,0	42,3	3,0	67,7	1,3	56,0
00:06:46	32,0	42,3	2,0	67,7	0,6	55,0
00:07:26	52,0	42,0	4,0	83,1	2,5	56,0
00:08:07	5,0	38,6	4,0	10,2	0,6	48,0

Другий тест з одночасним розпізнаванням двох абсолютно однакових аудіо файлів зайняв 39 хв 28 сек. Обидва файли одночасно почали розпізнаватися і одночасно завершилися. Враховуючи, що хронометраж кожного аудіо файлу (копії одного і того ж самого файлу під різними назвами) має 57 хв 39 сек, то відношення часу витраченого на розпізнавання до хронометражу кожного файлу сягає 1.47, а відношення до загального хронометражу (сума хронометражу файлів) сягає позначці 2.93 – це трохи швидше за хронометраж файлів. Це говорить про те, що швидше було б по черзі розпізнати кожен файл окремо, ніж одночасне їх розпізнавання. Показники другого тесту з одночасним розпізнаванням мовлення двох ідентичних файлів наведено у таблиці 3.2.

Таблиця 3.2

Результати вимірювань другого тесту
з одночасним розпізнаванням мовлення двох ідентичних аудіо файлів
за допомогою розробленої програмної платформи

Time	**CPU %**	**RAM %**	**GPU Utilization %**	**GPU dedicated memory %**	**GPU shared memory %**	**GPU temp °C**
00:00:00	4	32,6	3	10,2	0,6	33
00:02:19	100	39,2	2	33,2	1,3	36
00:04:39	99	39,2	2	33,2	1,3	34
00:06:58	37	50,5	20	98,4	1,9	40
00:09:17	41	62,7	42	98,4	6,9	41
00:11:36	49	60,5	31	99,7	8,8	43
00:13:56	43	53,0	22	99,7	5,0	40
00:16:15	40	59,6	25	98,4	10,0	44
00:18:34	48	62,1	35	99,7	9,4	42
00:20:54	35	57,7	25	99,7	11,3	46
00:23:13	51	55,5	28	97,1	8,1	44
00:25:32	40	54,5	25	99,7	10,0	43
00:27:52	51	49,5	13	99,7	3,1	45
00:30:11	61	54,9	14	98,4	4,4	43
00:32:30	43	53,6	18	99,7	1,3	46
00:34:49	40	62,4	30	99,7	16,9	44
00:37:09	38	58,9	31	98,4	20,6	43
00:39:28	4	32,6	3	5,1	0,6	39

Третій тест з одночасним розпізнаванням трьох абсолютно однакових аудіо файлів зайняв 3 год 16 хв 26 сек. Усі три файли одночасно почали розпізнаватися і одночасно завершилися. Відношення часу витраченого на розпізнавання до хронометражу кожного файлу сягає 0.30, а відношення до загального хронометражу (сума хронометражу файлів) сягає позначці 0.89 – це навіть менше ніж хронометраж файлів. Тому краще на даній програмно-апаратній збірці не розпізнавати мовлення одночасно з трьох файлів. Показники третього тесту з одночасним розпізнаванням мовлення трьох ідентичних файлів наведено у таблиці 3.3.

Таблиця 3.3

Результати вимірювань третього тесту
з одночасним розпізнаванням мовлення трьох ідентичних аудіо файлів
за допомогою розробленої програмної платформи

Time	CPU %	RAM %	GPU Utilization %	GPU dedicated memory %	GPU shared memory %	GPU temp °C
00:00:00	4	32,9	3	8,9	0,6	33
00:11:33	100	41,4	8	44,7	1,9	42
00:23:07	100	41,7	2	44,7	1,9	39
00:34:40	65	74,6	83	99,7	21,9	45
00:46:13	67	75,2	97	98,4	23,8	52
00:57:46	55	75,2	92	99,7	22,5	57
01:09:20	69	74,9	100	99,7	23,8	51
01:20:53	70	74,3	97	98,4	23,1	47
01:32:26	72	84,3	93	99,7	45,0	43
01:44:00	58	81,5	89	99,7	38,8	41
01:55:33	62	76,5	80	99,7	30,0	41
02:07:06	81	76,2	91	99,7	32,5	42
02:18:40	63	76,8	89	99,7	33,8	42
02:30:13	64	45,5	5	94,6	18,8	43
02:41:46	30	40,1	4	97,1	8,8	44
02:53:19	49	40,8	2	60,1	0,6	45
03:04:53	20	36,4	2	66,4	0,6	47
03:16:26	5	31,3	4	5,1	0,6	40

Для того, щоб зрозуміти чому зі збільшенням одночасної кількості аудіо файлів так знижається швидкість розпізнавання мовлення, результати з таблиць 3.1–3.2 зображено у вигляді графіків на рисунках 3.10–3.12 відповідно.

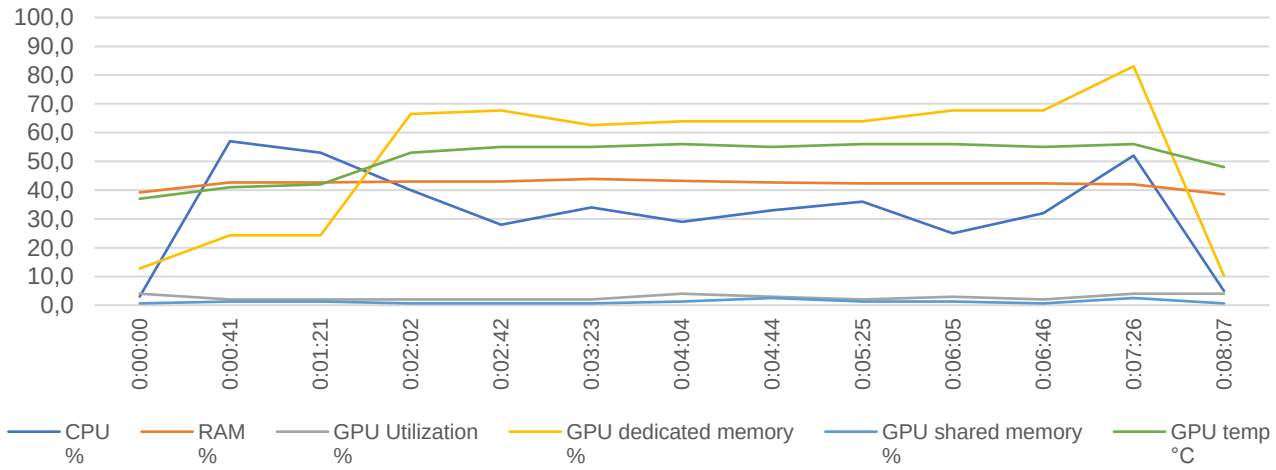


Рис. 3.10. Графік показників вимірювання першого тесту

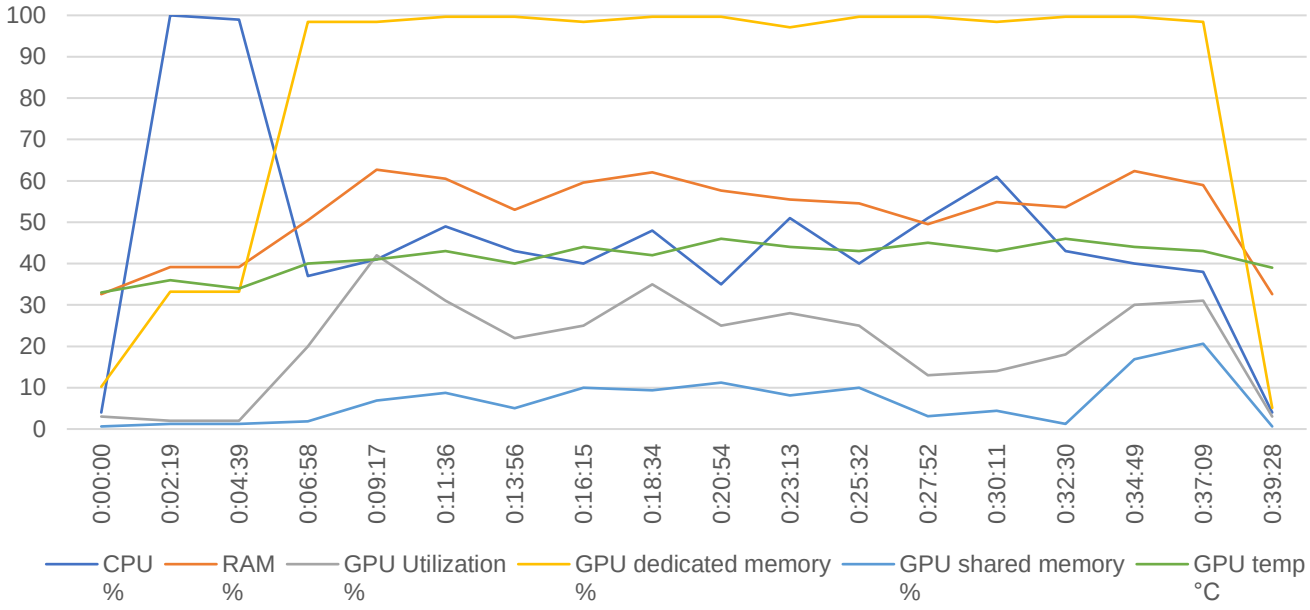


Рис. 3.11. Графік показників вимірювання другого тесту

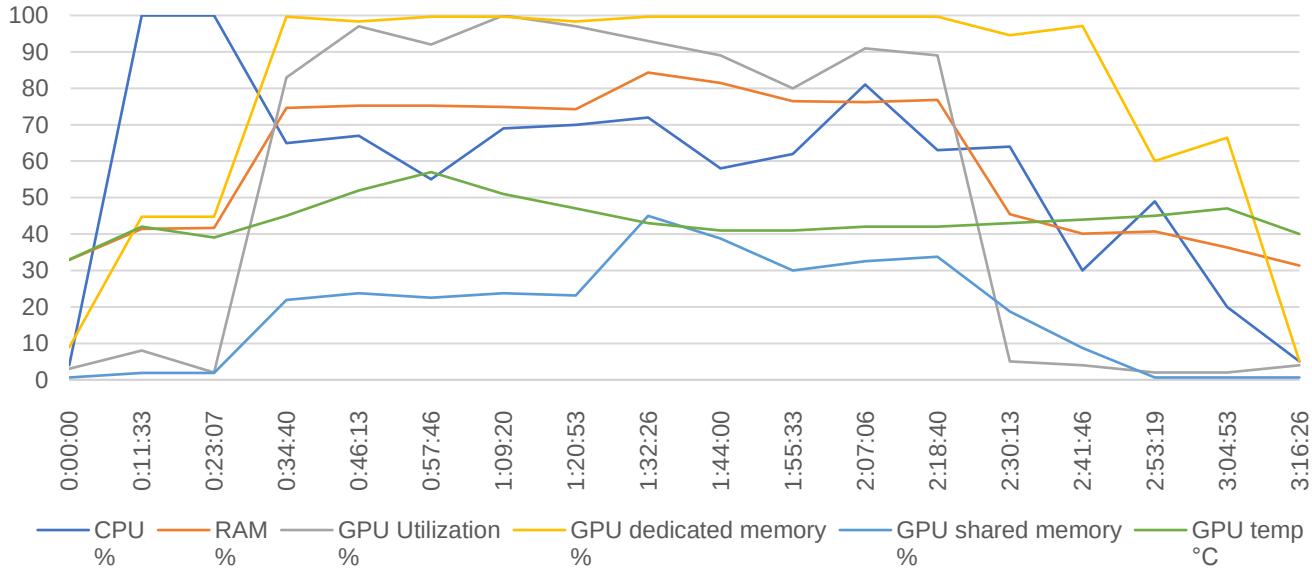


Рис. 3.12. Графік показників вимірювання третього тесту

На графіках видно, що об’єм пам’яті графічної карти, який у сягає 8 Гб, так як і в технічних вимогах, наведених в документації [16], є мінімальним. Під час розпізнавання двох і більше файлів вся пам’ять, яка фізично інтегрована у графічну карту, тобто, спеціально виділена для її використання (GPU dedicated memory) використовувалася максимально і її не вистачало і для цього використовувалася виділена пам’ять з оперативної пам’яті (GPU shared memory). І ми бачимо, що чим

більше файлів одночасно оброблюються, тим більше оперативної пам’яті (RAM) для обробки у графічній карті виділяється. Об’єму оперативної пам’яті обсягом у 32 Гб якраз достатньо для її виділення у графічну пам’ять, тому можна сказати, що цей обсяг оперативної пам’яті – це теж є мінімальним об’ємом для розпізнавання трьох файлів. При цьому завантаження графічного процесора (GPU utilization) теж збільшується при збільшенні обробки одночасних аудіо файлів, а також при обробці трьох аудіо файлів графічний процесор був завантажений майже до максимальної поділки. Завантаження звичайного процесора (CPU) теж збільшується але, навіть при обробці трьох файлів, ще є запас процесора.

Використання оперативної пам'яті як «спільної» графічної пам'яті (GPU shared memory) дійсно може підвищити навантаження на процесор. Це зумовлено тим, що для доступу до оперативної пам'яті (RAM) CPU часто бере участь у контролі потоків даних між GPU і RAM, особливо якщо інтегрована графіка використовує системну пам'ять замість виділеної графічної пам'яті. Такий механізм інтенсивно використовує шину процесора, що спричиняє більші затримки доступу до даних і, відповідно, зростання загального завантаження CPU. Це особливо помітно при великих обсягах обробки [25].

Дослідження також показують, що у випадках, коли GPU «позичає» пам'ять з оперативної, CPU може зазнавати затримок, що збільшує загальний час обробки, особливо при високих графічних навантаженнях [26]. Це обмежує продуктивність та створює потенційні "вузькі місця" у продуктивності CPU, коли він зайнятий управлінням пам'яттю для графічних обчислень. Додатково, більша кількість запитів до оперативної пам'яті з боку GPU збільшує використання контролера пам’яті CPU, що впливає на його роботу в цілому і може знижувати ефективність роботи інших процесів

3.9 Висновки до розділу

У даному розділі була визначена архітектура програмної платформи, яка складається з таких ключових компонентів, як:

- API, за допомогою якого здійснюється обмін даними з клієнтами через HTTP-запити;

- БД MongoDB, у якій зберігаються завдання (Tasks), створені на основі вхідних даних, отриманих від клієнтів;

- Redis, як кеш бази даних і одночасно інструмент повідомлень про оновлення прогресу виконання завдання для його моніторингу;

- безпосередньо сам репозиторій WhisperX, навколо якого для взаємодії з API був створений консольний інтерфейс, який отримує вхідні параметри, та запускає процес розпізнавання у WhisperX. Його основними компонентами є скрипти:

а) «main.py», який отримує вхідні параметри, передає їх як аргументи для розпізнавання мовлення, з якого отримує результати і виводить їх у консоль, звідки їх розбирає API за допомогою об’єкта «transcriber»;

б) «transcriber.py», який відповідає за розпізнавання мовлення із аудіо файлу у текстовому вигляді;

в) «diarization_utils.py», який відповідає за діаризацію спікерів за допомогою завантажених офлайн моделей та бібліотеки для обробки аудіо pyannote.audio.

Проведене тестування платформи для одночасного розпізнавання одного, двох та трьох ідентичних аудіо файлів, показало, що апаратна частина відповідає лише мінімальним вимогам для розпізнавання мовлення із аудіо файлів. Було визначено, що з такою апаратною збіркою, оптимальним є одночасне розпізнавання лише одного або максимум двох файлів. Три і більше файлів значно понижують швидкість розпізнавання.

Для збільшення швидкості розпізнавання мовлення рекомендується використання більш потужних графічних карт з більшим обсягом графічної пам’яті, ніж 8 Гб. Важливими є також і оперативна пам’ять, та процесор. Тому що оперативна пам’ять необхідна для продовження роботи графічної карти тоді, коли графічної пам’яті, яка вбудована у графічну карту, є недостатньо. Але оперативна пам’ять у ролі графічної пам’яті значно менш продуктивна, ніж сама графічна пам’ять. Також використання оперативної пам’яті як графічну збільшує навантаження і на процесор CPU через такі основні аспекти:

- *Додаткове навантаження на системну шину:* коли GPU звертається до shared memory, він фактично використовує оперативну пам’ять через спільну системну шину. Це може призвести до збільшення навантаження на шину, яку також використовує CPU для доступу до оперативної пам’яті. У результаті CPU може отримувати дані повільніше, що підвищує його завантаження при обробці інформації.

- *Координація доступу до пам’яті:* CPU може витрачати більше часу на координацію доступу до оперативної пам'яті, особливо якщо GPU інтенсивно використовує shared memory. Це може викликати затримки, оскільки обидва процесори (CPU і GPU) фактично конкурують за доступ до одного і того ж ресурсу – оперативної пам'яті.

- *Переривання та управління пам’яттю:* при використанні shared memory операційна система і CPU також виконують більше переривань та інших дій, пов'язаних з керуванням пам'яттю. Це додає навантаження на CPU, особливо при великих графічних навантаженнях.

Таким чином, активне використання GPU shared memory може призвести до непрямого збільшення завантаження CPU і зниження загальної продуктивності системи, оскільки і CPU, і GPU починають конкурувати за доступ до оперативної пам'яті. Тому пріоритетом є, звісно, обсяг графічної пам’яті, але також оперативна пам’ять та потужність процесора CPU теж є важливим фактором. Крім того, потужність графічного процесору теж потрібно враховувати, тому що при обробці трьох файлів одночасно, графічний процесор був майже повністю завантаженим.

Тому з наведених факторів, крім того, що необхідно збільшувати потужність апаратної частини, ще рекомендується змінити архітектуру розробленої програмної платформи таким чином, щоб вона була не монолітною, як зараз, а мікросервісною, що дозволить їй стати горизонтально масштабованою. Це збільшить складність системи як у розробці, так і в обслуговуванні, але й збільшить продуктивність системи [27].

ВИСНОВКИ

В результаті проведеного дослідження була розроблена програмна платформа для автоматичного розпізнавання мовлення на основі моделі WhisperX. Аналіз сучасних систем ASR підтвердив їх значення в різних сферах, зокрема в медичній, юридичній та бізнесовій діяльності, де швидка і точна обробка мовленнєвих даних стає критично важливою.

Перший розділ продемонстрував еволюцію технологій розпізнавання мовлення, підкресливши важливість сучасних моделей, таких як Transformer, які забезпечують високу точність завдяки контекстуальному аналізу.

У другому розділі було обрано оптимальну модель ASR, зокрема WhisperX, яка відповідала всім критеріям, визначеним для побудови масштабованої та ефективної платформи. Ця модель продемонструвала високу продуктивність і гнучкість у використанні, що робить її ідеальною для інтеграції в різні програмні рішення.

Третій розділ був присвячений архітектурі програмної платформи, що включає важливі компоненти, такі як API, MongoDB та Redis. Проведене тестування підтвердило, що для досягнення оптимальної продуктивності важливо враховувати апаратні вимоги системи. Важливим аспектом є рекомендація щодо переходу на мікросервісну архітектуру, що дозволить підвищити адаптивність і масштабованість платформи, незважаючи на поточну реалізацію на монолітній архітектурі.

Таким чином, розроблена платформа має потенціал для подальшого розвитку та вдосконалення. Перехід на мікросервісну архітектуру може стати важливим кроком для підвищення продуктивності та ефективності системи в умовах зростаючого попиту на автоматизацію обробки мовленнєвої інформації. Подальші дослідження можуть зосередитися на вдосконаленні алгоритмів розпізнавання, розширенні функціональності платформи та інтеграції з новими технологіями.

ПЕРЕЛІК ПОСИЛАНЬ

1. Touihri, Amani & Hamza, Salma & Amara, Wael. (2020). Speech Recognition for COVID-19 keywords Using Machine Learning. International journal of scientific research in computer science and engineering. Vol.8, Issue.4, pp.51-57, August (2020)

2. C. Zhang and P. C. Woodland, DNN speaker adaptation using parameterized sigmoid and ReLU hidden activation functions. 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Shanghai, China, 2016, pp. 5300-5304, doi: 10.1109/ICASSP.2016.7472689.

3. Salem, Fathi. (2022). Recurrent Neural Networks (RNN). 10.1007/978-3-030-89929-5_3.

4. Li, Xi-Lin. (2016). Recurrent neural network training with preconditioned stochastic gradient descent. 10.48550/arXiv.1606.04449.

5. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need” in Advances in neural information processing systems, 2017, pp. 5998–6008.

6. Mikolov, Tomas & Chen, Kai & Corrado, G.s & Dean, Jeffrey. (2013). Efficient Estimation of Word Representations in Vector Space. Proceedings of Workshop at ICLR. 2013.

7. A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, I. Sutskever. (2022). Robust Speech Recognition via Large-Scale Weak Supervision. 10.48550/arXiv.2212.04356.

8. A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, I. Sutskever. (2022) Introducing Whisper. Електронний ресурс: https://openai.com/index/whisper/

9. Baevski, A., Zhou, H., Mohamed, A., and Auli, M. Wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. Art. no. arXiv:2006.11477, 2020. doi:10.48550/arXiv.2006.11477.

10. Репозиторій openai/whisper: https://github.com/openai/whisper

11. Репозиторій Wav2Vec2: https://huggingface.co/docs/transformers/model_doc/wav2vec2

12. OpenAI Platform. Docs. Speech to text. Supported languages: https://platform.openai.com/docs/guides/speech-to-text/supported-languages

13. A. Seagraves. Benchmarking Top Open Source Speech Recognition Models: Whisper, Facebook wav2vec2, and Kaldi. DeepGram Article·AI & Engineering·Dec 19, 2022. Електронний ресурс: https://deepgram.com/learn/benchmarking-top-open-source-speech-models

14. M. Bain, J. Huh, T. Han, A. Zisserman. WhisperX: Time-Accurate Speech Transcription of Long-Form Audio. Visual Geometry Group, University of Oxford. arXiv:2303.00747v2 [cs.SD] 11 Jul 2023

15. Репозиторій m-bain/whisperX: https://github.com/m-bain/whisperX

16. Документація репозиторія m-bain/whisperX у електронному форматі. Електронний ресурс: https://github.com/m-bain/whisperX/blob/main/README.md

17. Hunt, John. (2023). Pip and Conda Virtual Environments. 10.1007/978-3-031-40336-1_57.

18. Bredin, Hervé. (2023). pyannote.audio 2.1 speaker diarization pipeline: principle, benchmark, and recipe. 1983-1987. 10.21437/Interspeech.2023-105.

19. Прикладний програмний інтерфейс. Електронний ресурс: https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B8%D0%B9_%D1%96%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81

20. Pyannote-audio tutorials applying_a_pipeline. Offline use. Електронний ресурс: https://github.com/pyannote/pyannote-audio/blob/develop/tutorials/applying_a_pipeline.ipynb

21. Desai, Krutika & Fiaidhi, Jinan. (2022). Developing a Social Platform using MERN Stack. 10.36227/techrxiv.21699764.v1.

22. Murge, Lokesh & Sampangi, Sandhya. (2024). REST AS A WEB ARCHITECTURAL DESIGN 1. International Journal of Engineering Technology and Management Sciences. 2. 461-464.

23. Necula, Sabina. (2024). Exploring The Model-View-Controller (MVC) Architecture: A Broad Analysis of Market and Technological Applications. 10.20944/preprints202404.1860.v1.

24. Kore, Pratik & Lohar, Mayuresh & Surve, Madhusudan & Jadhav, Snehal. (2022). API Testing Using Postman Tool. International Journal for Research in Applied Science and Engineering Technology. 10. 841-843. 10.22214/ijraset.2022.48030.

25. Mills, Richard Tran, "Dynamic adaptation to CPU and memory load in scientific applications" (2004). Dissertations, Theses, and Masters Projects. William & Mary. Paper 1539623457.

https://dx.doi.org/doi:10.21220/s2-my1g-nn08

26. T. Allen and R. Ge, "Characterizing Power and Performance of GPU Memory Access," 2016 4th International Workshop on Energy Efficient Supercomputing (E2SC), Salt Lake City, UT, USA, 2016, pp. 46-53, doi: 10.1109/E2SC.2016.012.

27. С. Немчинський. Про мікросервісну архітектуру. FoxmindEd. 26.03.2024. Електронний ресурс: https://foxminded.ua/mikroservisna-arkhitektura

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка програмної платформи для розпізнавання тексту на базі моделей машинного навчання»

Виконав: здобувач вищої освіти гр. ІСДМ-63
Богдан ВОВКОТРУБ

Керівник: Андрій БОНДАРЧУК
д.т.н., професор

Київ, 2024

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

Актуальність

Метою даної роботи є розробка програмної платформи для автоматичного розпізнавання мовлення на базі моделей машинного навчання, зокрема системи WhisperX, для забезпечення високої точності та швидкості перетворення аудіо в текст.

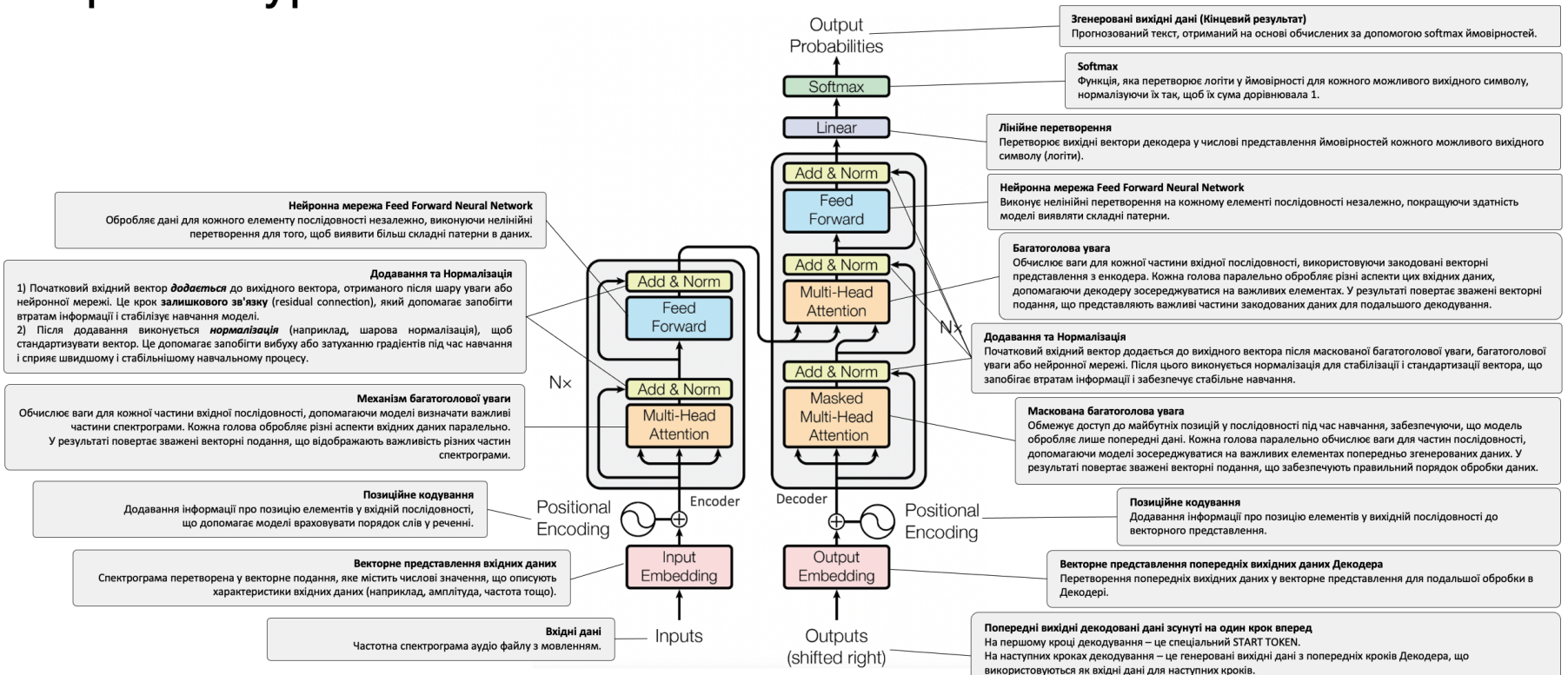
Об'єкт дослідження – технологія автоматичного розпізнавання мовлення.

Предмет дослідження – методологія та технічні рішення для автоматизації розпізнавання мовлення з аудіо файлів з використанням сучасних архітектур і програмних інтерфейсів.

Основні завдання:

- Дослідити сучасні архітектури технології автоматичного розпізнавання мовлення;
- Визначити критерії вибору, на основі яких провести вибірковий аналіз сучасних моделей розпізнавання мовлення на базі сучасних архітектур, які задовольняють потребам у різних галузях;
- На основі обраної моделі побудувати програмну платформу для автоматичного розпізнавання мовлення аудіо файлів, з можливістю її використання в різних галузях;
- Зробити висновки щодо ефективності розробленої платформи та надати рекомендації для її подальшого вдосконалення.

Архітектура TRANSFORMER



Згенеровані вихідні дані (Кінцевий результат)
Прогнозований текст, отриманий на основі обчислених за допомогою softmax ймовірностей.

Softmax
Функція, яка перетворює логіти у ймовірності для кожного можливого вихідного символу, нормалізуючи їх так, щоб їх сума дорівнювала 1.

Лінійне перетворення
Перетворює вихідні вектори декодера у числові представлення ймовірностей кожного можливого вихідного символу (логіти).

Нейронна мережа Feed Forward Neural Network
Виконує нелінійні перетворення на кожному елементі послідовності незалежно, покращуючи здатність моделі виявляти складні патерни.

Багатоголова увага
Обчислює ваги для кожної частини вхідної послідовності, використовуючи закодовані векторні представлення з енкодера. Кожна голова паралельно обробляє різні аспекти цих вхідних даних, допомагаючи декодеру зосереджуватися на важливих елементах. У результаті повертає зважені векторні подання, що представляють важливі частини закодованих даних для подальшого декодування.

Додавання та Нормалізація
Початковий вхідний вектор додається до вихідного вектора після маскованої багатоголової уваги, багатоголової уваги або нейронної мережі. Після цього виконується нормалізація для стабілізації і стандартизації вектора, що запобігає втратам інформації і забезпечує стабільне навчання.

Маскована багатоголова увага
Обмежує доступ до майбутніх позицій у послідовності під час навчання, забезпечуючи, що модель обробляє лише попередні дані. Кожна голова паралельно обчислює ваги для частин послідовності, допомагаючи моделі зосереджуватися на важливих елементах попередньо згенерованих даних. У результаті повертає зважені векторні подання, що забезпечують правильний порядок обробки даних.

Позиційне кодування
Додавання інформації про позицію елементів у вихідній послідовності до векторного представлення.

Векторне представлення попередніх вихідних даних Декодера
Перетворення попередніх вихідних даних у векторне представлення для подальшої обробки в Декодері.

Попередні вихідні декодовані дані зсунуті на один крок вперед
На першому кроці декодування – це спеціальний START TOKEN.
На наступних кроках декодування – це генеровані вихідні дані з попередніх кроків Декодера, що використовуються як вхідні дані для наступних кроків.
Це дозволяє моделі прогнозувати наступні елементи у послідовності під час генерації тексту.

Нейронна мережа Feed Forward Neural Network
Обробляє дані для кожного елементу послідовності незалежно, виконуючи нелінійні перетворення для того, щоб виявити більш складні патерни в даних.

Додавання та Нормалізація
1) Початковий вхідний вектор ***додається*** до вихідного вектора, отриманого після шару уваги або нейронної мережі. Це крок **залишкового зв'язку** (residual connection), який допомагає запобігти втратам інформації і стабілізує навчання моделі.
2) Після додавання виконується ***нормалізація*** (наприклад, шарова нормалізація), щоб стандартизувати вектор. Це допомагає запобігти вибуху або затуханню градієнтів під час навчання і сприяє швидшому і стабільнішому навчальному процесу.

Механізм багатоголової уваги
Обчислює ваги для кожної частини вхідної послідовності, допомагаючи моделі визначати важливі частини спектрограми. Кожна голова обробляє різні аспекти вхідних даних паралельно.
У результаті повертає зважені векторні подання, що відображають важливість різних частин спектрограми.

Позиційне кодування
Додавання інформації про позицію елементів у вхідній послідовності, що допомагає моделі враховувати порядок слів у реченні.

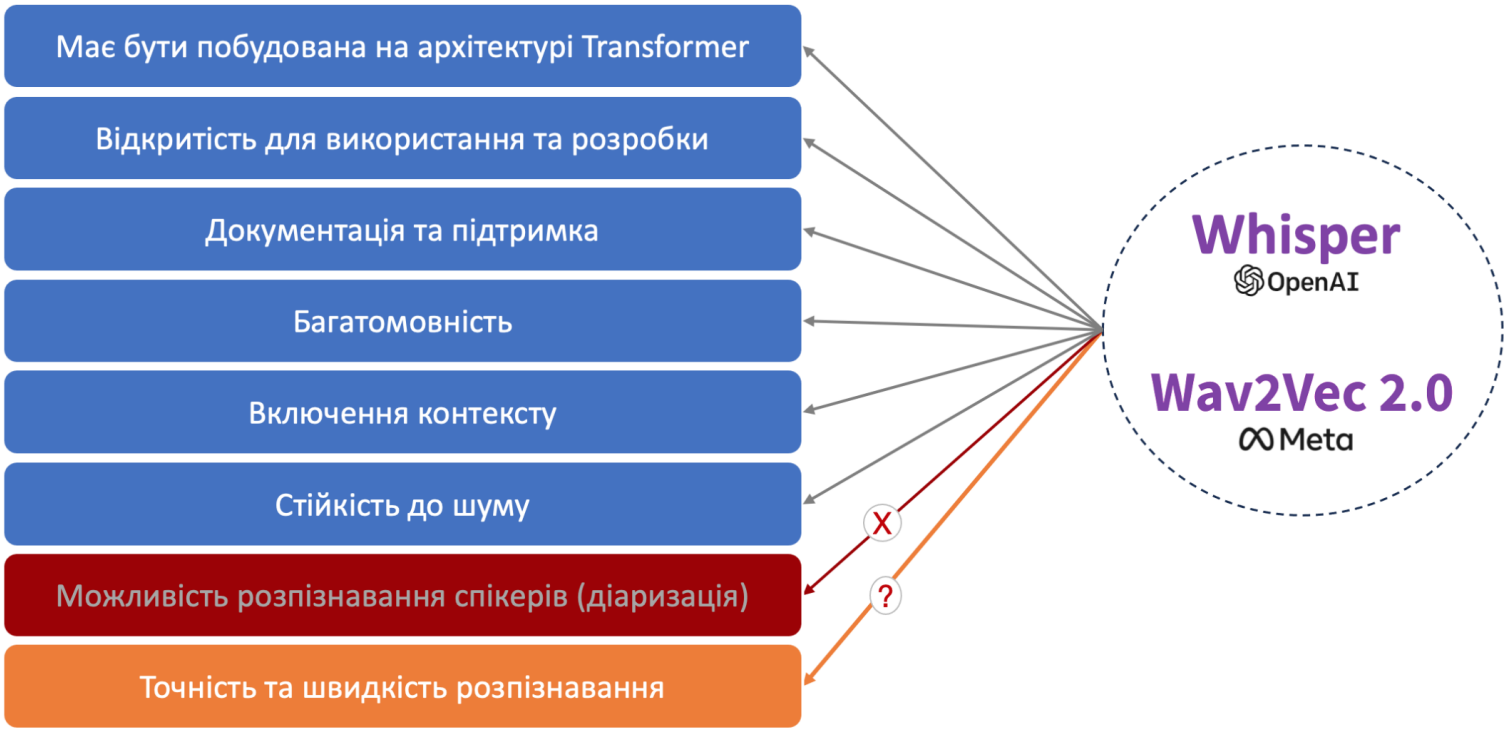
Векторне представлення вхідних даних
Спектрограма перетворена у векторне подання, яке містить числові значення, що описують характеристики вхідних даних (наприклад, амплітуда, частота тощо).

Вхідні дані
Частотна спектрограма аудіо файлу з мовленням.

Encoder (кодер)
Обробляє вхідні дані за допомогою кількох шарів багатоголової уваги і нейронних мереж з прямою передачею

Decoder (декодер)
Генерує вихідні дані за допомогою маскованої багатоголової уваги, багатоголової уваги і нейронних мереж з прямою передачею.

Критерії вибору системи розпізнавання мовлення



Wav2Vec-2.0 чи Whisper ?

$$WER = \frac{S + D + I}{N}$$

Word Error Rate

де S *(Substitutions)* – кількість замін неправильних слів (коли одне слово помилково замінене іншим);

D *(Deletions)* – кількість пропущених слів у транскрипції;

I *(Insertions)* – кількість зайвих доданих слів;

N – загальна кількість слів у правильній транскрипції (референтний текст).

Чим менший показник WER,
тим менше було отримано помилок
і, відповідно, **більш точною є система**

Загальний показник WER транскрипцій,
отриманих за допомогою ASR систем Whisper від OpenAI та Wav2Vec 2.0 від META

Назва набору аудіо файлів	**Середня тривалість аудіо файлів**	**Whisper** (WER)	**Wav2Vec 2.0** (WER)
Діалог людини з штучним інтелектом	39 сек	19.9	36.3
Запис телефонної розмови	2 хв 12 сек	16.6	31.0
Зустрічі та конференції	36 хв 6 сек	13.9	27.4
Відео	2 хв 54 сек	9.7	28.1
Телефонна або відео конференція	39 хв 30 сек	8.9	23.3

Більш точна система,
тому що показник помилок WER має менші значення
для всіх наборів аудіо файлів

Підготовлене програмно-апаратне середовище

- **Графічна карта (GPU) NVIDIA RTX 3070**, з пам'яттю, що відповідає мінімальним вимогам, у розмірі **8 Гб**.
  Для використання GPU, встановлено бібліотеки NVIDIA, зокрема **cuBLAS 11.8** і **cuDNN 8.9.7**

- **Процесор (CPU) Intel Core i7-8700** разом з **оперативною пам'яттю (RAM) об'ємом 32 Гб**,
  що дозволить, окрім WhisperX, запустити інші компоненти платформи на цій же машині.

- **Python 3.10** та **PyTorch 2.0.0** – саме на цих версіях Python та PyTorch буде гарантовано працювати WhisperX

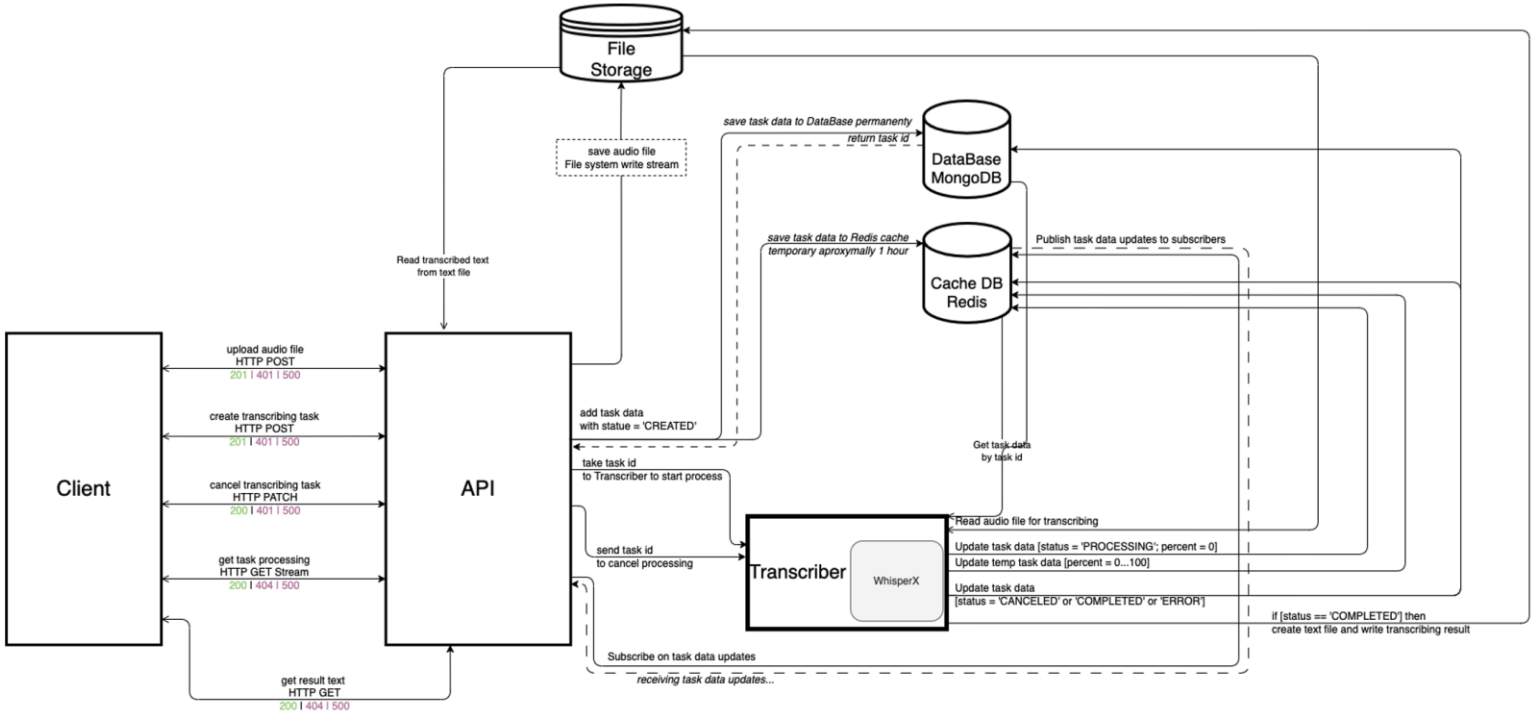
- Запуск та робота Python відбувається через віртуальне середовище за допомогою **Conda**.

- Для виявлення голосової активності (VAD) та розпізнавання спікерів (діаризація) встановлено бібліотеку **pyannote.audio**

- Розробка основної частини платформи на **NodeJS** з використанням бібліотеки **ExpressJS** для взаємодії з клієнтами

- База даних **MongoDB** та кеш бази даних за допомогою **Redis**

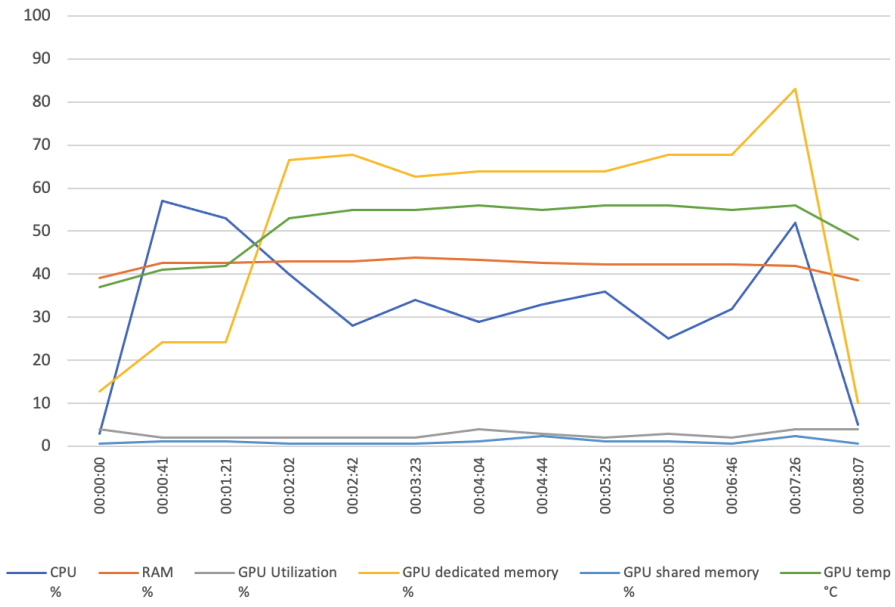
Архітектура програмної платформи



Апробація та результати розробки

Тест 1 - Розпізнавання мовлення з **одного** аудіо файлу,
який має хронометраж 57 хв 39 сек, кодек – PCM, частоту – 16 кГц та один канал (моно)

Time	CPU %	RAM %	GPU Utilization %	GPU dedicated memory %	GPU shared memory %	GPU temperature °C
00:00:00	3,0	39,2	4,0	12,8	0,6	37,0
00:00:41	57,0	42,6	2,0	24,3	1,3	41,0
00:01:21	53,0	42,6	2,0	24,3	1,3	42,0
00:02:02	40,0	42,9	2,0	66,4	0,6	53,0
00:02:42	28,0	42,9	2,0	67,7	0,6	55,0
00:03:23	34,0	43,9	2,0	62,6	0,6	55,0
00:04:04	29,0	43,3	4,0	63,9	1,3	56,0
00:04:44	33,0	42,6	3,0	63,9	2,5	55,0
00:05:25	36,0	42,3	2,0	63,9	1,3	56,0
00:06:05	25,0	42,3	3,0	67,7	1,3	56,0
00:06:46	32,0	42,3	2,0	67,7	0,6	55,0
00:07:26	52,0	42,0	4,0	83,1	2,5	56,0
00:08:07	5,0	38,6	4,0	10,2	0,6	48,0

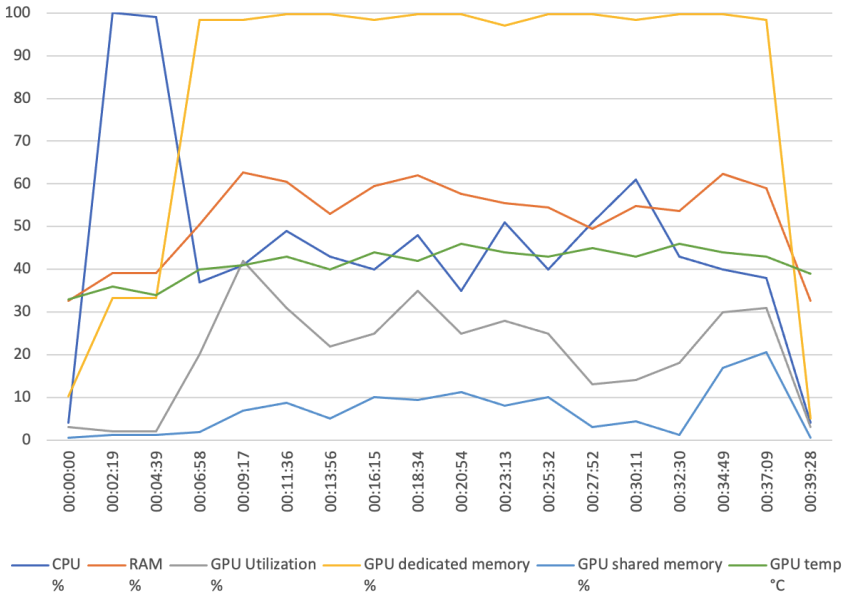


Процес розпізнавання мовлення у текстовий формат з одного аудіо файл триває протягом **8 хв 7 сек**

Апробація та результати розробки

Тест 2 – Одночасне розпізнавання мовлення з **двох** ідентичних аудіо файлів,
які мають хронометраж по 57 хв 39 сек, кодек – PCM, частоту – 16 кГц та один канал (моно)

Time	CPU %	RAM %	GPU Utilization %	GPU dedicated memory %	GPU shared memory %	GPU temp °C
00:00:00	4	32,6	3	10,2	0,6	33
00:02:19	100	39,2	2	33,2	1,3	36
00:04:39	99	39,2	2	33,2	1,3	34
00:06:58	37	50,5	20	98,4	1,9	40
00:09:17	41	62,7	42	98,4	6,9	41
00:11:36	49	60,5	31	99,7	8,8	43
00:13:56	43	53,0	22	99,7	5,0	40
00:16:15	40	59,6	25	98,4	10,0	44
00:18:34	48	62,1	35	99,7	9,4	42
00:20:54	35	57,7	25	99,7	11,3	46
00:23:13	51	55,5	28	97,1	8,1	44
00:25:32	40	54,5	25	99,7	10,0	43
00:27:52	51	49,5	13	99,7	3,1	45
00:30:11	61	54,9	14	98,4	4,4	43
00:32:30	43	53,6	18	99,7	1,3	46
00:34:49	40	62,4	30	99,7	16,9	44
00:37:09	38	58,9	31	98,4	20,6	43
00:39:28	4	32,6	3	5,1	0,6	39

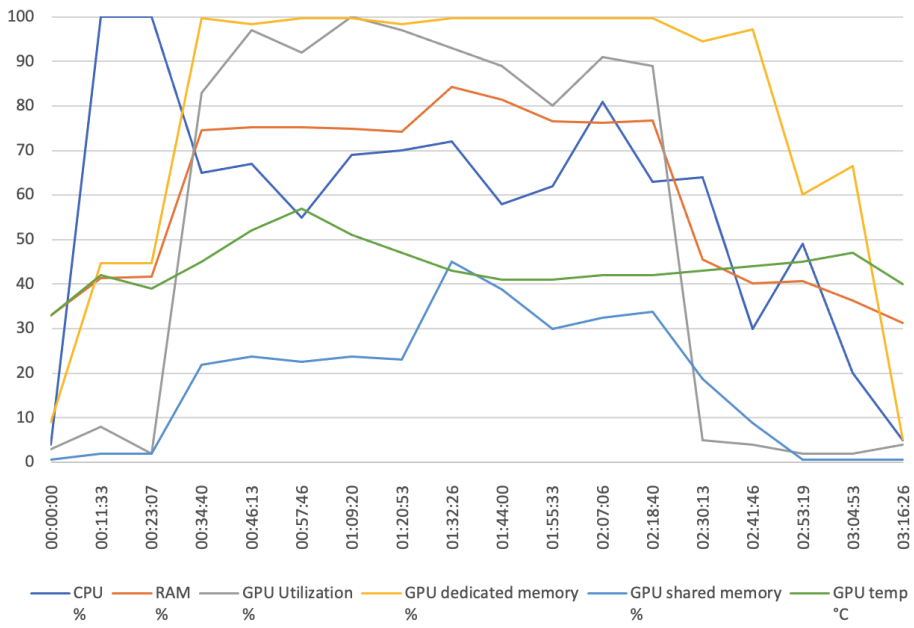


Процес розпізнавання мовлення у текстовий формат одночасно з двох ідентичних аудіо файлів триває протягом **39 хв 28 сек**

Апробація та результати розробки

Тест 3 – Одночасне розпізнавання мовлення з **трьох** ідентичних аудіо файлів,
які мають хронометраж по 57 хв 39 сек, кодек – PCM, частоту – 16 кГц та один канал (моно)

Time	CPU %	RAM %	GPU Utilization %	GPU dedicated memory %	GPU shared memory %	GPU temp °C
00:00:00	4	32,9	3	8,9	0,6	33
00:11:33	100	41,4	8	44,7	1,9	42
00:23:07	100	41,7	2	44,7	1,9	39
00:34:40	65	74,6	83	99,7	21,9	45
00:46:13	67	75,2	97	98,4	23,8	52
00:57:46	55	75,2	92	99,7	22,5	57
01:09:20	69	74,9	100	99,7	23,8	51
01:20:53	70	74,3	97	98,4	23,1	47
01:32:26	72	84,3	93	99,7	45,0	43
01:44:00	58	81,5	89	99,7	38,8	41
01:55:33	62	76,5	80	99,7	30,0	41
02:07:06	81	76,2	91	99,7	32,5	42
02:18:40	63	76,8	89	99,7	33,8	42
02:30:13	64	45,5	5	94,6	18,8	43
02:41:46	30	40,1	4	97,1	8,8	44
02:53:19	49	40,8	2	60,1	0,6	45
03:04:53	20	36,4	2	66,4	0,6	47
03:16:26	5	31,3	4	5,1	0,6	40



Процес розпізнавання мовлення у текстовий формат одночасно з трьох ідентичних аудіо файлів триває протягом **3 год 16 хв 26 сек**

Висновки

Розроблена програмна платформа точно та швидко перетворює мовлення з аудіофайлів у текст за допомогою сучасної системи розпізнавання мовлення WhisperX, модель якої побудована на найсучаснішій архітектурі Transformer

Користувачі можуть в реальному часі спостерігати за процесом розпізнавання мовлення та управляти ним через HTTP REST API

Використання HTTP REST API дозволяє інтегрувати платформу в інші системи

Випробування показали потребу у більшому обсязі графічної пам'яті (понад 8 ГБ) для ефективної роботи платформи

При такому апаратному середовищі, на якому проводилися випробування, з його ресурсами, оптимально розпізнавати лише один або, максимум, два аудіо файли одночасно

Згідно наведених вище висновків надано наступні **рекомендації**:

Оптимізація пам'яті: Розглянути можливості оптимізації використання графічної пам'яті для покращення ефективності

Масштабування: Для великих підприємств рекомендується перехід на мікросервісну архітектуру для розподілу обчислювальних ресурсів між кількома машинами

Підвищення потужності: Розглянути використання кількох і потужніших графічних карт для збільшення обсягу графічної пам'яті та розподілу навантаження між ними, що дозволить обробляти більше аудіофайлів одночасно

Дякую за увагу!