Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

**РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКУ ДЛЯ АВТОМАТИЧНОГО КОНСПЕКТУВАННЯ ТА АНАЛІЗУ НАВЧАЛЬНИХ МАТЕРІАЛІВ З ВИКОРИСТАННЯМ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ**
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-22-1
Шевченко І.С.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Вечірська І.Д.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки

(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика

(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2026 р.

# ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Шевченку Івану Сергійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення вебзастосунку для автоматичного конспектування та аналізу навчальних матеріалів з використанням великих мовних моделей

затверджена наказом університету від 26 травня 2026 року № 435Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 25 червня 2026 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, документація до технологій FastAPI, React, PostgreSQL, Redis, Celery, MinIO, FFmpeg, faster-whisper, CLIP, Tessersct OCR, PyMuPDF, python-pptx, ReportLab nf python-docx, відкриті відеолекції й презентаційні матеріали для тестування застосунку.

4. Перелік питань, що потрібно опрацювати в роботі

1. Огляд існуючих підходів до автоматичного конспектування навчальних матеріалів.
2. Аналіз методів автоматичного розпізнання мовлення та вибір моделі транскрипції.
3.Дослідження способів опрацювання презентаційних матеріалів у форматах PDF та PPTX.
4. Розроблення алгоритму синхронізації слайдів із відеозаписом лекції.
5. Проєктування архітектури вебзастосунку для автоматичного формування конспектів.
6. Реалізація backend та frontend частин застосунку.
7. Тестування корректності роботи а також якості та швидкості створення конспектів.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми автоматичного конспектування навчальних матеріалів, порівняння існуючих підходів до конспектування, схема процесу опрацювання презентацій, схема автоматичного реферування тексту, структура набору навчальних матеріалів, архітектура моделі Whisper, блок-схема алгоритму синхронізації слайдів зі відео, діаграми порівняння ASR моделей та методів виявлення слайдів і LLM моделей, діаграма варіантів використання застосунку, схема взаємодії компонентів системи, діаграма бази даних, схема файлового сховища, схема алгоритму обробки лекції, зображення сторінок вебзастосунку, графіки результатів тестування.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| | | | |
| | | | |

**КАЛЕНДАРНИЙ ПЛАН**

| № з/п | Назва етапів роботи | Строк / терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання на кваліфікаційну роботу | 13.04.2026 | |
| 2 | Аналіз завдання, підбір літератури | 15.04.26-17.04.26 | |
| 3 | Аналіз літератури з проблеми, що вирішується | 17.04.26-22.04.26 | |
| 4 | Аналіз існуючих методів вирішення задачі | 22.04.26-24.04.26 | |
| 5 | Формування кроків вирішення проблеми | 24.04.26-07.05.26 | |
| 6 | Програмна реалізація | 24.04.26-01.06.26 | |
| 7 | Аналіз отриманих результатів | 02.05.26-06.06.26 | |
| 8 | Оформлення пояснювальної записки | 06.06.26-10.06.26 | |
| 9 | Перевірка на нормоконтроль | 10.06.26-19.06.26 | |
| 10 | Перевірка на плагіат | 11.06.26-30.06.26 | |
| 11 | Рецензування | 20.06.26-30.06.26 | |
| 12 | Підготовка презентації та доповіді | 20.06.26-30.06.26 | |
| 13 | Занесення роботи в електронний архів | 30.06.26-09.07.26 | |
| 14 | Попередній захист кваліфікаційної роботи | 30.06.26-09.07.26 | |

Дата видачі завдання 13 квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Вечірська І.Д.
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 80 с., 15 табл., 22 рис., 54 джерела.

ВЕБЗАСТОСУНОК, ГЕНЕРАЦІЯ, КОНСПЕКТУВАННЯ, МУЛЬТИМОДАЛЬНІ МОДЕЛІ, CLAUDE SONNET 4.6, CLIP, PYTHON, WHISPER.

Об’єктом роботи є процес автоматичного формування конспектів навчальних матеріалів на основі мультимодальних даних.

Метою роботи є розроблення вебзастосунку для автоматичного створення структурованих конспектів лекцій.

У роботі проведено аналіз методів автоматичного розпізнавання мовлення, обробки слайдів презентацій, мультимодального аналізу та великих мовних моделей для реферування тексту. Виконано експериментальне порівняння моделей транскрипції, алгоритмів кластеризації та моделей автоматичного реферування.

У результаті роботи реалізовано вебзастосунок для транскрипції відеолекцій, синхронізації слайдів і автоматичного формування структурованих конспектів.

CLAUDE SONNET 4.6, CLIP, GENERATION, MULTIMODAL MODELS, PYTHON, SUMMARIZATION, WEB APPLICATION, WHISPER.

The object of the work is the process of automatic generation of lecture notes based on multimodal educational data.

The goal of the work is to develop a web application for automatic generation of structured lecture notes.

The work includes an analysis of automatic speech recognition methods, presentation slide processing, multimodal analysis, and large language models for text summarization. Experimental comparisons of transcription models, clustering algorithms, and summarization models were carried out.

As a result of the work, a web application was developed for lecture transcription, slide synchronization, and automatic generation of structured lecture notes.

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Транскрипція – текстовий запис мовлення, отриманий після розпізнавання аудіодоріжки лекції.

Шумова точка – елемент даних, який алгоритм кластеризації не відніс до жодного стабільного кластера.

API (Application Programming Interface) – інтерфейс прикладного програмування, набір правил взаємодії між програмними компонентами.

ASR (Automatic Speech Recognition) – автоматичне розпізнавання мовлення, тобто перетворення голосу в текст.

CLIP (Contrastive Language-Image Pre-training) – модель для спільного векторного подання тексту та зображень.

CPU (Central Processing Unit) – центральний процесор комп'ютера.

CUDA (Compute Unified Device Architecture) – програмно-апаратна платформа NVIDIA для паралельних обчислень на GPU.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) – алгоритм кластеризації за щільністю, що дозволяє знаходити кластери та шумові точки.

Docker – платформа контейнеризації для запуску сервісів у відокремлених середовищах.

Embedding – векторне подання об'єкта, що використовується моделями машинного навчання для порівняння змісту або ознак.

endpoint – конкретний шлях API, до якого клієнт надсилає HTTP-запит.

eps (Epsilon) – параметр DBSCAN, що задає максимальну відстань між сусідніми точками.

Frontend – клієнтська частина застосунку, з якою користувач взаємодіє у браузері.

GPT (Generative Pre-trained Transformer) – сімейство генеративних попередньо навчених трансформерних моделей.

GPU (Graphics Processing Unit) – графічний процесор, що може використовуватися для прискорення обчислень моделей.

HDBSCAN (Hierarchical DBSCAN) – ієрархічний варіант DBSCAN для роботи з кластерами різної щільності.

HTTP (HyperText Transfer Protocol) – протокол передавання даних між клієнтом і сервером у вебзастосунках.

JSON (JavaScript Object Notation) – текстовий формат обміну структурованими даними.

JSONB – бінарний формат зберігання JSON-даних у PostgreSQL.

Markdown – легка мова розмітки, використана для збереження та редагування конспекту.

min_samples – параметр DBSCAN, що задає мінімальну кількість елементів для формування кластера.

MKV (Matroska Video) – контейнерний формат відеофайлів.

MLX (Machine Learning eXplore) – фреймворк машинного навчання, оптимізований для Apple Silicon.

MOV – контейнерний формат відеофайлів, поширений в екосистемі Apple.

MP4 (MPEG-4 Part 14) – поширений контейнерний формат відеофайлів.

OCR (Optical Character Recognition) – оптичне розпізнавання символів у зображеннях або сканованих документах.

OpenAPI – специфікація опису REST API, на основі якої генерується документація backend-сервісу.

ORM (Object-Relational Mapping) – підхід до роботи з реляційною базою даних через об'єкти мови програмування.

REST (Representational State Transfer) – архітектурний стиль побудови веб API.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) – метрика оцінювання якості автоматичного реферування тексту.

UMAP (Uniform Manifold Approximation and Projection) – метод зменшення розмірності та візуалізації багатовимірних даних.

URL (Uniform Resource Locator) – адреса ресурсу в мережі або вебзастосунку.

ViT-B/32 (Vision Transformer Base, patch size 32) – варіант візуального трансформера, використаний у CLIP для обробки зображень.

WAV (Waveform Audio File Format) – формат аудіофайлу, до якого система приводить аудіодоріжку перед транскрипцією.

WEBM – відкритий формат відеофайлів для вебсередовища.

WebSocket – протокол двостороннього обміну даними між браузером і сервером у реальному часі.

WER (Word Error Rate) – частка словесних помилок, основна метрика оцінювання якості транскрипції.

YAKE (Yet Another Keyword Extractor) – алгоритм автоматичного виділення ключових слів із тексту.

# ВСТУП

Студенти університету за семестр опрацьовують десятки відеолекцій, презентацій, аудіозаписів семінарів і PDF-статей. Викладачі підтримують кілька паралельних курсів водночас, оновлюють слайди між семестрами й додають нові джерела до програм. Обсяг цифрових матеріалів зростає швидше, ніж швидкість, з якою їх можна переглянути й засвоїти. Проблема зміщується з доступу до інформації на дефіцит часу для її опрацювання. Така ситуація має системний характер і є стійкою тенденцією розвитку сучасної освіти.

Інструмент фіксації залишається попереднім – конспект, написаний від руки або набраний у редакторі. Створити його довго. Треба переглянути лекцію, прослухати незрозумілі моменти повторно, узагальнити й структурувати. Якість результату залежить від уваги студента, його підготовки й навички конспектування. Через це різні студенти виносять із того самого відеозапису різні за змістом нотатки.

Програми вирішують окремі частини проблеми. Є інструменти, що роблять транскрипції аудіо, є алгоритми стислого викладу PDF а також генератори коротких резюме. Але вони найчастіше працюють із одним типом даних, наприклад текст чи аудіо. Водночас сучасні лекції часто поєднують кілька каналів: мовлення викладача, слайди, схеми на дошці, формули. Користувач отримує транскрипт із Whisper, короткий виклад слайдів із Claude і сам зводить це в єдиний документ. Це знижує ефективність таких інструментів.

Великі мовні моделі вміють узагальнювати інформацію, структурувати текст і виділяти ключові ідеї з довгих документів. Мультимодальні версії такі, як: GPT-4o, Gemini 3.5 Pro, Claude Sonnet 4.6 – приймають на вхід текст, зображення й аудіо в межах однієї задачі. З різнорідних вхідних даних вони складають цілісний конспект лекції.

Студенти натрапляють на цю проблему щодня. Матеріалів стає більше, асинхронне навчання розширюється, а часу на підготовку до іспиту чи семінару як не вистачало, так і не вистачає. Виникає запит на інструменти, здатні

узагальнювати інформацію швидше за людину. Ще однією проблемою є доступність навчання для студентів із різними потребами, зокрема осіб із порушеннями слуху, труднощами концентрації або тих, хто навчається нерідною мовою.

За таких умов розроблення спеціалізованого вебзастосунку для автоматизованого формування структурованих конспектів є обґрунтованим. Вебзастосунок як форма реалізації забезпечує незалежність від операційної системи клієнта та відсутність необхідності локального встановлення. Ресурсоємні операції з обробки даних виконуються на серверній стороні, а взаємодія з користувачем здійснюється через браузер.

Метою цього проєкту є практична реалізація описаного підходу — розробка інтелектуального вебзастосунку, який візьме на себе рутинну роботу з конспектування. Планується створити систему, де користувач зможе в одному вікні завантажити повний пакет матеріалів до лекції, завдяки інтеграції API сучасних мультимодальних моделей, швидко отримати зведений та логічно структурований документ.

У результаті застосунок має стати інструментом, який не просто зекономить час на підготовку до занять, а й змістить фокус студента з механічного фіксування інформації на її глибоке розуміння. Це дозволить перетворити хаос із десятків різних файлів на персоналізовану та впорядковану базу знань, роблячи процес навчання ефективнішим, зручнішим та доступнішим для кожного.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВА ЗАДАЧІ

## 1.1 Підходи до автоматичного конспектування навчальних матеріалів

Форма конспекту змінювалася разом із формою лекції. Доки лекція була виключно аудиторним явищем, єдиним способом її зберегти залишався рукописний запис. Аудіо- та відеозапис створили окремий клас цифрових інструментів. За останні роки вони суттєво ускладнились: транскрипція мовлення подешевшала майже до нуля, а великі мовні моделі автоматизували реферування та структурування отриманого тексту.

Рукописне написання було й залишається найпоширенішим способом, щоб робити конспекти. Фішер і Гарріс експериментально довели, що студенти, які пишуть власний конспект і потім його переглядають, пишуть тести краще за тих, хто отримав готові матеріали від лектора для підготовок. Існують формальні методики такі, як Cornell-нотатки а також Bartush Active Method, але у бакалаврському навчанні їх застосовують рідко. Цифрові альтернативи рукопису такі, як наприклад Evernote, Notability та Penultimate підтримують клавіатурне введення поряд із рукописними позначками. Їх користь залежить від того, веде користувач запис активно, чи лише копіює готові фрагменти тексту [1, 2].

Автоматичне розпізнавання мовлення дало змогу конспектувати через транскрипцію з подальшим екстрактивним реферуванням. Лекцію перетворюють на текст системами автоматичного розпізнавання мовлення (Automatic Speech Recognition, ASR) такими, як: Whisper від OpenAI, після чого алгоритм виокремлює з нього найбільш інформативні речення. У дослідженнях лекційного контенту для цього використовують виявлення тематичних меж за паузами в мовленні та алгоритми типу YAKE (Yet Another Keyword Extractor) для виділення ключових слів у ковзному вікні над транскриптом [3]. Перевагою такого підходу є точність формулювань, оскільки текст береться напряму з оригіналу. Водночас недоліком є те, що конспект складається з вихідних речень

дослівно, без перефразування, і нерідко містить надлишкові деталі або втрачає логічний зв'язок між обраними фрагментами.

Великі мовні моделі дали змогу перейти до абстрактивного реферування, а саме: генерації нового тексту, що передає зміст оригіналу своїми словами. На відміну від екстрактивних методів, модель може об'єднати декілька реплік лектора в єдине формулювання, відкинути повтори й відновити логіку викладу. ChatGPT, Claude, Gemini та інші великі мовні моделі виконують реферування довгих документів на рівні, який ще декілька років тому залишався невирішеною задачею [4]. Але згенерований текст іноді містить факти, яких в оригіналі не було, або ключові акценти зміщені. Конспект, що був отриманий від моделей, потрібно порівнювати з вихідним матеріалом на точність перед використанням [3].

Найбільш сучасним напрямом є мультимодальні підходи до конспектування, які враховують різні типи даних такі, як текст, аудіо та відео. Зараз у освітньому процесі лекції часто представлені у вигляді відеозаписів, що супроводжуються слайдами, а твакожусним поясненням викладача. Використання лише текстових методів є недостатнім. Мультимодальні системи використовують інформацію з різних джерел, сегментують лекції, потім визначають ключові моменти й формують більш повні структуровані конспекти [3, 5]. Зокрема, аналіз візуальних елементів (слайдів), аудіо (мовлення) та текстових транскрипцій забезпечує точніше відображення логіки викладу матеріалу.

Підходи до конспектування еволюціонують від ручних і текстоорієнтованих методів до комплексних інтелектуальних систем. Мультимодальний підхід найповніше відповідає реальній структурі сучасного навчального контенту, який поєднує різні канали подання інформації. Водночас такі рішення залишаються найменш представленими серед доступних інструментів.

У перспективі розвиток інструментів конспектування, може, зосередиться на інтеграції мультимодальних рішень безпосередньо в освітній платформі та їх персоналізації під когнітивні особливості конкретного користувача.

## 1.2 Аналіз існуючих програмних рішень

Автоматичне конспектування змінювалося разом із технологіями обробки природної мови та розпізнавання мовлення. Спочатку такі системи просто перетворювали аудіо в текст. Сучасні рішення поєднують різні підходи до аналізу інформації й частково автоматизують процес створення структурованих конспектів.

Існує багато програмних засобів, які реалізують функції автоматичного конспектування або окремі його складові. Залежно від домінуючого підходу до обробки даних, їх можна поділити на кілька груп: сервіси автоматичної транскрипції, інструменти для створення нотаток під час відеозустрічей, рішення на базі великих мовних моделей, сервіси для обробки відеоконтенту, а також інтегровані офісні помічники на основі штучного інтелекту.

Нижче наведено порівняння сучасних програмних рішень для автоматичного конспектування за основними характеристиками, зокрема типами вхідних даних (табл. 1.1), режимом роботи, мовною підтримкою, можливістю редагування результату, засобами експорту, моделлю доступу та приватністю (табл. 1.2) [6–17].

Таблиця 1.1 – Характеристики обробки даних програмних рішень

| **Програмне рішення** | **Аудіо** | **Відео** | **Текст** | **Слайди** |
|---|---|---|---|---|
| Otter.ai | так | так | так | так |
| Fireflies.ai | так | так | так | ні |
| Read.ai | так | так | так | частково |
| Tactiq | так | так | так | ні |
| Notion AI | ні | ні | так | ні |
| NotebookLM | так | частково | так | так |
| ChatGPT | так | частково | так | так |
| Claude | ні | частково | так | так |
| Eightify | так | так | так | ні |

Продовження таблиці 1.1

| **Програмне рішення** | **Аудіо** | **Відео** | **Текст** | **Слайди** |
|---|---|---|---|---|
| Mindgrasp | так | так | так | так |
| Microsoft 365 Copilot | так | частково | так | так |
| Gemini Workspace | так | частково | так | так |

Таблиця 1.2 – Функціональні характеристики програмних рішень

| **Програмне рішення** | **Мовна підтримка** | **Редагування** | **Експорт** | **Модель доступу** | **Приватність** |
|---|---|---|---|---|---|
| Otter.ai | обмежена | так | так | умовно безкоштовний / платний | низька |
| Fireflies.ai | обмежена | так | так | платний | низька |
| Read.ai | обмежена | так | так | платний | низька |
| Tactiq | обмежена | обмежене | так | умовно безкоштовний / платний | низька |
| Notion AI | багатомовна | так | так | платний | середня |
| NotebookLM | багатомовна | так | обмежений | безкоштовний (обмежено) | середня |
| ChatGPT | багатомовна | так | так | умовно безкоштовний / платний | середня |
| Claude | багатомовна | так | так | платний | середня |
| Eightify | обмежена | обмежене | ні | платний | низька |
| Mindgrasp | багатомовна | так | так | платний | низька |
| Microsoft 365 Copilot | багатомовна | так | так | платний | середня |
| Gemini Workspace | багатомовна | так | так | платний | середня |

Проведений аналіз показав, що меншість сучасних рішень орієнтовані на обробку всіх типів даних. Повноцінна підтримка одночасного аналізу відео, аудіо та слайдів реалізована лише частково, а переважна більшість сервісів використовує хмарну модель роботи та платний доступ. Це створює обмеження як з точки зору конфіденційності даних, так і щодо можливості використання таких систем у середовищах із нестабільним доступом до мережі Інтернет.

Сервіси автоматичної транскрипції, наприклад Otter.ai та Fireflies.ai допомагають перетворити мовлення в текст у режимі реального часу або з запису. Їх перевага – швидкість роботи та інтеграція з платформами для відеозв'язку. Водночас є недолік, а саме: результатом є виключно текстова транскрипція без урахування структури навчального матеріалу або візуальних елементів презентації.

Подальшим розвитком транскрипційних сервісів стали інструменти , які автоматично створюють нотатки під час онлайн-зустрічей. Прикладом таких програм є Read.ai і Tactiq. Наведені рішення формують короткі підсумки розмов, виділяють ключові тези та завдання. Попри розширену функціональність, подібні сервіси більш орієнтовані на бізнес-комунікацію, а не на освітній контент.

Окрему групу становлять сервіси на базі великих мовних моделей: Notion AI, NotebookLM, ChatGPT, Claude та інші. Вони формують структуровані текстові конспекти та узагальнюють інформацію на основі вхідного тексту. Важлива особливість можна виділити високу якість формування зв'язного тексту й можливість адаптації стилю викладу. Але для роботи з лекційними матеріалами деякі з них потребують попереднього перетворення аудіо- або відеоданих у текстовий формат.

До окремої категорії належать сервіси для аналізу відеоконтенту такі, як: Eightify і Mindgrasp. Вони швидко формують короткий зміст відеоматеріалу. Практика використання показує, що такі системи зручні для швидкого ознайомлення зі змістом відео, однак підтримка складних візуальних компонентів, таких як презентаційні слайди, реалізована обмежено. До того ж

Eightify працює тільки з сервісом YouTube, що призводить до ще більш обмеженого використання.

Інтегровані офісні помічники на основі штучного інтелекту – Microsoft 365 Copilot і Google Gemini for Workspace – автоматизують роботу з документами, електронною поштою та зустрічами. На відміну від спеціалізованих сервісів конспектування, такі системи є частиною ширших офісних екосистем і не орієнтовані виключно на освітні сценарії використання.

Сучасні програмні рішення здебільшого реалізують окремі складові автоматичного конспектування, однак не забезпечують комплексної підтримки мультимодального навчального контенту. Більшість із них працює у хмарному середовищі, використовує платну модель доступу та лише частково враховує одночасну наявність відео, аудіо й презентаційних матеріалів. Це підтверджує доцільність розроблення спеціалізованого програмного засобу для автоматизованого формування конспектів на основі мультимодальних даних.

## 1.3 Методи аналізу та формування конспекту

### 1.3.1 Методи розпізнавання мовлення

Перетворення усного мовлення у текстову форму це один з ключових етапів автоматичного конспектування навчальних матеріалів. Для цього використовуються ASR-системи, які отримують текстову транскрипцію аудіо або відеозаписів лекцій [18].

ASR-системи будуються на згорткових і трансформерних архітектурах, натренованих на великих корпусах розмічених аудіозаписів [18]. Їх використання автоматизує формування тексту без ручного введення інформації – важливу складову автоматизованого створення конспектів.

Найвідомішою з таких моделей є Whisper від OpenAI [18]. Його особливістю є підтримка понад 100 мов, в тому числі української, і можливість

локального запуску, бо вона є моделлю з відкритим кодом. Whisper використовує підхід слабко контрольованого навчання. Це підвищує точність, якщо на записі наявні шуми, людина говорить з акцентом, або якість запису є поганою, адже модель тренується на великих наборах неструктурованих аудіоданих [18].

Якість ASR-систем визначається метрикою Word Error Rate (WER) (1.1), яка показує співвідношення кількості помилок до кількості слів у транскрипції [18]:

$$WER = \frac{S + D + I}{N}, \tag{1.1}$$

де $S$ - кількість замін слів;

$D$ - кількість пропущених слів;

$I$ - кількість вставлених слів;

$N$ - кількість слів у еталонному тексті.

Менше значення WER відповідає вищій точності розпізнавання мовлення.

Крім Whisper, значного поширення набули хмарні сервіси розпізнавання мовлення, серед яких Google Cloud Speech-to-Text, Azure AI Speech, Deepgram та AssemblyAI [19–22]. Такі системи надають готовий API для інтеграції у програмні застосунки та забезпечують обробку аудіо у режимі реального часу.

Google Cloud Speech-to-Text підтримує широкий перелік мов і виконує автоматичну пунктуацію, визначення мовця та адаптацію до специфічної термінології [19]. Azure AI Speech інтегрується з екосистемою Microsoft і підтримує налаштування власних мовних моделей [20]. Deepgram орієнтований на високу швидкість обробки аудіо та потокове розпізнавання [21], тоді як AssemblyAI додатково підтримує автоматичне визначення тем, емоційного забарвлення мовлення та формування коротких підсумків [22].

До основних характеристик ASR-систем належать підтримка мов, можливість локального запуску, швидкість обробки, точність розпізнавання та спосіб інтеграції у програмні застосунки (табл. 1.3).

Таблиця 1.3 – Порівняння сучасних ASR-систем

| **Система** | **Підтримка української мови** | **Локальний запуск** | **API** | **Робота в реальному часі** | **Додаткові можли-вості** | **Модель доступу** |
|---|---|---|---|---|---|---|
| Whisper | так | так | обмежено | ні | переклад, багато-мовність | Безкошто-вна |
| Google Cloud Speech-to-Text | так | ні | так | так | автоматична пунктуація, diarization | платна |
| Azure AI Speech | так | частково | так | так | кастомні мовні моделі | платна |
| Deepgram | частково | ні | так | так | потокова обробка аудіо | платна |
| Assembly AI | частково | ні | так | так | summary, topic detection | платна |

Важливою перевагою локальних моделей є можливість обробки даних без передачі інформації на сторонні сервера. Це дозволяє підвищити рівень конфіденційності та забезпечити роботу системи без постійного підключення до мережі Інтернет. Водночас хмарні рішення зазвичай мають вищу швидкість роботи та не потребують значних обчислювальних ресурсів на стороні користувача.

Вибір оптимального варіанту залежить від конкретних завдань. Для досягнення найкращого балансу дедалі частіше використовують гібридний підхід: найбільш чутливі дані обробляються локальними моделями для гарантії безпеки, а ресурсомісткі делегують хмарним сервісам.

Процес роботи ASR-систем зазвичай складається з кількох етапів: попередньої обробки аудіо, виділення акустичних ознак, декодування мовлення та формування текстової транскрипції (рис. 1.1). У сучасних системах окремі етапи можуть об’єднуватися в межах єдиної нейромережевої моделі.

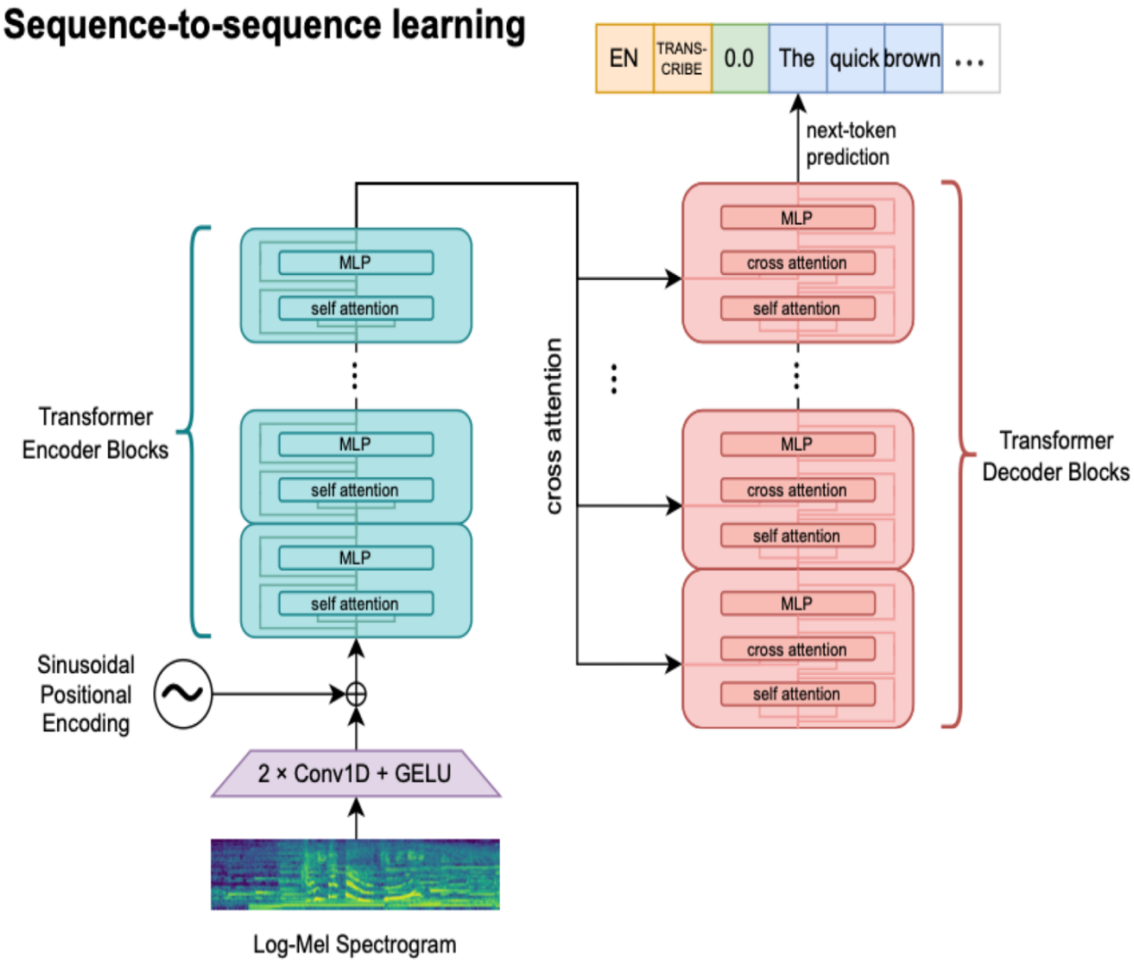


Рисунок 1.1 – Архітектура моделі Whisper [18]

Сучасні системи автоматичного розпізнавання мовлення забезпечують достатню якість транскрипції для задач автоматичного конспектування навчальних матеріалів. Найперспективнішими для використання у цій роботі є моделі з підтримкою української мови та можливістю локального запуску, оскільки вони поєднують високу точність розпізнавання із незалежністю від хмарної інфраструктури. Такий автономний режим роботи додатково гарантує конфіденційність оброблюваних даних, унеможливлюючи їх витік до сторонніх комерційних сервісів. Згодом отримані текстові масиви можуть безпечно передаватися іншим алгоритмам для формування фінального конспекту.

### 1.3.2 Методи опрацювання слайдів презентацій

Аналіз презентаційних слайдів є важливою складовою конспектування навчальних матеріалів, бо значна частина інформації під час лекцій подається у візуальній формі. Слайди можуть містити як ключові тези та визначення, так і формули, зображення та інші елементи, які не завжди повністю озвучуються під час усного пояснення.

Для опрацювання презентацій текстова та структурна інформація витагується з PDF і PPTX файлів. Одним з поширених інструментів для роботи з PDF-документами є бібліотека PyPDF [23], яка отримує текстовий вміст сторінок, метадані документа а також структурні елементи.

Для роботи з презентаціями зробленими у PowerPoint використовують бібліотеку python-pptx [24], що дає змогу отримувати текст без необхідності додаткового розпізнавання символів у випадках, коли презентація містить текстовий шар.

У багатьох випадках в перезентаціях використовують матеріали у вигляді зображень або сканованих сторінок. В такому випадку застосовуються системи оптичного розпізнавання символів (Optical Character Recognition, OCR), які перетворюють текст із зображення у цифрову форму.

Одним з найпоширеніших OCR-інструментів є Tesseract OCR [25]. Його перевагою є відкритий вихідний код і можливість локального запуску без використання хмарних сервісів. Tesseract підтримує значну кількість мов, в тому числі й українську. Але якість розпізнавання залежить від якості зображення, шрифтів та складності оформлення слайдів.

Альтернативним рішенням є Google Vision API [26], який використовує хмарні моделі комп'ютерного зору для розпізнавання тексту та аналізу зображень. Перевагою є висока точність розпізнавання, навіть якщо структура документів є складною. Але використання цього сервісу передає дані до інфраструктур та потребує з’єднання з інтернетом.

Порівняння засобів опрацювання слайдів наведено нижче (табл. 1.4).

Таблиця 1.4 – Порівняння засобів опрацювання презентацій

| Засіб | Тип даних | OCR | Локальний запуск | API / бібліотека | Особливості |
|---|---|---|---|---|---|
| PyPDF | PDF | ні | так | бібліотека Python | витягування тексту та метаданих |
| python-pptx | PPTX | ні | так | бібліотека Python | робота зі структурою слайдів |
| Tesseract OCR | зображення | так | так | OCR-система | підтримка української мови |
| Google Vision API | PDF, зображення | так | ні | хмарний API | висока точність розпізнавання |

Під час автоматичного аналізу презентацій важливим є не лише витягування тексту, а й визначення структури документа. До основних структурних елементів належать: заголовки, марковані списки, таблиці, зображення, формули, підписи до рисунків.

Наявність структурованої інформації дає змогу формувати якісніший конспект та визначати логічні зв'язки між окремими частинами навчального матеріалу.

Процес аналізу презентацій зазвичай включає відкриття документа, витягування текстового вмісту, OCR-обробку зображень та подальше формування структурованого представлення даних (рис. 1.2). Отримана інформація може використовуватись разом із транскрипцією мовлення для формування мультимодального конспекту лекції.

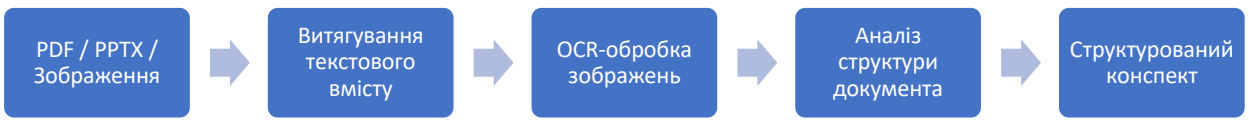


Рисунок 1.2 – Схема процесу опрацювання презентацій

Завдяки сучасним методам опрацювання презентаційних матеріалів стало можливо автоматично отримувати текстову та структурну інформацію зі слайдів.

Найефективнішим є комбінування прямого аналізу цифрових документів із використанням OCR для зображень для повного охоплення навчального контенту.

### 1.3.3 Методи реферування на основі великих мовних моделей

Після отримання текстової транскрипції лекційного матеріалу наступним етапом є формування структурованого конспекту. Для цього використовуються методи автоматичного реферування тексту, які скорочують обсяг інформації й виділяють ключові змістові елементи.

Сучасні системи автоматичного реферування можна поділити на два основні підходи: екстрактивне та абстрактивне реферування [27].

Екстрактивне реферування обирає найважливіші речення або фрагменти тексту без зміни їх формулювання [27]. Цей підхід забезпечує високу точність передачі змісту. Але сформований текст часто менш зв’язний і може містити повтори.

Абстрактивне реферування генерує новий текст на основі аналізу змісту документа [27]. Модель формує узагальнений конспект із використанням власних формулювань. Саме цей підхід використовують сучасні великі мовні моделі (Large Language Models, LLM).

Однією з перших спеціалізованих моделей для абстрактивного реферування був PEGASUS [27], яка використовує попереднє навчання на основі відновлення прихованих речень документа. Такий підхід підвищує якість формування коротких змістовних резюме.

Сучасні великі мовні моделі, а саме: GPT-4o, Claude Sonnet 4.6, Gemini 3.5 Pro, Llama 4 Maverick, Mistral Large 3 і Qwen3 [28–33] – забезпечують ширші можливості роботи з текстом. Вони формують короткі підсумки, структурують матеріал, виділяють ключові поняття, створюють списки тез та адаптують стиль викладу під конкретний формат конспекту.

Одна з найважливіших характеристик сучасних мовних моделей – це розмір контекстного вікна: обсяг тексту, який модель може обробляти одночасно [23–30]. Для автоматичного конспектування лекцій це важливо, оскільки тривалі відеолекції можуть містити десятки тисяч слів.

Крім розміру контекстного вікна, важливі характеристики великих мовних моделей – підтримка мультимодальних даних, можливість автономного локального розгортання, вартість використання API та якість генерації тексту (табл. 1.5).

Таблиця 1.5 – Порівняння сучасних великих мовних моделей

| **Модель** | **Контекстне вікно** | **Мульти-модальність** | **Локальний запуск** | **API-доступ** | **Особливості** |
|---|---|---|---|---|---|
| GPT-4o | до 128k токенів | так | ні | так | висока якість генерації та підтримка аудіо й зображень |
| Claude Sonnet 4.6 | до 200k токенів | так | ні | так | ефективна робота з довгими документами та зображеннями |
| Gemini 3.5 Pro | до 1M токенів | так | ні | так | довгий контекст і нативна підтримка PDF та мультимодальних даних |
| Llama 4 Maverick | до 1M токенів | так | так | так | open-weight модель із мультимодальністю та локальним розгортанням |

Продовження таблиці 1.5

| Модель | Контекстне вікно | Мульти-модальність | Локальний запуск | API-доступ | Особливості |
|---|---|---|---|---|---|
| Mistral Large 3 | до 256k токенів | так | так | так | open-weight мультимодальна модель з високою швидкістю роботи |
| Qwen3 | до 256k токенів | частково | так | так | сильна багатомовність і підтримка локального запуску |

Для оцінювання якості автоматичного реферування використовуються спеціалізовані набори даних та метрики. Найпоширенішою автоматичною метрикою залишається ROUGE [34], а серед уживаних наборів даних для узагальнення текстів і діалогів часто використовують CNN/DailyMail [35] та SAMSum [36].

Можливість генерації неточностей або так званих «галюцинацій», коли модель формує інформацію, відсутню у вихідному тексті є одною з основних проблем абстрактивного реферування [9]. Це особливо важливо для автоматичного конспектування навчальних матеріалів, оскільки помилки можуть призводити до спотворення змісту лекції.

У більшості сучасних систем процес формування конспекту включає попередню обробку тексту, аналіз контексту, генерацію узагальненого представлення й формування фінального результату (рис. 1.3).

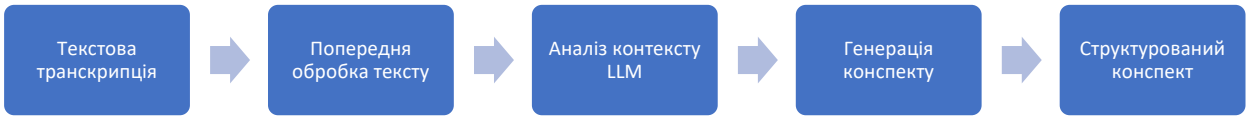


Рисунок 1.3 – Схема процесу автоматичного реферування тексту

Великі мовні моделі забезпечують високий рівень автоматизації процесу формування конспектів і створюють структуровані узагальнення великих обсягів тексту. Найперспективнішими для використання у цій роботі є моделі з великим контекстним вікном і підтримкою мультимодальних даних, оскільки вони забезпечують можливість одночасної обробки різних типів навчального контенту.

## 1.4 Постановка задачі

Згідно з проведеним вище аналізом, на сьогодні бракує програмних засобів, здатних комплексно опрацьовувати навчальні матеріали, що містять відео, аудіо та презентаційні слайди. Більшість наявних рішень орієнтовані на окремі етапи обробки даних – транскрипцію мовлення, реферування тексту або створення коротких підсумків відеоматеріалів. При цьому повноцінне поєднання різних типів навчального контенту в межах єдиного процесу формування конспекту реалізоване лише частково. Також значна частина сучасних сервісів є хмарними й комерційними, що може створювати обмеження з погляду конфіденційності, вартості використання та залежності від сторонньої інфраструктури.

Це створює актуальну потребу в розробленні вебзастосунку, здатного автоматично формувати структурований конспект навчального матеріалу на основі аналізу відео, аудіо та презентаційних матеріалів із використанням сучасних методів розпізнавання мовлення, опрацювання документів і великих мовних моделей.

Об’єктом роботи є процес автоматизованого формування конспекту навчального матеріалу на основі мультимодального контенту.

Метою роботи є розроблення вебзастосунку для автоматичного формування структурованих конспектів навчальних матеріалів на основі аналізу відео, аудіо та презентаційних слайдів.

Для досягнення цієї мети необхідно вирішити такі завдання:

− провести аналіз існуючих підходів до автоматичного конспектування навчальних матеріалів;

− проаналізувати сучасні програмні рішення для транскрипції, реферування та створення автоматичних нотаток;

− дослідити методи автоматичного розпізнавання мовлення для отримання текстової транскрипції лекційного матеріалу;

− дослідити засоби опрацювання презентаційних матеріалів, а саме: витягування тексту з PDF та PPTX, а також OCR-розпізнавання;

− проаналізувати можливості великих мовних моделей для формування структурованого конспекту;

− розробити алгоритм оброблення мультимодального навчального матеріалу;

− спроєктувати архітектуру вебзастосунку для автоматичного формування конспектів;

− реалізувати вебзастосунок для завантаження навчальних матеріалів, їх оброблення та формування результату;

− провести тестування розробленого застосунку та оцінити якість сформованих конспектів.

# 2 РОЗРОБЛЕННЯ ТА АНАЛІЗ МОДЕЛЕЙ І АЛГОРИТМІВ ДЛЯ АВТОМАТИЧНОГО КОНСПЕКТУВАННЯ НАВЧАЛЬНИХ МАТЕРІАЛІВ

## 2.1 Аналіз обсягу й структури навчального контенту

Для оцінки типового обсягу навчальних матеріалів було проведено невелике опитування серед студентів технічних спеціальностей. В опитуванні взяли участь 24 особи. Респонденти навчалися на 3-4 курсі та робили запис лекційних занять.

Анкета містила три групи запитань:

− тривалість і кількість відеолекцій протягом семестру;

− формати супровідних матеріалів;

− час, який студенти витрачають на ручне створення конспекту.

Результати показали, що більшість дисциплін містять від 20 до 45 годин відеоматеріалів за семестр. У студентів ІТ-спеціальностей це значення виявилось вищим через велику кількість лабораторних пояснень і записаних практичних занять.

Частина викладачів використовує лише презентації. Інші поєднують PPTX-файли з PDF-конспектами або рукописними поясненнями на графічному планшеті. У кількох курсах траплялись лекції без слайдів – лише відеозапис екрана й голос лектора.

Середній час створення конспекту вручну для однієї 80-хвилинної лекції становив приблизно 105-120 хвилин. Студенти, які переглядали лекцію повторно для уточнення матеріалу, витрачали більше години.

З огляду на такі великі об’єми контенту, впровадження автоматизованих мультимодальних систем для генерації конспектів є об'єктивною необхідністю, що дозволить суттєво оптимізувати та спростити навчальний процес для того щоб вивільнити час для практичного засвоєння знань.

У таблиці 2.1 наведено узагальнені результати опитування.

Таблиця 2.1 – Результати аналізу структури навчального контенту

| **Параметр** | **Значення** |
|---|---|
| Кількість опитаних | 24 |
| Середня тривалість лекції | 78 хв |
| Середня кількість відеолекцій за семестр | 31 |
| PPTX-презентації | 83 % |
| PDF-матеріали | 54 % |
| Рукописні пояснення | 29 % |
| Лекції без слайдів | 17 % |
| Середній час ручного конспектування | 113 хв |

Більшість лекцій містила одразу декілька типів контенту. Найчастіше зустрічалася комбінація: відео, аудіо пояснення, презентаційні слайди.

Саме така структура матеріалів використовувалась надалі під час формування набору даних для експериментів.

## 2.2 Формування набору даних для досліджень

Для тестування алгоритмів автоматичного конспектування було сформовано власний набір навчальних матеріалів. До нього увійшли 16 відеолекцій українською та англійською мовами.

Підбір лекцій виконувався так, щоб набір містив різні типи контенту. Частина відео складалася переважно зі слайдів і голосових пояснень. Інші лекції містили фрагменти коду, математичні формули або рукописні записи викладача.

До набору ввійшли матеріали з таких дисциплін: програмування, математичний аналіз, комп’ютерна лінгвістика, машинне навчання, бази даних, комп’ютерні мережі.

Тривалість лекцій коливалась від 18 до 96 хвилин. Найбільший обсяг даних

дали записи курсів із програмування, оскільки вони містили довгі пояснення й демонстрацію коду.

Для кожної лекції було збережено: відеофайл, презентацію у форматі PPTX або PDF, автоматичну транскрипцію, еталонний конспект.

Еталонні конспекти створювалися вручну. Під час розмітки фіксувалися: основні тези лекції, структура слайдів, пояснення викладача, ключові терміни. Такі конспекти використовувались як еталонні тексти під час оцінювання ROUGE-метрик у розділі 2.7 [34].

Характеристики сформованого набору даних наведено в таблиці 2.2.

Таблиця 2.2 – Характеристики набору даних

| **Дисципліна** | **Мова** | **Тривалість, хв** | **Кількість слайдів** | **Формат презентації** |
|---|---|---|---|---|
| Програмування | укр | 64 | 28 | PPTX |
| Машинне навчання | англ | 91 | 42 | PDF |
| Математичний аналіз | укр | 57 | 19 | PDF |
| Комп’ютерні мережі | англ | 73 | 31 | PPTX |
| Бази даних | укр | 49 | 24 | PPTX |
| Алгоритми та структури даних | англ | 82 | 37 | PDF |
| Комп’ютерна лінгвістика | укр | 46 | 18 | PPTX |
| Операційні системи | англ | 88 | 39 | PDF |
| Дискретна математика | укр | 53 | 22 | PDF |
| Data Science | англ | 96 | 44 | PPTX |
| Вебпрограмування | укр | 41 | 17 | PPTX |

Продовження таблиці 2.2

| **Дисципліна** | **Мова** | **Тривалість, хв** | **Кількість слайдів** | **Формат презентації** |
|---|---|---|---|---|
| Software Engineering | англ | 78 | 33 | PDF |
| Комп’ютерна графіка | укр | 59 | 26 | PPTX |
| Artificial Intelligence | англ | 84 | 36 | PDF |
| Теорія ймовірностей | укр | 52 | 21 | PDF |
| Natural Language Processing | англ | 87 | 41 | PPTX |

Для частини лекцій додатково зберігалися часові мітки зміни слайдів. Це дозволило тестувати алгоритми синхронізації відеоряду й презентаційних матеріалів.

Набір даних вийшов неоднорідним. У цьому був окремий сенс. Whisper large-v3 працює стабільно на чистих аудіозаписах, але гірше розпізнає лекції з фоновим шумом або швидкою зміною мовців. Аналогічно CLIP точніше співставляє слайди зі статичними кадрами, ніж із відео, де присутня анімація або курсор викладача.

Різноманітність матеріалів дала можливість оцінити алгоритми в умовах, ближчих до реального навчального процесу.

## 2.3 Аналіз моделей автоматичного розпізнавання мовлення

Транскрипція лекцій стала першим етапом обробки навчального контенту. Помилки ASR-моделі накопичуються далі у конвеєрі обробки: неправильні терміни потрапляють у запит, велика мовна модель гірше формує підсумок, а прив’язка пояснень до слайдів зміщується в часі. Через це якість транскрипції перевірялась окремо.

У роботі тестувалися три моделі Whisper від OpenAI: medium, large-v3, large-v3-turbo.

Додатково порівнювались дві локальні реалізації: faster-whisper, whisper.cpp.

Тестування виконувалось на сформованому наборі лекцій із розділу 2.2. Для оцінювання використовувався показник WER (Word Error Rate). Метрика показує частку помилково розпізнаних слів відносно еталонної транскрипції.

### 2.3.1 Модель whisper від OpenAI

Модель whisper OpenAI опублікувала у 2022 році. Модель навчалась на 680 тисячах годин багатомовного аудіо й до цього часу використовується як базовий ASR-інструмент для автономної транскрипції [18].

Модель whisper medium працює швидше за великі моделі, але частіше помиляється на технічних термінах. Особливо це помітно в українських лекціях із програмування, де модель плутає англомовні назви бібліотек і скорочення.

Версія large-v3 дала найнижчий WER серед протестованих моделей. Недолік – швидкість. Обробка 60-хвилинної лекції на RTX 4060 займала приблизно 17 хвилин.

Версія large-v3-turbo працювала швидше. Різниця в якості виявилась невеликою, але модель частіше втрачала розділові знаки й часові паузи в довгих поясненнях.

Для української мови найбільше помилок виникало у трьох випадках:

− змішані українсько-англійські речення;

− математичні позначення;

− швидке мовлення викладача.

Тому large-v3 є найкращим вибором для технічних україномовних лекцій.

Англомовні лекції моделі обробляли стабільніше. Різниця між моделями medium і large-v3 там була менш помітною.

На рисунку 2.1 показано процес транскрипції лекційного аудіо моделлю whisper.

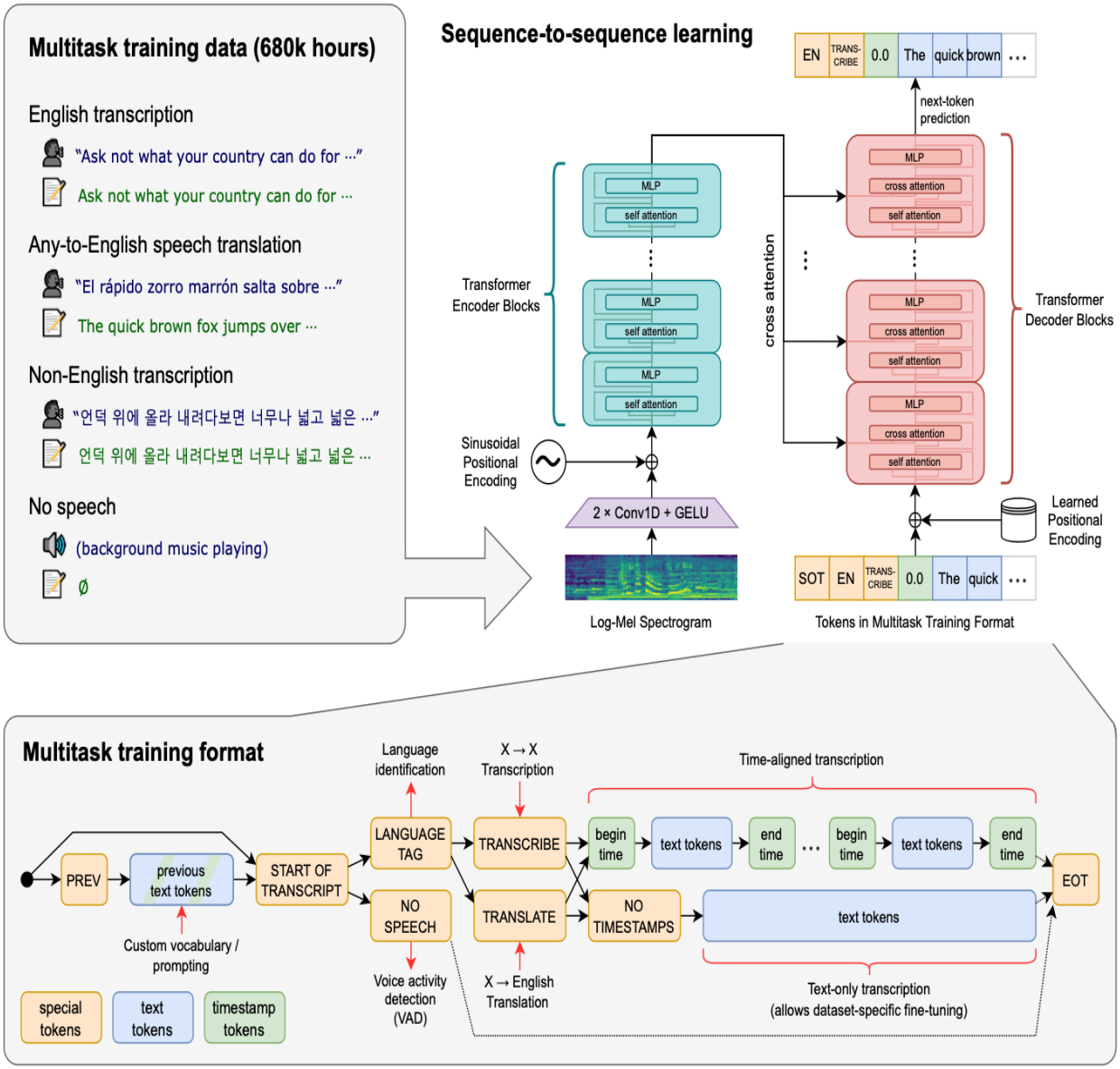


Рисунок 2.1 – Архітектура моделі whisper [18]

### 2.3.2 Моделі faster-whisper і whisper.cpp

Модель faster-whisper використовує оптимізовану реалізацію виконання моделі через CTranslate2. У нашому середовищі ця модель працювала у режимі центрального процесора, тому за швидкістю поступалась MLX-реалізаціям на Apple Silicon.

Під час тестування faster-whisper large-v3 обробляла 60 хв лекції приблизно за 46 хв. Для порівняння, whisper-large-v3-turbo потребувала 4 хв, а whisper medium – 3 хв. Така різниця пояснюється тим, що faster-whisper у цьому експерименті працювала лише на центральному процесорі, тоді як MLX-версії використовували апаратне прискорення Apple Silicon.

За українським набором faster-whisper large-v3 дала найнижчий WER (13.0 %), але перевага в якості не компенсувала значно довший час обробки лише на центральному процесорі.

Інструмент whisper.cpp тестувався окремо. Ця реалізація орієнтована на виконання на центральному процесорі й локальний запуск без MLX або CUDA. Модель medium працювала стабільно, але для українського набору WER зріс до 50.5 %.

Перевага whisper.cpp – автономність. Недолік – вища кількість помилок і нижча якість транскрипції в порівнянні з MLX-рішеннями.

### 2.3.3 Порівняння WER і вибір моделі

Для англомовної лекції найкращий стабільний результат показала Whisper large-v3-turbo з WER 4.4 %. Окремий прогін Whisper large-v3 для англійської дав повторювану транскрипцію з галюцинаціями, тому це значення не використовувалося в таблиці.

Для українського набору найнижчий WER показала faster-whisper large-v3, але як компроміс між якістю та швидкістю для основного конвеєра обробки доцільно використовувати whisper-large-v3-turbo.

Інструмент whisper.cpp виявився корисним для локального запуску на центральному процесорі, але для обробки великих лекцій його точності та швидкості недостатньо.

У таблиці 2.3 показано порівняння моделей.

Таблиця 2.3 – Порівняння ASR-моделей за WER

| **Модель** | **Мова** | **Середній WER, %** | **Час обробки 60 хв відео** | **GPU** |
|---|---|---|---|---|
| Whisper medium | укр | 19.2 | 3 хв | так |
| Whisper large-v3 | укр | 13.7 | 29 хв | так |
| Whisper large-v3-turbo | укр | 13.6 | 4 хв | так |
| faster-whisper large-v3 | укр | 13.0 | 46 хв | ні |
| whisper.cpp medium | укр | 50.5 | 11 хв | ні |
| Whisper medium | англ | 8.8 | 3 хв | так |
| Whisper large-v3 | англ | – | 29 хв | так |
| Whisper large-v3-turbo | англ | 4.4 | 4 хв | так |
| faster-whisper large-v3 | англ | 6.1 | 46 хв | ні |
| whisper.cpp medium | англ | 6.6 | 11 хв | ні |

## 2.4 Виявлення слайдів у відеоряді

Транскрипції недостатньо для побудови повного конспекту лекції. Частина інформації міститься лише на слайдах: формули, фрагменти коду, схеми або таблиці. Через це відеоряд потрібно синхронізувати з презентацією на основі зіставлення візуальних ознак кадрів і слайдів [4, 37, 38].

У роботі тестувалися три підходи:

− порівняння кадрів через векторні подання CLIP;

− виявлення меж сцен через PySceneDetect;

− OCR-порівняння тексту.

PySceneDetect добре працював для різких переходів між слайдами. Проблема виникала в лекціях із плавною анімацією або записом екрана. У таких

випадках бібліотека часто пропускала момент зміни слайда або створювала зайві переходи.

OCR-порівняння дало кращі результати для PDF-презентацій із великою кількістю тексту. Якщо на слайді переважали схеми або зображення, точність різко падала. Tesseract також погано розпізнавав дрібний текст після стиснення відео. Для структурно складних зображень у суміжних задачах застосовують методи сегментації на основі матриць збігів і енергетичних характеристик Лавса [39–41].

Найстабільніший результат дала покадрова векторизація через CLIP, оскільки такий підхід близький до дескрипторних методів побудови ознак і пошуку зображень [37, 38, 42]. Модель переводить зображення у векторні подання, після чого кадри відео можна порівнювати зі слайдами за косинусною подібністю.

Для тестування використовувалась модель ViT-B/32. Вона працювала швидше за великі варіанти CLIP і не вимагала значного обсягу відеопам’яті.

Після отримання векторних подань для кадрів відео та слайдів потрібно визначити, наскільки вони схожі між собою. Для цього в роботі використовувалась косинусна подібність.

CLIP формує векторні подання у спільному векторному просторі. Якщо кадр відео відповідає певному слайду, вектори розташовуються близько один до одного. Якщо зображення різні, кут між векторами збільшується.

Косинусна подібність обчислюється за формулою (2.1):

$$\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}, \tag{2.1}$$

де $A$ – векторне подання кадру відео;

$B$ – векторне подання слайда;

$A \cdot B$ – скалярний добуток векторів;

$\|A\|$ і $\|B\|$ – довжини векторів.

Алгоритм синхронізації складався з кількох етапів:

Крок 1. Із відео виділялись кадри з інтервалом 1 кадр на секунду.

Крок 2. Усі слайди презентації конвертувались у PNG-зображення.

Крок 3. CLIP генерував векторні подання: для кадрів відео, для слайдів презентації.

Крок 4. Для кожного кадру обчислювалась косинусна подібність з усіма слайдами.

Крок 5. Слайд із найбільшим значенням подібності прив’язувався до поточного моменту відео.

Крок 6. Короткі випадкові переходи фільтрувались через кластеризацію сусідніх кадрів.

На рисунку 2.2 наведено блок-схему алгоритму синхронізації слайдів.

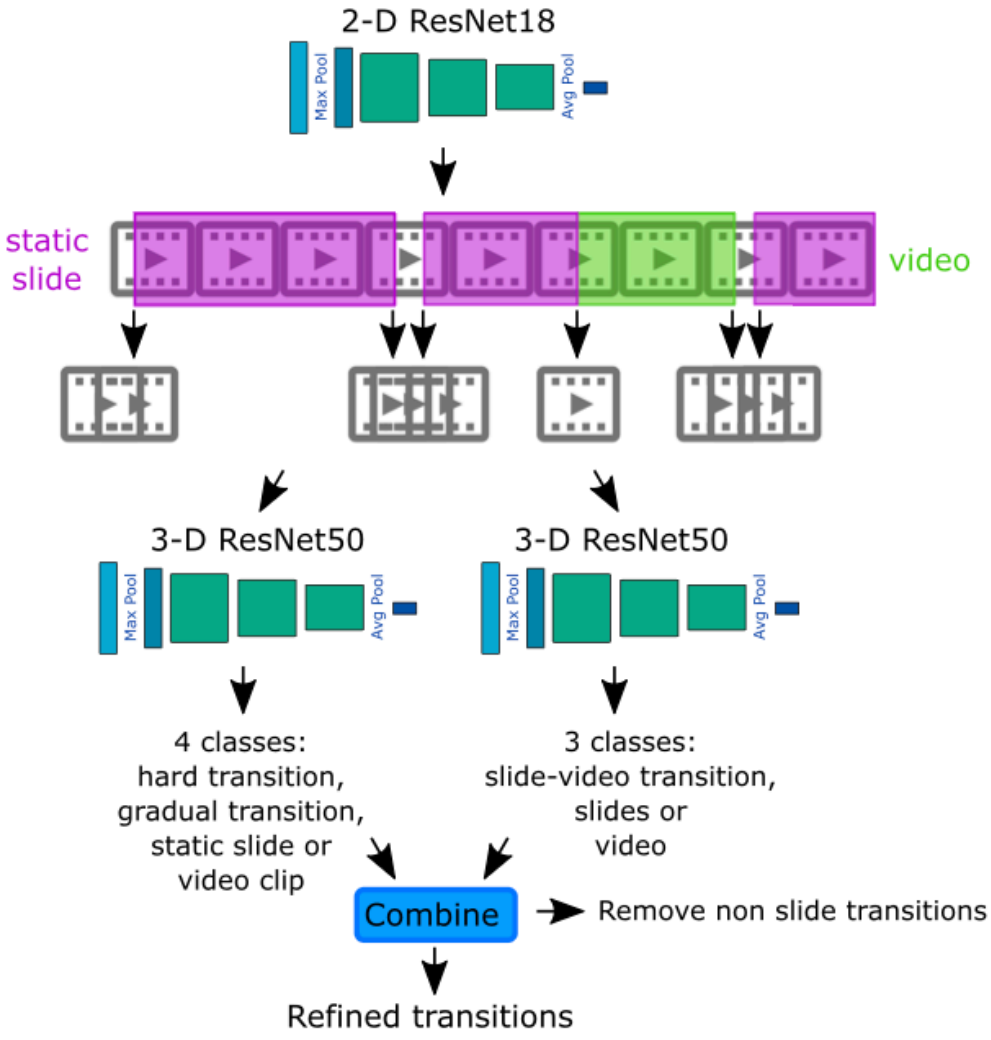


Рисунок 2.2 – Блок-схема алгоритму виявлення моменту зміни слайда

Під час тестування виникли дві проблеми. Перша – курсор викладача. Якщо він активно рухався поверх презентації, подібність між кадром і слайдом зменшувалась. Друга проблема – анімовані слайди, де текст з’являвся поступово.

Для часткового вирішення цього використовувалось згладжування результатів по часовому вікну. Якщо новий слайд з’являвся лише на 1–2 кадрах, алгоритм ігнорував таку зміну.

У таблиці 2.4 наведено порівняння протестованих підходів.

Таблиця 2.4 – Порівняння методів виявлення слайдів

| **Метод** | **Переваги** | **Недоліки** |
|---|---|---|
| PySceneDetect | швидка обробка | погано працює з анімаціями |
| OCR-порівняння | висока точність для текстових слайдів | нестабільний OCR |
| CLIP similarity | стабільна робота зі схемами й зображеннями | вища обчислювальна складність |

## 2.5 Міра подібності між кадром відео і слайдом

У роботі тестувалась евклідова відстань. Вона гірше працювала для векторних подань CLIP, оскільки сильніше залежала від довжини векторів. Косинусна подібність виявилась стабільнішою: модель правильно співставляла слайди навіть після масштабування, стиснення відео або зміни яскравості кадру.

Для однакових або майже однакових зображень косинусна подібність у більшості випадків перевищувала 0.85, а для ідентичних кадрів середнє значення становило 0.9211.

Для кадрів з іншої лекції середнє значення дорівнювало 0.4482, тоді як для анімованих слайдів воно становило 0.8857.

У таблиці 2.5 наведено експериментальну статистику косинусної подібності для різних типів кадрів: кількість пар, середнє значення та інтервал P10–P90.

Таблиця 2.5 – Експериментальні значення косинусної подібності

| **Тип порівняння** | **Кількість пар** | **Середнє** | **P10–P90** |
|---|---|---|---|
| Ідентичний слайд і кадр | 64 | 0.9211 | 0.9052–0.9371 |
| Слайд із незначними змінами | 212 | 0.8983 | 0.8521–0.9316 |
| Анімований слайд | 29 | 0.8857 | 0.8495–0.9151 |
| Інший слайд тієї ж лекції | 19825 | 0.6346 | 0.4845–0.8586 |
| Кадр з іншої лекції | 31482 | 0.4482 | 0.3383–0.5575 |

## 2.6 Кластеризація кадрів за подібністю до слайдів

Після обчислення косинусної подібності кадри відео все ще містили шум. Частина з них помилково прив’язувалась до сусідніх слайдів через анімації, рух курсора та короткі переходи між сценами. Через це результати потрібно згладити.

Для цього тестувалися два алгоритми кластеризації: DBSCAN, HDBSCAN.

Обидва підходи працюють без попереднього задання кількості кластерів. Це важливо для відеолекцій, оскільки число слайдів у різних записах відрізняється.

DBSCAN групує об’єкти за щільністю розташування у векторному просторі. Алгоритм використовує два параметри:

− eps – максимальна відстань між сусідніми точками;

− min_samples – мінімальна кількість елементів у кластері.

Кадри, що не потрапили до жодної групи, позначаються як шумові точки.

Алгоритм кластеризації HDBSCAN працює схожим чином, але краще обробляє кластери різної щільності. Недолік – вища складність і повільніша обробка великих наборів кадрів.

Для кластеризації використовувались: векторні подання кадрів, косинусна подібність, часові мітки відео.

Алгоритм обробки складався з кількох етапів:

Крок 1. Для кожного кадру обчислювалось векторне подання через CLIP.

Крок 2. Визначалась косинусна подібність між кадром і слайдами презентації.

Крок 3. Формувався набір векторів із часовими координатами.

Крок 4. DBSCAN або HDBSCAN об’єднував сусідні кадри у кластери.

Крок 5. Для кожного кластера визначався найбільш імовірний слайд.

Під час тестування DBSCAN працював швидше й стабільніше на коротких лекціях. Проблеми виникали в записах із великою кількістю анімованих переходів. У таких випадках алгоритм створював дрібні зайві кластери.

HDBSCAN краще згладжував нестабільні ділянки відео. Водночас час обробки зростав приблизно в 1.5–2 рази.

Для підбору параметрів проводився окремий експеримент на контрольному 5-хвилинному відео. Значення eps змінювалось у межах від 0.05 до 0.40, а min_samples – від 3 до 15.

Помилковим переходом вважався короткий сегмент тривалістю до 2 с або відхилення кількості отриманих кластерів від еталонної послідовності стабільних слайдів. Результати експерименту наведено в таблиці 2.6.

Таблиця 2.6 – Експериментальний вплив параметрів DBSCAN на кількість помилкових переходів

| **eps** | **min_samples** | **Кількість кластерів** | **Помилкові переходи** |
|---|---|---|---|
| 0.05 | 3 | 23 | 10 |
| 0.10 | 5 | 23 | 10 |
| 0.15 | 5 | 14 | 5 |
| 0.20 | 7 | 13 | 6 |
| 0.30 | 10 | 10 | 9 |
| 0.40 | 15 | 2 | 16 |

Найкращий результат для контрольного відео дав набір параметрів: eps = 0.15, min_samples = 5.

При менших значеннях eps алгоритм залишав у послідовності багато дрібних шумових переходів. При більших – надмірно об’єднував різні слайди в одну групу.

Для візуального аналізу векторних подань кадрів використовувалась UMAP-проєкція. Вона переводила багатовимірний простір векторних подань CLIP у двовимірне представлення без втрати локальної структури сусідства.

На рисунку 2.3 наведено приклад кластеризації кадрів лекції після UMAP-зменшення розмірності.

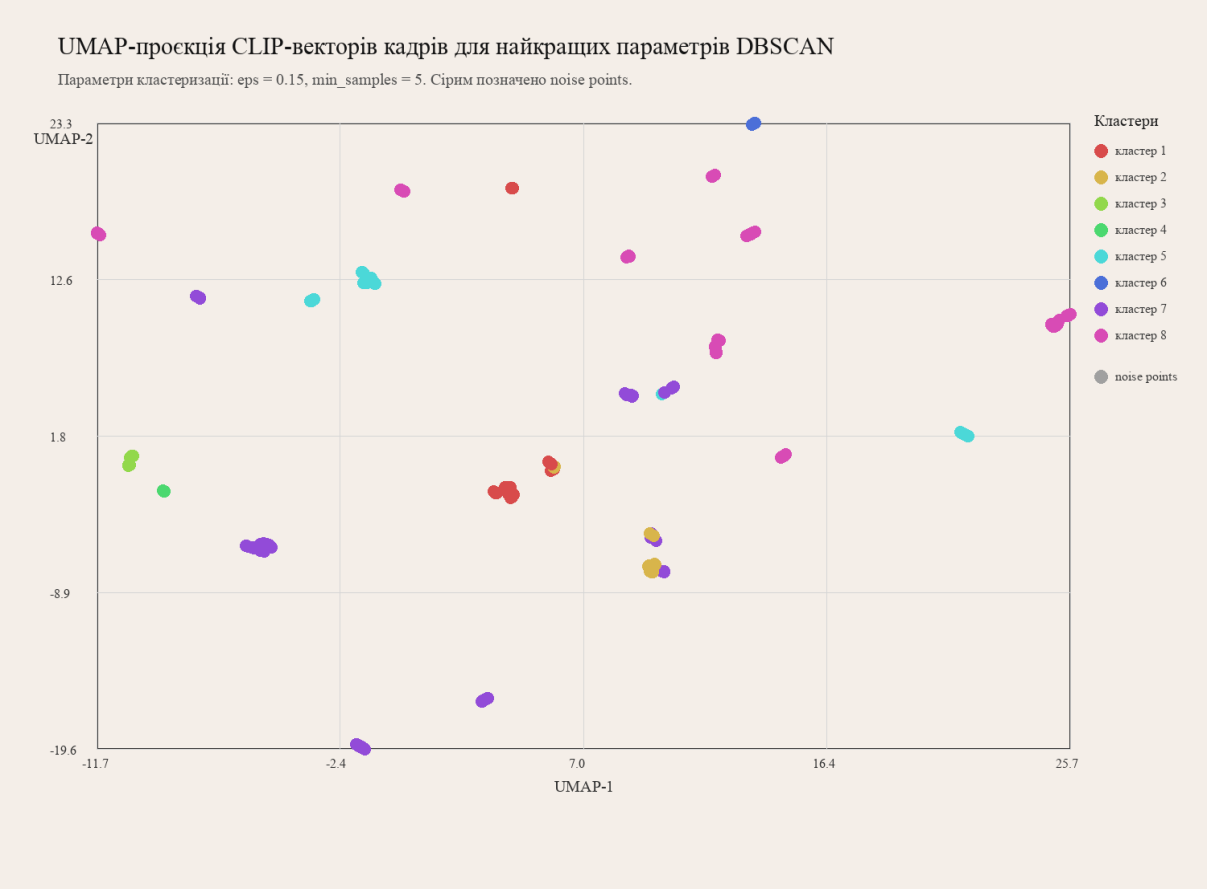


Рисунок 2.3 – UMAP-проєкція CLIP-векторів кадрів і результат кластеризації методом DBSCAN за параметрів eps = 0.15, min_samples = 5

На проєкції видно, що більшість кадрів формують компактні групи, які відповідають стабільним слайдам. Окремі віддалені точки відповідають

шумовим точкам і коротким перехідним станам між слайдами або фрагментам з анімаціями.

## 2.7 Аналіз великих мовних моделей для реферування

Після транскрипції лекції та синхронізації слайдів формується текстовий контекст для генерації конспекту. На цьому етапі якість результату вже залежить від великої мовної моделі: наскільки добре вона утримує довгий контекст, працює з технічними термінами й структурує матеріал.

У роботі тестувалися:

− GPT-4o;

− GPT-5.5;

− Claude Sonnet 4.6;

− Claude Opus 4.8;

− Gemini 3.5 Pro;

− Gemini 3.5 Flash;

− Llama 4 Maverick;

− Mistral Large 3;

− Qwen3.

Для порівняння використовувався той самий набір лекцій із розділу 2.2. Моделі отримували: транскрипцію, структуру слайдів, часові мітки та шаблон запиту.

У роботі Кобиліна Олега, Вечірської Ірини та Афанасьєва Анатолія було розглянуто обробку природної мови моделями глибинного навчання та порівняння їх результатів за такими метриками, як: F1-score, accuracy, recall та precision [43]. Проте для задачі автоматичного реферування доцільніше використовувати метрики ROUGE-1 (2.2), ROUGE-2 (2.3) та ROUGE-L (2.4), що обчислювалися за підходом Lin [34]. ROUGE-1 та ROUGE-2 оцінювали

перекриття однослівних і двослівних n-грам між згенерованим та еталонним конспектами, а ROUGE-L враховувала довжину найдовшої спільної підпослідовності:

$$ROUGE-1 = \frac{Overlapping\ n-grams}{Total\ n-grams\ in\ Generated\ Summary}, \quad (2.2)$$

$$ROUGE-2 = \frac{Overlapping\ n-grams}{Total\ n-grams\ in\ Referenced\ Summary}, \quad (2.3)$$

$$ROUGE-L = \frac{Longest\ Common\ Subsequence}{Total\ words\ in\ Referenced\ Summary}. \quad (2.4)$$

Додатково фіксувався: час відповіді, приблизна вартість обробки 60 хв лекції.

### 2.7.1 Модель GPT-4o і GPT-5.5

Модель GPT-4o показала найстабільніший результат на довгих лекціях. Модель добре працювала з фрагментами коду, таблицями та математичними поясненнями. Найменше помилок виникало під час структурування секцій конспекту.

Модель GPT-5.5 генерувала текст трохи повільніше. Різниця в ROUGE виявилась невеликою, але GPT-5.5 частіше створювала надто детальні абзаци замість коротких тез.

Обидві моделі добре працювали з українською мовою. Проблеми виникали переважно у змішаних лекціях, де викладач постійно переходив між українською та англійською.

Платою за якість залишилась вартість API. Для довгих лекцій витрати виявились помітно вищими за відкриті моделі.

Загалом, використання моделей сімейства GPT доцільно розглядати як еталонне рішення для обробки складних технічних дисциплін.

Їх варто застосовувати у випадках, коли пріоритетом є бездоганне збереження специфічної термінології, математичних формул та архітектури коду, а бюджет проєкту дозволяє покрити високі витрати на комерційні API.

### 2.7.2 Моделі Claude Sonnet 4.6 і Claude Opus 4.8

Модель Claude Sonnet 4.6 добре структурувала текст. Конспекти виходили читабельними навіть без додаткової післяобробки.

Модель Claude Opus 4.8 точніше працювала з довгими поясненнями й рідше втрачала контекст між секціями лекції. Різниця особливо помітна у 70–90-хвилинних відео.

Недолік Claude – обережна генерація. Модель іноді пропускала другорядні технічні деталі, якщо вони слабо повторювались у транскрипції.

Час відповіді виявився меншим, ніж у GPT-5.5 , але більшим за Gemini 3.5 Flash.

З огляду на ці особливості, сімейство Claude можна вважати оптимальним вибором для створення конспектів гуманітарних або теоретичних лекцій.

Їхня здатність генерувати високоякісний, структурований і літературно зв’язний текст зводить до мінімуму потребу в ручній редактурі, роблячи ці моделі чудовим інструментом для підготовки фінальних навчальних матеріалів.

### 2.7.3 Моделі Gemini 3.5 Pro і Gemini 3.5 Flash

Модель Gemini 3.5 Pro тестувалась через великий контекст. Модель стабільно приймала лекції обсягом понад 100 тисяч токенів без попереднього поділу на фрагменти.

Для мультимодальних запитів Gemini працювала краще за більшість конкурентів. PDF-слайди можна було передавати без окремого OCR-етапу.

Модель Gemini 3.5 Flash відповідала швидше. На коротких лекціях різниця сягала 2–3 разів у порівнянні з GPT-4o.

Недолік Flash-моделі – коротші й менш деталізовані конспекти. Іноді модель пропускала пояснення викладача між слайдами.

Завдяки таким характеристикам екосистема Gemini виявляється найбільш ефективною для комплексного аналізу мультимедійних матеріалів.

Можливість одночасного завантаження об'ємних відеолекцій разом із супровідними презентаціями без необхідності складного попереднього парсингу тексту робить їх незамінними для масової автоматизації обробки навчального контенту.

### 2.7.4 Моделі Llama 4 Maverick, Mistral Large 3 і Qwen3

Відкриті моделі тестувались локально через Ollama. Це дозволило оцінити можливість автономного локального запуску без зовнішнього API.

Модель Llama 4 Maverick працювала стабільно на англомовних лекціях, але гірше обробляла українську мову. Найчастіше помилки виникали у відмінках і складних технічних конструкціях.

Модель Mistral Large 3 генерувала коротші конспекти. У деяких випадках це навіть покращувало читабельність, але частина деталей втрачалась.

Модель Qwen3 показала найкращий результат серед моделей. Вона краще працювала з багатомовними лекціями й рідше втрачала структуру документа.

Перевага відкритих моделей – локальний запуск. Недолік – нижчий ROUGE у порівнянні з GPT-4o та Claude Opus 4.8.

Підсумовуючи, використання відкритих моделей залишається стратегічно важливим напрямком, незважаючи на їхнє певне відставання в метриках якості від комерційних лідерів.

Локальне розгортання (особливо Qwen3) є оптимальним рішенням для конфіденційних даних та роботи в ізольованих мережах. До того ж, відсутність

плати за токени API робить цей підхід вигіднішим при масовій обробці лекцій, а можливість їх подальшого донавчання дозволяє адаптувати генерацію під специфічну термінологію.

### 2.7.5 Порівняння моделей

Результати експериментального порівняння наведено в таблиці 2.7.

Таблиця 2.7 – Порівняння великих мовних моделей для генерації конспектів

| **Модель** | **ROUGE-1** | **ROUGE-2** | **ROUGE-L** | **Час відповіді** | **Вартість 60 хв лекції** |
|---|---|---|---|---|---|
| GPT-4o | 0.71 | 0.53 | 0.66 | 28 с | висока |
| GPT-5.5 | 0.72 | 0.54 | 0.67 | 39 с | висока |
| Claude Sonnet 4.6 | 0.69 | 0.51 | 0.64 | 31 с | висока |
| Claude Opus 4.8 | 0.73 | 0.56 | 0.69 | 34 с | висока |
| Gemini 3.5 Pro | 0.68 | 0.49 | 0.63 | 24 с | середня |
| Gemini 3.5 Flash | 0.63 | 0.44 | 0.58 | 12 с | низька |
| Llama 4 Maverick | 0.57 | 0.38 | 0.52 | 41 с | локальний запуск |
| Mistral Large 3 | 0.55 | 0.36 | 0.50 | 37 с | локальний запуск |
| Qwen3 | 0.61 | 0.42 | 0.57 | 33 с | локальний запуск |

Модель Claude Opus 4.8 дала найвищий ROUGE-L. GPT-4o працювала стабільніше на мультимодальних лекціях і краще структурувала великі конспекти. Gemini 3.5 Pro виграла за обсягом контексту. Qwen3 виявилась найсильнішою серед відкритих моделей.

# 3 РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯАВТОМАТИЧНОГО КОНСПЕКТУВАННЯ НАВЧАЛЬНИХ МАТЕРІАЛІВ

## 3.1 Вибір інструментарію для розробки застосунку

Для розробки системи автоматичного формування конспектів можна використати різні технологічні підходи: десктопний, мобільний, або веб-застосунок. Для того щоб охопити ширше коло користувачів було обрано саме веб версію. Таким чином користувач зможе працювати із системою не встановлюючи додаткового програмного забезпечення, а використовуючи браузер на будь-якій операційній системі, що фактично робить розроблене рішення кросплатформеним.

Вибір мов програмування та технологій для реалізації системи є вирішальним фактором як для продуктивності роботи застосунку, так і для зручності подальшої розробки та підтримки. Окрім цього, на вибір архітектури також вплинули технології, що використовуються у функціональній частині. Оскільки більшість інструментів для роботи з відео, аудіо, зображеннями та роботою з моделями штучного інтелекту мають найкращу підтримку в середовищі Python [44], серверна частина була розроблена на базі саме цієї мови програмування. Водночас для клієнтської частини було обрано TypeScript [45] та React [46], які дозволяють розробити інтерактивний інтерфейс із чіткою типізацією та зручною структурою компонентів.

Такий підхід дозволяє ефективно поєднати вебінтерфейс, обробку мультимедійних файлів, фонові задачі та генерацію навчальних матеріалів в єдиній системі.

### 3.1.1 Вибір платформи для розробки фронтенд частини

Для реалізації клієнтської частини застосунку було використано сучасні

технології: HTML5, CSS та JavaScript, а також бібліотеку React 18. React дає змогу побудувати інтерфейс, як набір окремих компонентів, таких як форму завантаження (рис. 3.1), індикатор прогресу (рис. 3.2), переглядач конспекту (рис. 3.3) тощо. Це значно спрощує розробку, підтримку та повторне використання елементів інтерфейсу.

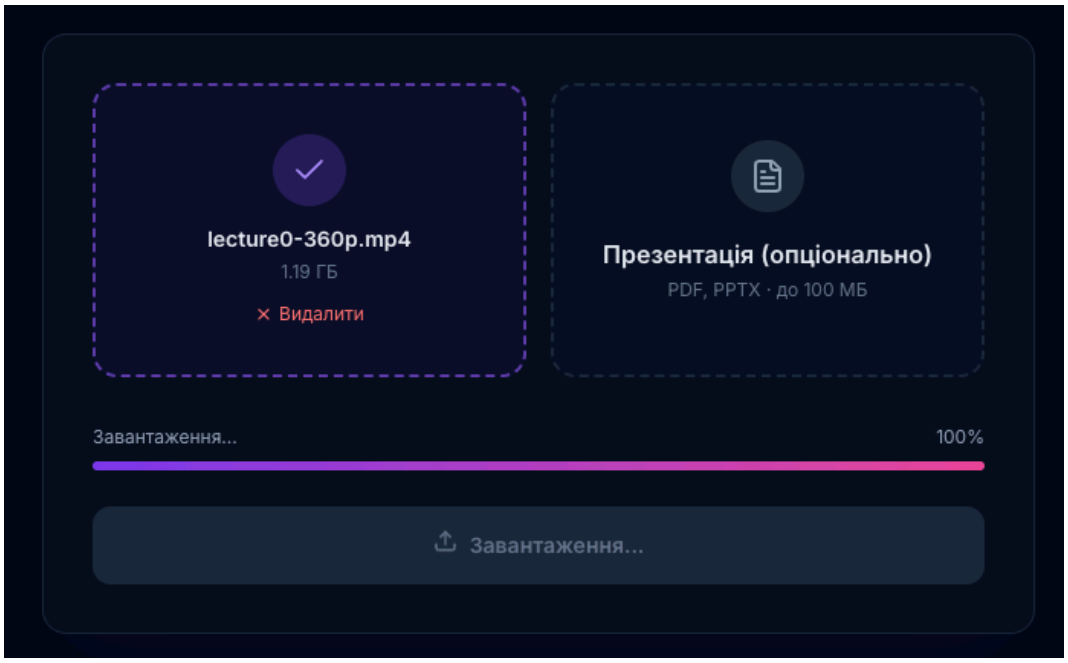


Рисунок 3.1 – Форма завантаження

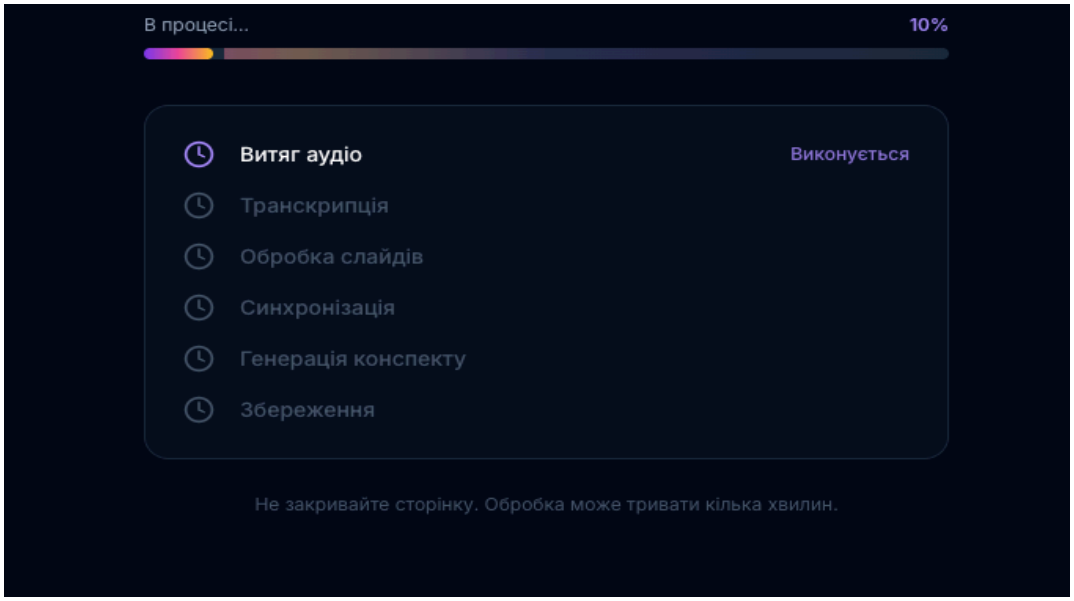


Рисунок 3.2 – Індикатор прогресу


Рисунок 3.3 – Переглядач конспекту

Для стилізації використано Tailwind CSS [47], завдяки якому інтерфейс адаптується під різні розміри екрана, а основні сторінки мають єдиний візуальний стиль.

Як мову програмування для frontend-частини було використано TypeScript, який є надбудовою над JavaScript із підтримкою статичної типізації, що підвищує надійність клієнтської частини, оскільки можливі помилки пов’язані з використанням неправильних типів даних можуть бути помічені ще на етапі розробки. Це особливо корисно під час роботи з відповідями API, статусами задач, даними транскрипції, слайдами та результатами генерації конспекту. Завдяки цьому покращується розробницький досвід та спрощується подальше розширення функціональності системи.

### 3.1.2 Вибір платформи для розробки бекенд частини

Backend-частину системи реалізовано мовою Python з використанням фреймворку FastAPI [48]. Він виконує роль основного API-шару застосунку, а саме: приймає HTTP-запити від frontend-частини, обробляє завантаження відео та презентації, повертає статуси задач, результати обробки, а також забезпечує експорт сформованого конспекту. FastAPI було обрано через підтримку асинхронних endpoint-ів, зручну побудову REST API та автоматичну генерацію OpenAPI-документації. Крім того, FastAPI добре інтегрується з Pydantic схемами, які використовуються для опису структури відповідей API та валідації даних.

Оскільки обробка лекції є довготривалою операцією, FastAPI не виконує її безпосередньо в межах HTTP-запиту. Після завантаження файлів endpoint створює запис задачі в базі даних, зберігає службову інформацію про відео та презентацію, після чого передає ідентифікатор задачі у фонову чергу. Для цього в системі використовується Celery [49], який відповідає за запуск окремого worker-процесу. Саме worker виконує основні етапи обробки.

Для передавання задач між FastAPI та Celery використовується Redis [50]. У цій архітектурі Redis виконує роль проміжного шару між API та worker-процесом. FastAPI додає повідомлення про нову задачу в чергу, а Celery-worker

отримує його та починає виконання. Такий підхід дозволяє не блокувати HTTP-запит користувача, а сама обробка триває у фоновому режимі.

Для збереження стану задачі використовується PostgreSQL [51]. У базі даних зберігаються статус обробки, прогрес, повідомлення про помилки, дані транскрипції, інформація про синхронізовані слайди та сформований конспект. Завдяки цьому користувач може оновити сторінку або повернутися до результату пізніше, оскільки стан задачі не залежить від активності браузера.

Робота з базою даних у backend-частині реалізована за допомогою SQLAlchemy [52]. Ця бібліотека виконує роль ORM-шару, тобто дозволяє описувати таблиці бази даних як Python-класи та працювати з ними через об’єкти. У межах проєкту через SQLAlchemy описано сутності задачі, транскрипції, синхронізації слайдів і конспекту. Для керування змінами структури бази даних використовується Alembic [53], який дає змогу створювати та застосовувати міграції під час розвитку системи.

ML-частина системи відповідає за автоматичну обробку лекційних матеріалів після їх завантаження користувачем. Вона охоплює послідовні етапи роботи з відео, аудіо, презентацією та візуальними ознаками слайдів. Спочатку система отримує аудіодоріжку з відеофайлу, після чого виконує транскрипцію мовлення з часовими мітками. Окремо обробляється презентація: з неї вилучається текстовий вміст і створюються зображення слайдів, які надалі використовуються для зіставлення з кадрами відео. На завершальному етапі система порівнює візуальні ознаки слайдів і відеокадрів, щоб визначити, у який момент лекції демонструвався кожен слайд. Інструменти, що використовуються для реалізації цих етапів, наведено в таблиці 3.1.

Таблиця 3.1 – Інструменти обробки навчальних матеріалів

| **Етап** | **Інструмент** | **Результат** |
|---|---|---|
| Аудіо | FFmpeg | WAV файл |
| Транскрипція | Faster-whisper | Текст з сегментами |

Продовження таблиці 3.1

| Етап | Інструмент | Результат |
|---|---|---|
| Слайди | PyMuPDF, python-pptx, Tesseract | Текст і PNG зображення |
| Синхронізація | CLIP-ViT-B/32 | Прив’язка слайд-час |
| Конспект | Claude Sonnet 4.6 | Markdown документ |
| Експорт | ReportLab, python-docx | PDF і DOCX |

## 3.2 Проєктування застосунку

### 3.2.1 Опис функціональності та бізнес-кейсів застосунку

У застосунку основним актором системи є користувач – людина, яка має відеозапис лекції та хоче отримати структурований текстовий конспект без ручного перегляду всього матеріалу. Діаграма варіантів використання, наведена на рисунку 3.4, демонструє основні сценарії взаємодії користувача із системою автоматичного конспектування лекцій.

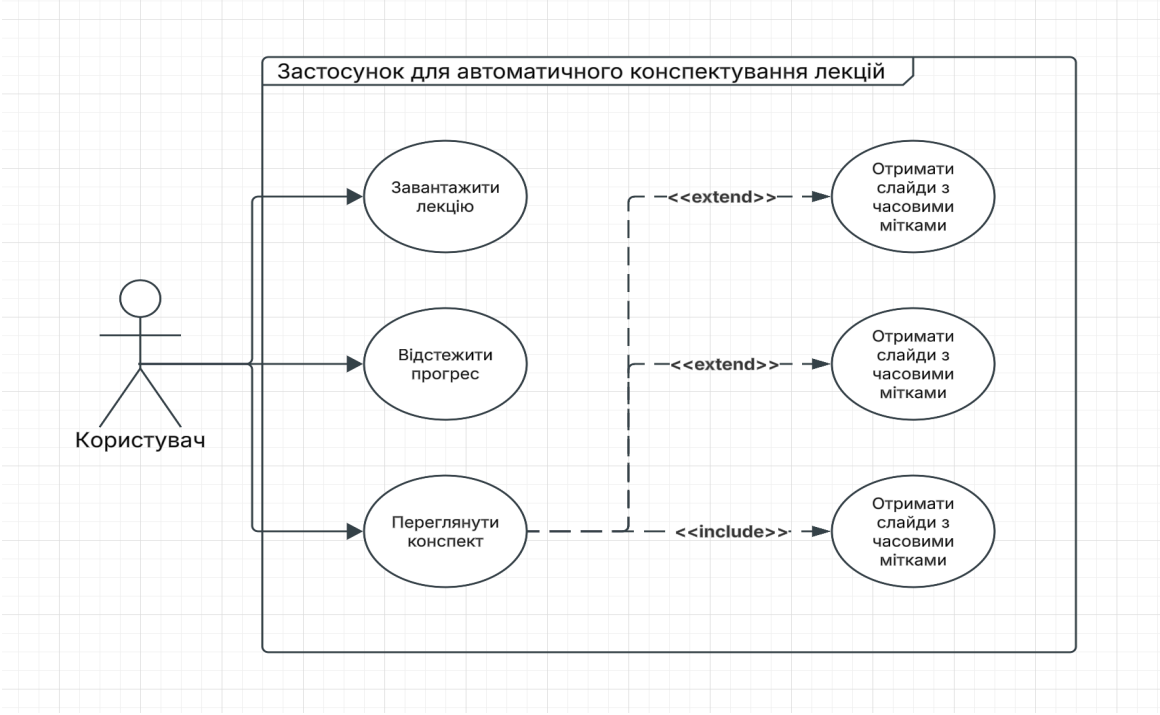


Рисунок 3.4 – Варіанти використання застосунку користувачем

Першим сценарієм є завантаження лекції. У межах цього варіанту використання користувач передає до системи відеофайл лекції, а за потреби також додає файл презентації. Відео є обов’язковим вхідним матеріалом, оскільки саме з нього система отримує аудіодоріжку та виконує транскрипцію в той час як презентація є додатковим матеріалом, тому сценарій отримання слайдів із часовими мітками позначається як розширення основного сценарію.

Другим сценарієм є відстеження прогресу обробки. Після завантаження матеріалів користувач не очікує завершення операції в межах одного HTTP-запиту, оскільки обробка лекції може тривати декілька хвилин. Замість цього система створює фонову задачу, а користувач бачить поточний стан виконання завдяки тому, що прогрес відображає основні етапи обробки та їх статус.

Третім сценарієм є перегляд сформованого конспекту. Після завершення обробки користувач переходить до результату, де може ознайомитися зі структурованим текстом лекції. Конспект містить ключові поняття, основний матеріал, підрозділи за темами та висновки. Якщо під час завантаження було додано презентацію, перегляд конспекту включає також роботу зі слайдами, прив’язаними до часових інтервалів відео.

### 3.2.2 Проєктування бекенд API

На вищому рівні backend-частини застосунку існує сервер FastAPI, який приймає запити від клієнтської частини, виконує первинну валідацію даних і передає керування відповідним сервісам системи. Backend не лише обробляє звичайні HTTP-запити, а й підтримує WebSocket-з’єднання для оновлення стану задачі в реальному часі. Основні endpoint-и серверної частини мають префікс /api, а кореневий шлях використовується для базової перевірки доступності сервера.

На сервері реалізовано такі відкриті вхідні точки:

− GET /api/health – повертає службовий статус системи. Цей endpoint використовується для перевірки працездатності API;

− GET / – повертає коротку інформацію про API та посилання на автоматично згенеровану документацію. Використовується для базової перевірки того, що backend-сервер запущений;

− POST /api/upload – завантажує матеріали лекції до системи. Приймає відеофайл лекції як обов’язковий параметр, а також презентацію як необов’язковий файл. Після отримання файлів endpoint створює задачу обробки, зберігає файли у сховищі, записує початкову інформацію в базу даних і передає ідентифікатор задачі у Celery для фонового виконання;

− GET /api/tasks/{task_id}/status – повертає поточний статус задачі за її ідентифікатором. Відповідь містить стан обробки, відсоток прогресу та повідомлення про помилку, якщо вона виникла. Цей endpoint використовується frontend-частиною для періодичної перевірки стану задачі;

− GET /api/tasks/{task_id}/result – повертає результат обробки лекції після завершення задачі. У відповідь входять дані транскрипції, сформований конспект, інформація про слайди та часові мітки, якщо під час завантаження була додана презентація;

− PATCH /api/tasks/{task_id}/notes – оновлює текст сформованого конспекту. Цей endpoint використовується після редагування конспекту користувачем у вебінтерфейсі;

− GET /api/tasks/{task_id}/export/{format} – експортує сформований конспект у файл. Параметр format визначає формат експорту та може набувати значень pdf або docx;

− WebSocket /api/ws/tasks/{task_id} – забезпечує отримання оновлень про стан задачі в реальному часі. Завдяки цьому користувач може бачити зміну етапів обробки без ручного оновлення сторінки.

Так як frontend-частина застосунку реалізована у вигляді вебінтерфейсу на React, у ній також використовується система маршрутизації. Вона відповідає за перехід між основними екранами взаємодії користувача із системою:

− / – початкова сторінка застосунку, на якій користувач завантажує відео лекції та за потреби додає презентацію;

– /tasks/:taskId – сторінка відстеження прогресу обробки. На ній відображаються поточний статус задачі, відсоток виконання та етап, який система виконує в цей момент;

– /tasks/:taskId/result – сторінка перегляду результату. На ній користувач бачить сформований конспект, синхронізовані слайди, може редагувати текст і експортувати результат у PDF або DOCX.

На рисунку 3.5 наведено діаграму взаємодії компонентів системи та користувача із застосунком автоматичного конспектування лекцій.

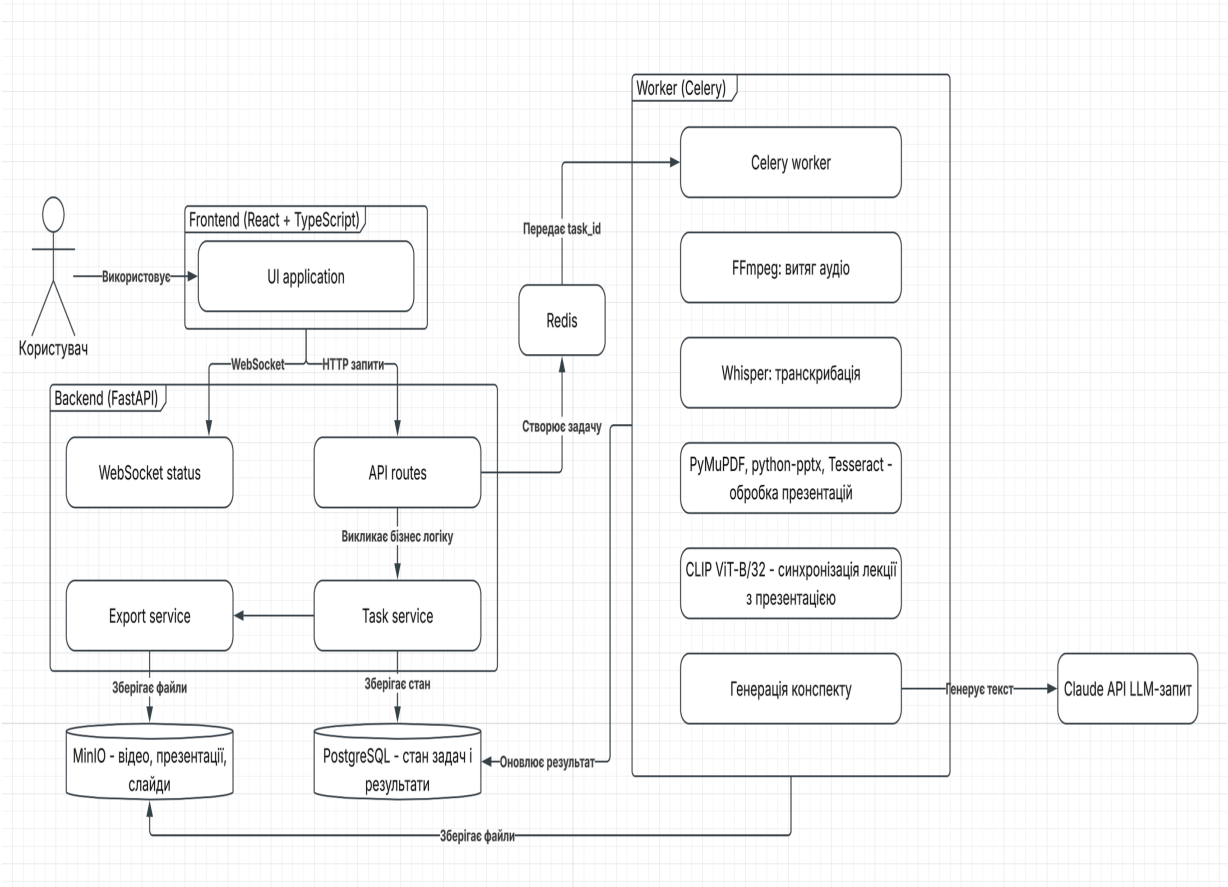


Рисунок 3.5 – Діаграма взаємодії компонентів системи

### 3.2.1 Проєктування бази даних

База даних системи містить чотири основні таблиці: tasks, transcripts, slide_syncs і notes, структура яких наведена на рисунку 3.6.

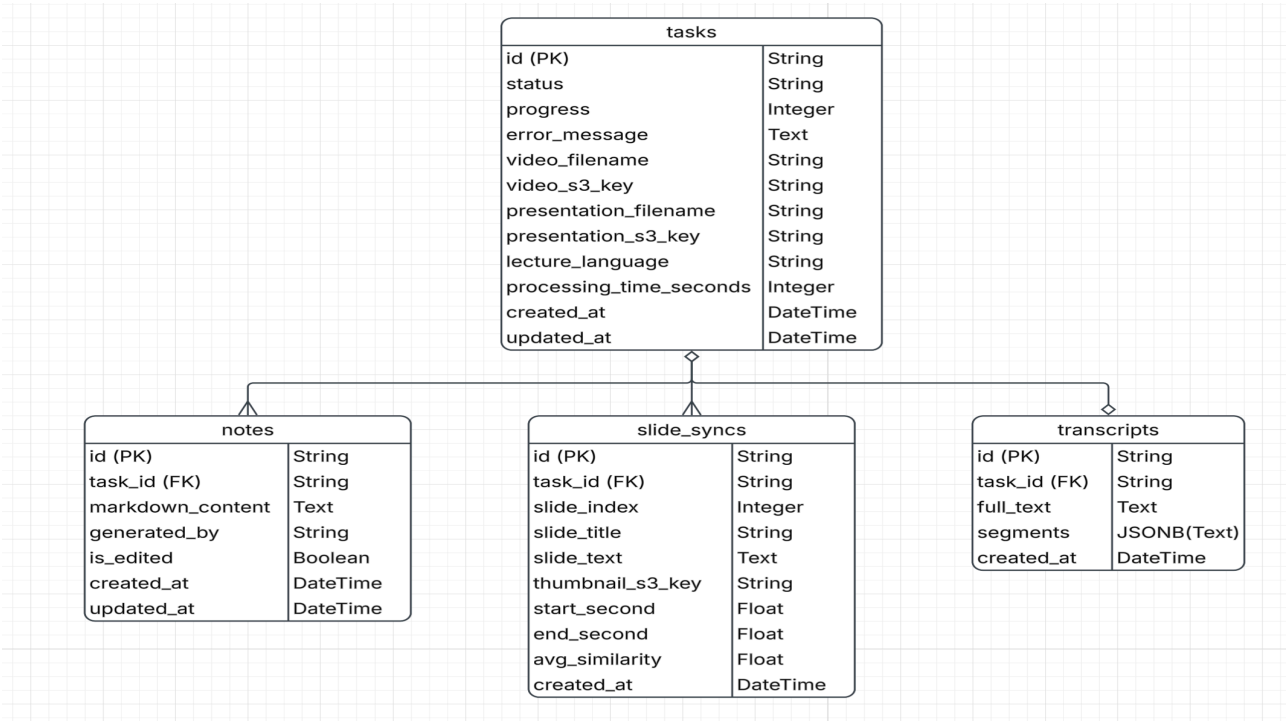


Рисунок 3.6 – Основні таблиці бази даних

Центральною таблицею є tasks. Вона описує окремий запуск обробки лекції. У ній зберігаються поточний статус задачі, відсоток прогресу, повідомлення про помилку, назви завантажених файлів, ключі об’єктів у MinIO, обрана LLM-модель, мова лекції та час виконання обробки.

Таблиця transcripts призначена для збереження результатів розпізнавання мовлення. Вона містить повний текст лекції, а також масив сегментів у форматі JSONB. Кожен сегмент може містити текстову частину та часові межі, що дозволяє пов’язувати зміст лекції з конкретними фрагментами відео.

Таблиця slide_syncs використовується тоді, коли користувач завантажує презентацію. У ній зберігаються номер слайду, його заголовок, вилучений текст, початок і кінець показу у відео, середня подібність під час зіставлення та ключ PNG-ескізу в MinIO.

Таблиця notes містить результат роботи LLM-модуля, тобто згенерований Markdown-конспект лекції. Також у ній зберігається ручне редагування та час оновлення.

Зв’язки між таблицями побудовані навколо tasks: для транскрипції та конспекту використано зв’язок один-до-одного, а для слайдів – один-до-багатьох.

Це пов’язано з тим, що одна задача має один фінальний текст транскрипції та один основний конспект, але може містити багато синхронізованих слайдів. Така структура забезпечує зручне збереження результатів обробки та дозволяє надалі розширювати систему без суттєвої зміни базової моделі даних.

### 3.2.2 Організація файлового сховища

Файли великого розміру не зберігаються безпосередньо в PostgreSQL, оскільки база даних у цій системі використовується насамперед для структурованих даних. Для збереження самих файлів застосовується MinIO, який виконує роль об’єктного сховища. Схема взаємодії між відео, презентацією, MinIO та записами в базі даних наведена на рисунку 3.7

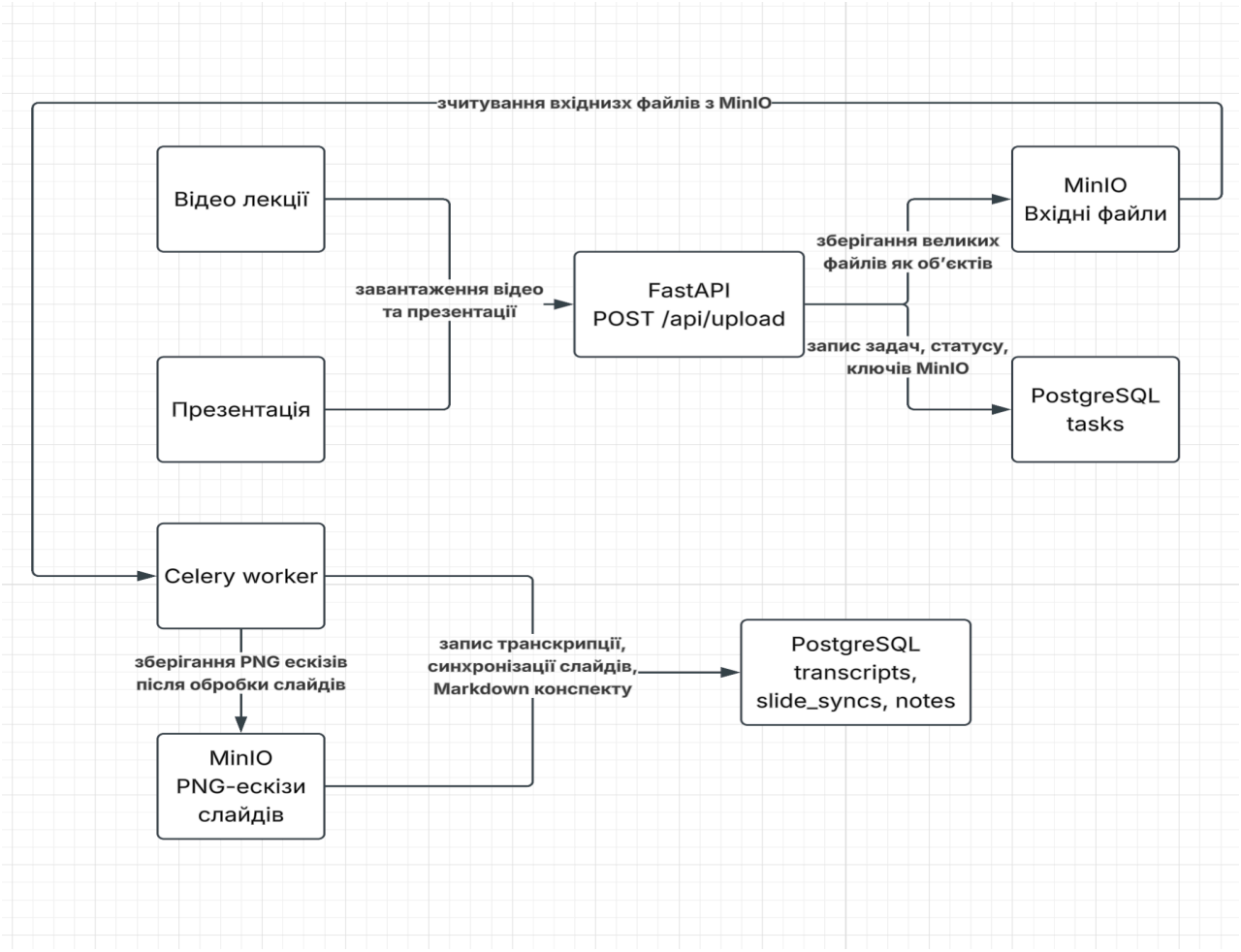


Рисунок 3.7 – Потік збереження файлів і метаданих

Під час завантаження лекції backend приймає відеофайл і, за наявності, файл презентації. Після цього файли передаються до MinIO, а в PostgreSQL зберігаються не самі бінарні дані, а лише службові ключі об’єктів. Наприклад, для відео формується ключ виду videos/{task_id}/{filename}, а для презентації – slides/{task_id}/{filename}. Такий підхід зменшує навантаження на базу даних, спрощує роботу з великими файлами та дозволяє backend-частині швидко знаходити потрібні об’єкти за їхніми ключами.

Після обробки презентації система створює PNG-зображення окремих слайдів. Ці зображення також завантажуються до MinIO з ключами виду slides/{task_id}/slide_{index}.png.

Для показу слайдів користувачу backend не передає облікові дані MinIO у браузер. Замість цього він створює тимчасові URL для потрібних зображень. Такий URL дозволяє frontend-частині отримати конкретний файл протягом обмеженого часу, не відкриваючи прямий доступ до сховища.

### 3.2.3 Проєктування станів задачі

Оскільки обробка лекції складається з декількох тривалих етапів, у системі введено статусну модель задачі. Вона дозволяє відображати користувачу конкретний поточний етап роботи pipeline (рис. 3.8). Після завантаження файлів endpoint POST /api/upload створює задачу, зберігає вхідні матеріали та передає її ідентифікатор у фонову чергу. Далі виконання переходить до Celery-worker, який послідовно оновлює статус задачі після початку кожного етапу.

У межах пайплайну послідовно виконується підготовка даних (вилучення аудіо, транскрибація, парсинг слайдів), їх синхронізація та генерація фінального конспекту за допомогою LLM. Наприкінці алгоритм зберігає отримані файли та позначає задачу як виконану.

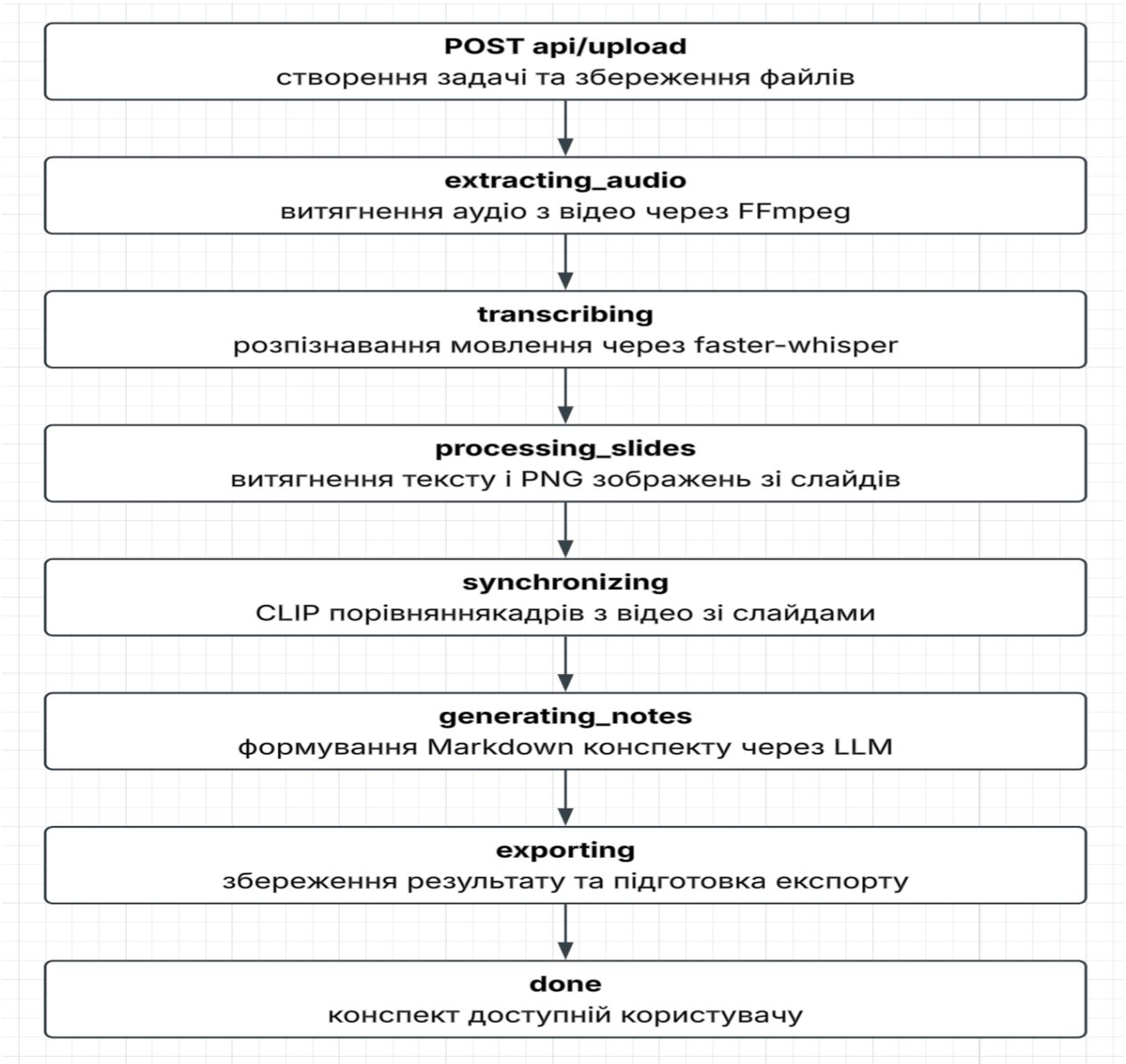


Рисунок 3.8 – Послідовність етапів конвеєра

За рахунок того, що актуальний статус зберігається в базі даних, інформація про поточний стан не втрачається після оновлення сторінки або перезапуску клієнтської частини застосунку. Якщо під час виконання конвеєра виникає виняток, задача переходить у стан помилки, а текст помилки записується в поле error_message. Завдяки цьому клієнтська частина може показати користувачу причину невдалого завершення, а не просто нескінченне очікування.

Головний конвеєр обробки реалізовано у функції process_lecture. Під час виконання створюється тимчасова директорія, до якої завантажується вхідне відео та, за наявності, презентація. Після цього задача проходить послідовні етапи: витягування аудіо, транскрипцію мовлення, обробку слайдів,

синхронізацію слайдів із відео, генерацію конспекту та збереження фінального результату.

Окремо в системі передбачено відображення прогресу виконання. Кожному етапу відповідає приблизна частка загального часу обробки, що дає змогу показувати користувачу не лише назву поточного стану, а й орієнтовний відсоток завершення задачі. Приблизний розподіл часу між етапами наведено на рисунку 3.9. Найбільшу частку займає транскрипція, оскільки розпізнавання мовлення потребує послідовної обробки аудіо.

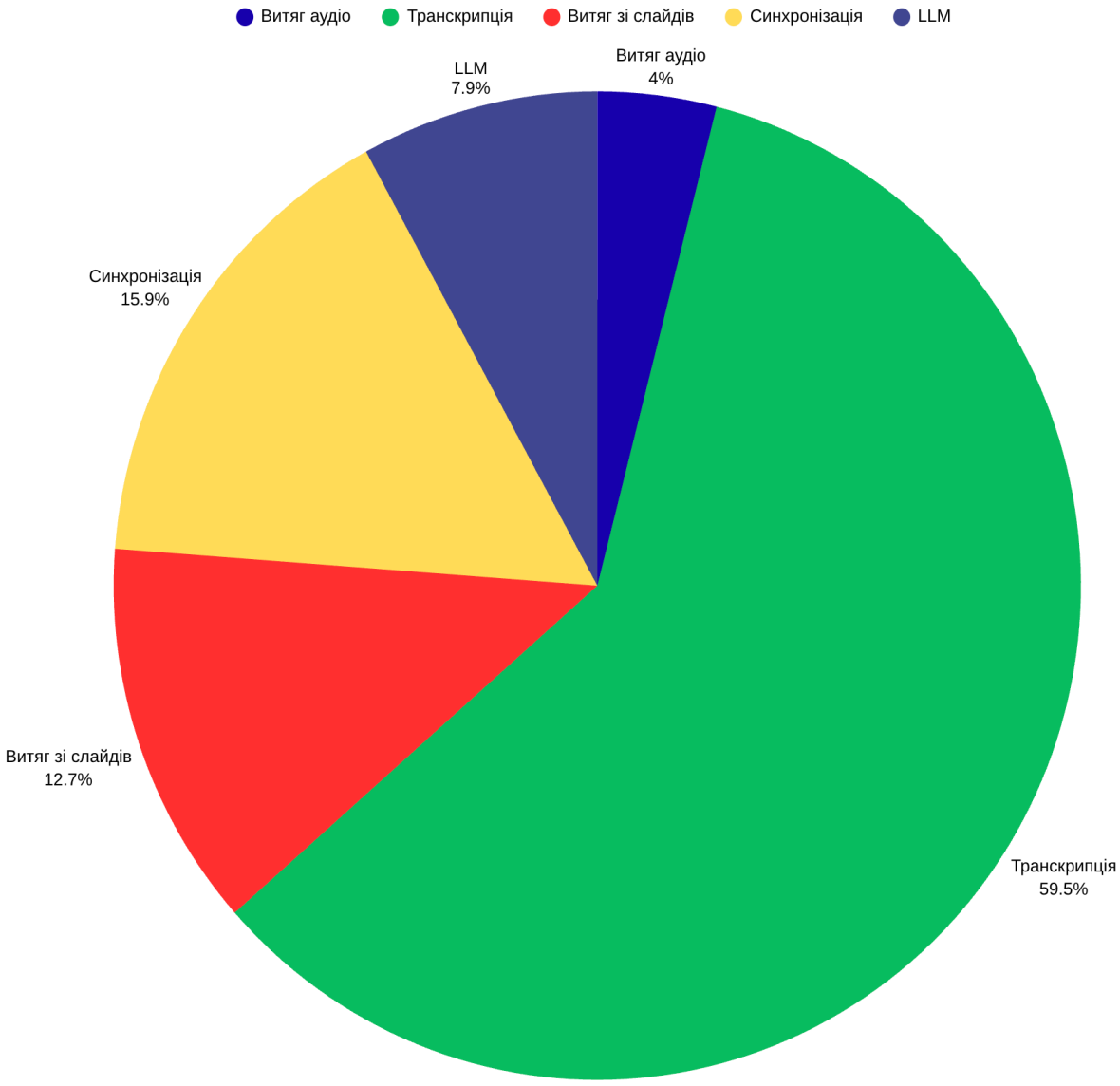


Рисунок 3.9 – Орієнтований розподіл часу між етапами обробки

Таким чином, статусна модель задачі виконує одразу кілька функцій: забезпечує прозорість роботи системи для користувача, дозволяє frontend-частині відображати прогрес у реальному часі, спрощує діагностику помилок і робить pipeline стійкішим до переривання взаємодії з браузером.

### 3.2.4 Етап обробки аудіо та транскрипції

На етапі витягування аудіо в розробленій системі використовується FFmpeg. У межах pipeline вхідний відеофайл приводиться до формату WAV із частотою дискретизації 16 кГц та одним аудіоканалом. Така уніфікація спрощує роботу pipeline та зменшує ймовірність помилок.

Для розпізнавання мовлення в системі використано бібліотеку faster-whisper та модель Whisper large-v3-turbo. Саме ця модель застосовується як основний інструмент транскрипції лекцій, оскільки вона забезпечує баланс між якістю розпізнавання та швидкістю роботи. У backend-частині транскрипція винесена в окремий сервіс TranscriptionService, який завантажує модель один раз у межах worker-процесу. Це зроблено тому, що ініціалізація моделі є ресурсомісткою операцією, і повторне завантаження перед кожною задачею суттєво збільшувало б загальний час обробки.

У результаті роботи TranscriptionService система отримує визначену мову лекції та повний текст транскрипції зі списком. Для сегментів зберігаються часові мітки на рівні слів, що дозволяє точніше аналізувати відповідність тексту фрагментам аудіо. Отримана транскрипція записується до бази даних і надалі використовується як основний текстовий контекст для генерації конспекту.

Така оптимізація процесу гарантує стабільну роботу системи без зайвих затримок. Збережений структурований текст із часовими мітками стає надійним фундаментом для наступного кроку.

### 3.2.5 Етап обробки відео та презентацій

Після завантаження файлу презентації модуль slides.py визначає його формат за розширенням і застосовує відповідний спосіб обробки.

Якщо користувач завантажив PDF-файл, кожна сторінка презентації рендериться у PNG-зображення за допомогою PyMuPDF, а текст слайда вилучається з текстового шару документа. Якщо текстовий шар відсутній або містить замало даних, система додатково використовує Tesseract OCR з мовами ukr+eng, що дозволяє працювати зі сканованими або зображеними слайдами.

Якщо ж користувач завантажив PPTX-файл, використовується бібліотека python-pptx. Вона читає структуру слайдів і отримати з них текстові блоки. Система також створює PNG-зображення кожного слайда для подальшого порівняння з відео.

Після підготовки слайдів система переходить до їх синхронізації з відеозаписом лекції. Для цього в проєкті використовується модель CLIP ViT-B/32, яка перетворює як PNG-зображення слайдів, так і кадри відео у вектори ознак однакової розмірності. Кадри відео вибираються з певною частотою, після чого для кожного з них обчислюється embedding. Аналогічно embedding створюється для кожного слайда. Після нормалізації векторів система обчислює матрицю косинусної подібності між кадрами відео та слайдами презентації.

Для кожного відеокадру обирається слайд із найбільшою подібністю. Далі ці результати агрегуються у часові інтервали, які показують, у який проміжок лекції демонструвався конкретний слайд. Отримані дані зберігаються в таблиці slide_syncs і надалі використовуються для відображення слайдів поруч із конспектом.

Наостанок система фільтрує отримані часові інтервали, усуваючи короткочасні хибні спрацьовування. Після очищення тимчасових файлів готові синхронізовані дані передаються на наступний етап — обробку аудіодоріжки та генерацію текстового конспекту.

### 3.2.6 Етап формування контексту для LLM

Після завершення транскрипції та синхронізації слайдів система формує текстовий контекст для мовної моделі. У розробленому застосунку для генерації конспекту використовується модель Claude Sonnet 4.6. До неї передається повний текст лекції, а також за наявності інформація про слайди у структурованому вигляді. Якщо слайди були синхронізовані з відео, до запиту додаються їхні номери, часові межі показу, заголовки та вилучений текст, завдяки якому модель отримує не лише суцільну транскрипцію, а й додаткову структуру лекції, яка допомагає логічно поділити майбутній конспект на тематичні частини.

Для довгих лекцій у LLM-модулі передбачено механізм chunking. Перед викликом, система приблизно оцінює обсяг промпта. Якщо він перевищує задану межу, транскрипція розбивається на менші частини. Для кожної частини окремо генерується фрагмент конспекту, після чого ці фрагменти передаються моделі повторно для об’єднання в один цілісний документ. Такий підхід дозволяє працювати з довгими лекціями, не перевищуючи обмеження контекстного вікна мовної моделі.

### 3.2.7 Етап експорту результатів

Готовий конспект зберігається у форматі Markdown, що є зручним для редагування, збереження заголовків, списків, фрагментів коду та подальшого перетворення в інші документи.

Для експорту у PDF використовується ReportLab, що дає змогу генерувати документ на backend-стороні без залежності від браузера користувача. Для DOCX використовується python-docx, який створює файл Word із базовою структурою заголовків і списків.

Експорт виконується з актуального тексту конспекту, тому якщо користувач відредагував Markdown, завантажений PDF або DOCX міститиме вже оновлену версію.

## 3.3 Ілюстрація роботи

### 3.3.1 Запуск застосунку на локальній машині

Для запуску застосунку на локальній машині на UNIX системі необхідно виконати ряд дій. Для початку треба завантажити Docker, Docker Compose, Node.js та Python 3.11. Після цього треба запустити інфраструктурні сервіси з кореневої папки проєкту командою docker compose up -d postgres redis minio. Наступним кроком треба створити та активувати віртуальне середовище Python в директорії backend та завантажити всі необхідні залежності:

Крок 1. Відкрити термінал комп’ютера.

Крок 2. cd backend.

Крок 3. python3 -m venv .venv.

Крок 4. source .venv/bin/activate.

Крок 5. pip install -r requirements.txt.

Крок 6. alembic upgrade head.

Після того як віртуальне середовище було активовано та залежності завантажені, треба запустити backend сервер командою uvicorn app.main:app --host 0.0.0.0 --port 8000. Наступним кроком в окремому вікні терміналу необхідно запустити Celery:

Крок 1. Відкрити нове вікно термінала комп’ютера.

Крок 2. cd backend.

Крок 3. source .venv/bin/activate.

Крок 4. celery -A app.workers.celery_app worker --loglevel=info -Q default --concurrency=1.

Останнім кроком треба запустити frontend частину:

Крок 1. Відкрити нове вікно термінала комп’ютера.

Крок 2. cd frontend.

Крок 3. npm install.

Крок 4. npm run dev -- --host 0.0.0.0 –port 3000.

### 3.3.2 Інтерфейс користувача розробленого застосунку

Робота із застосунком починається з відкриття головної сторінки (рис. 3.10) за адресою http://localhost:3000/. Користувач бачить форму створення нової лекції, де передбачені окремі області для відеозапису та презентації.

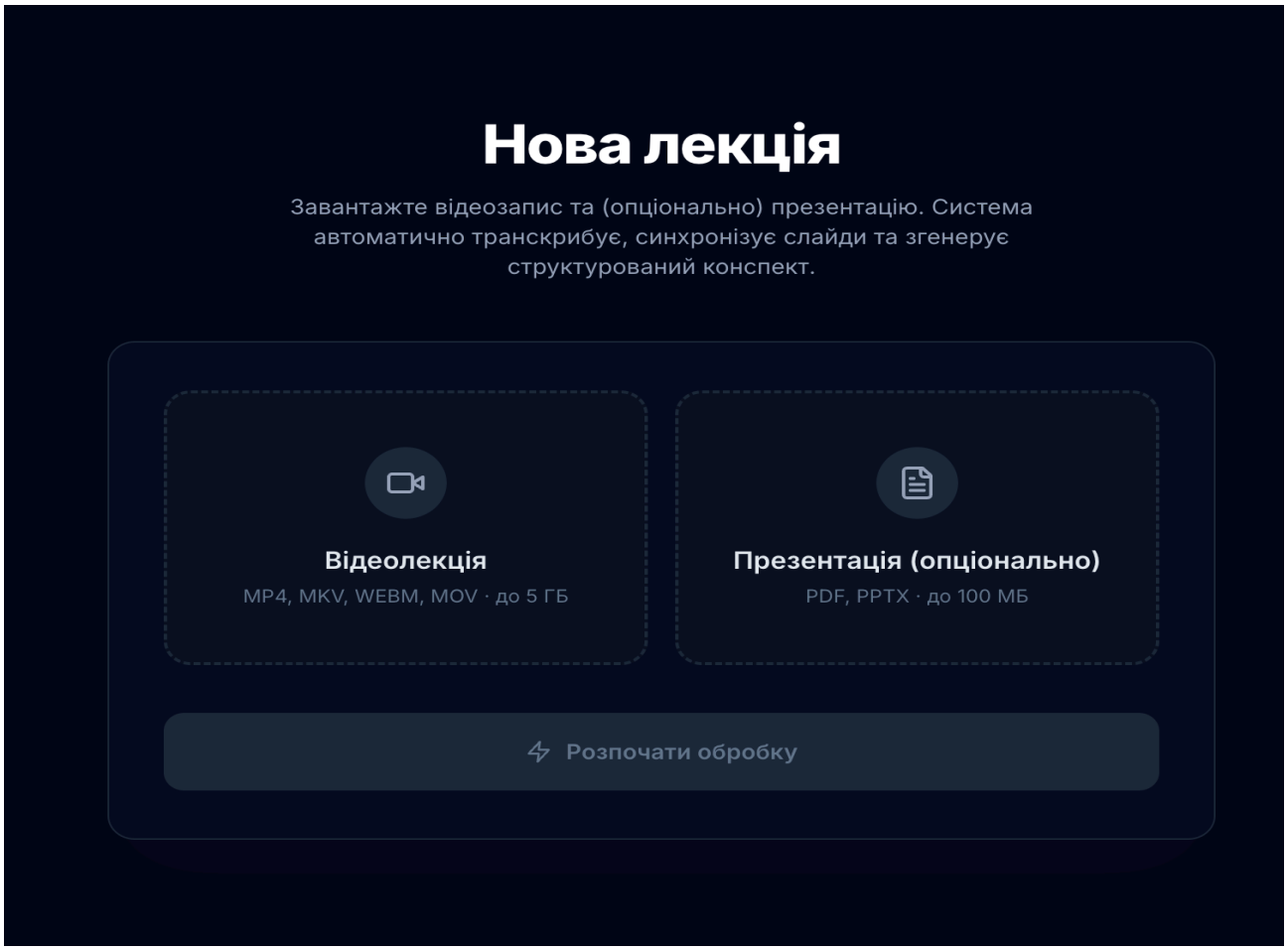


Рисунок 3.10 – Головна сторінка застосунку

На цьому етапі фронтенд виконує первинну перевірку типів файлів і розміру. Для відео підтримуються формати файлів MP4, MKV, WEBM і MOV, а для презентації – PDF і PPTX.

Після вибору файлів кнопка запуску стає активною. Під час завантаження відображається прогрес передачі даних на сервер.

Коли дані завантажились на сервер та закінчився процес створення задачі, користувач переходить на сторінку прогресу (рис. 3.11). Вона показує task_id, загальний відсоток виконання та список етапів pipeline.

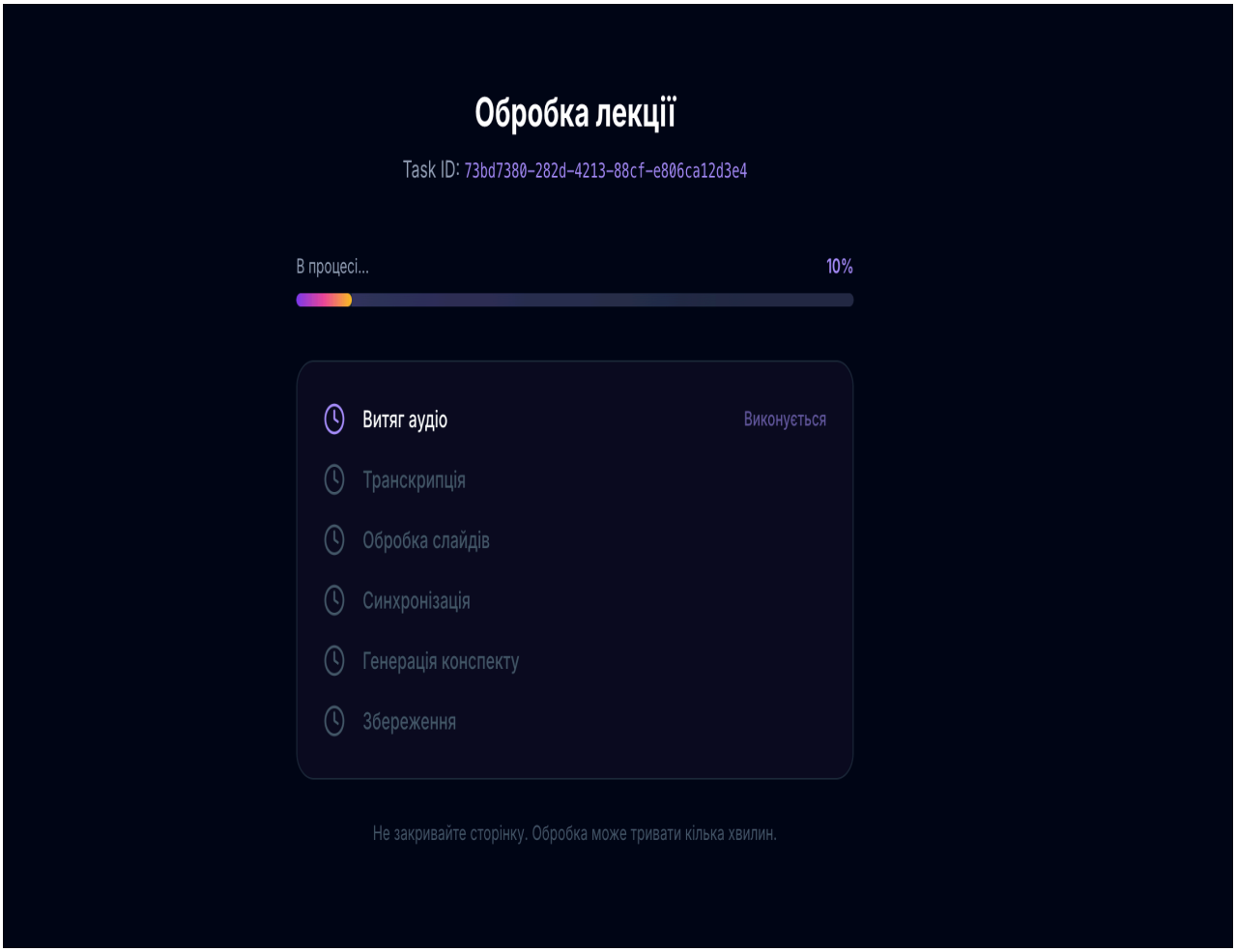


Рисунок 3.11 – Сторінка прогресу

Користувач бачить, який етап уже завершено, який виконується зараз і які етапи ще залишилися до виконання, а після переходу задачі у стан `done` сторінка прогресу автоматично перенаправляє користувача на сторінку результату (рис. 3.12).

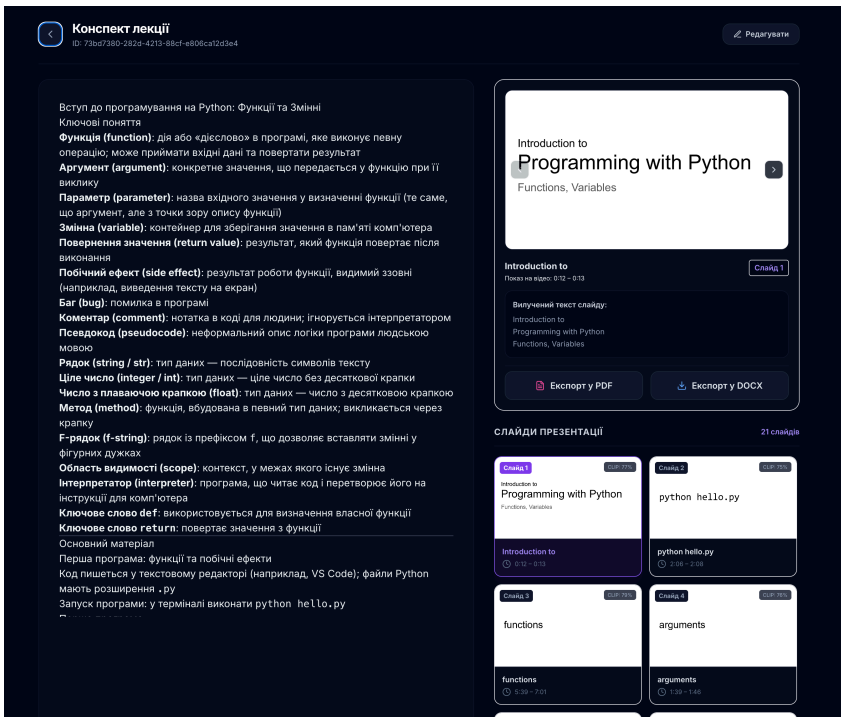


Рисунок 3.12 – Сторінка результату

Ліва частина цієї сторінки відведена під конспект, а права – під інформацію про слайди. На сторінці доступний режим редагування, який дозволяє швидко виправити неточності LLM-моделі або адаптувати конспект під власний стиль. Права панель містить часові межі показу слайда, витягнутий текст і кнопки переходу між слайдами. Також у правій частині можна знайти кнопки, що дозволяють завантажити конспект у форматі PDF або DOCX.

### 3.3.3 Адаптивність інтерфейсу

Оскільки застосунок було реалізовано у вигляді вебінтерфейсу, було важливо забезпечити коректне відображення не лише на екранах комп’ютера, а й на вузьких екранах мобільних телефонів. Для адаптивних сіток та зміни розташування компонентів залежно від ширини робочої області браузера було використано Tailwind CSS. На мобільних екранах горизонтальні інтерфейси були перебудовані в вертикальні (рис. 3.13, рис. 3.14, рис. 3.15), що робить застосунок універсальним.

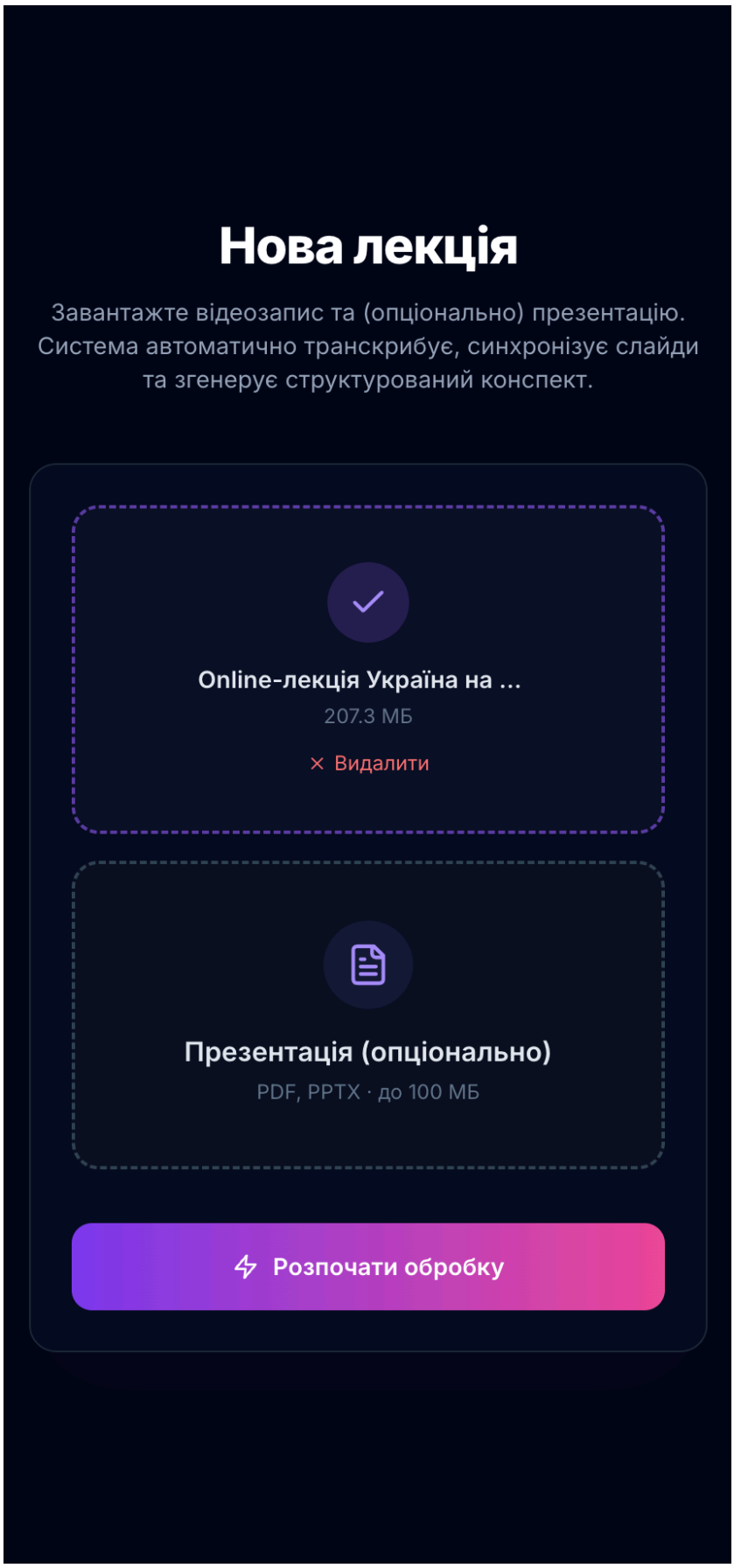


Рисунок 3.13 – Головна сторінка на телефоні

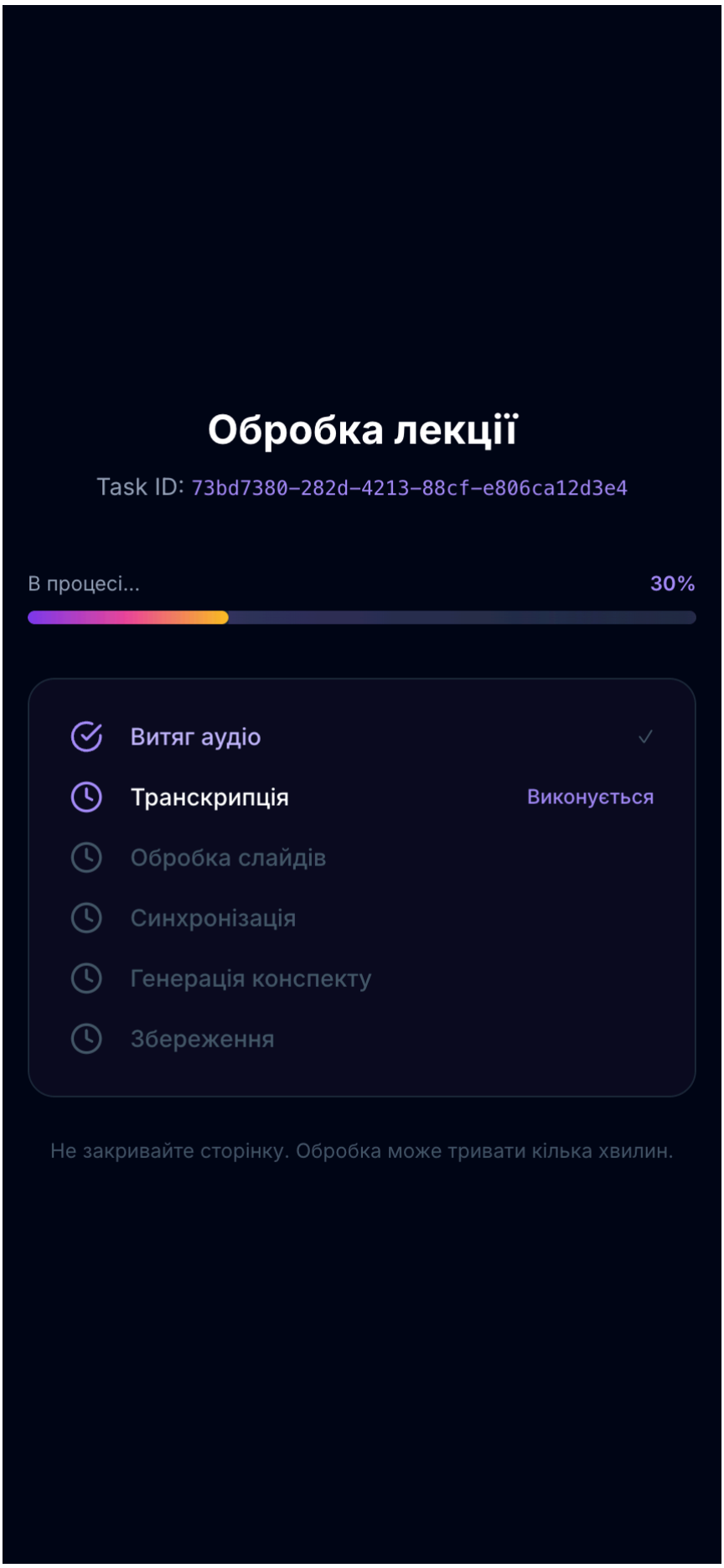


Рисунок 3.14 – Сторінка прогресу на телефоні

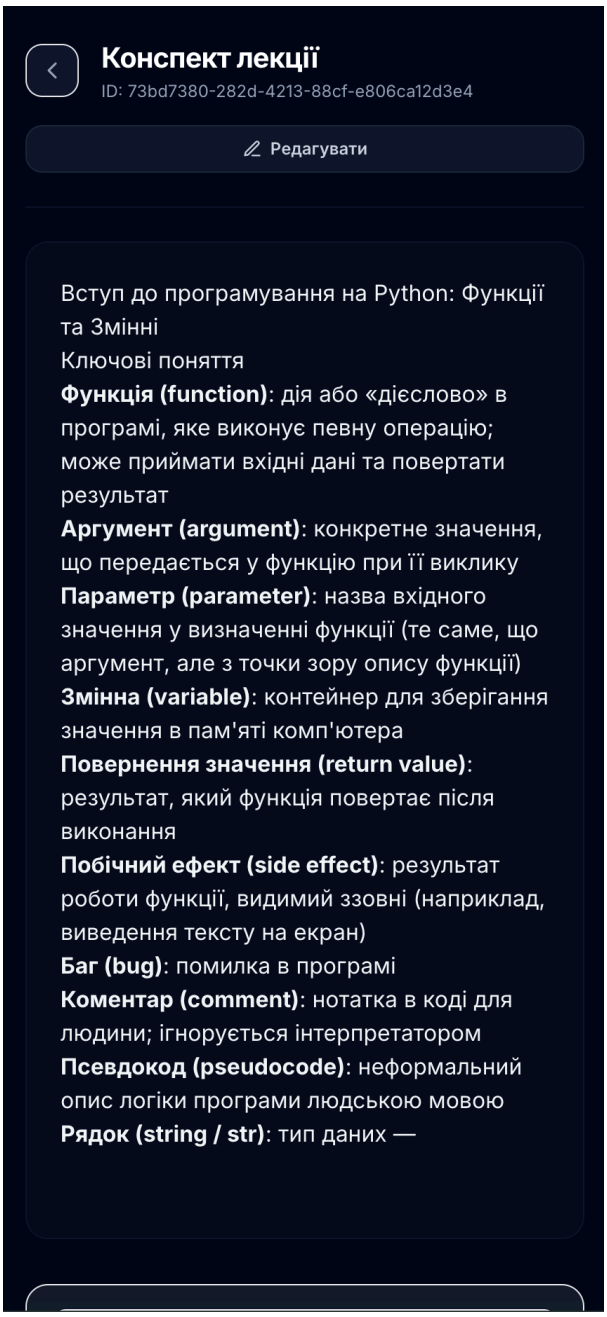


Рисунок 3.15 – Сторінка результату на телефоні

## 3.4 Аналіз результатів розробки

### 3.4.1 Тести

У backend частині було створено набір unit-тестів для перевірки роботи модулів pipeline-у, а також набір integration-тестів для перевірки API endpoint-ів, доступності Open-API-схеми та базової валідації upload-запитів (рис. 3.16). Ці

тести підтвердили коректність роботи окремих модулів, проте для повної валідації було проведено експерименти на реальних лекціях для розуміння якості конспектування.

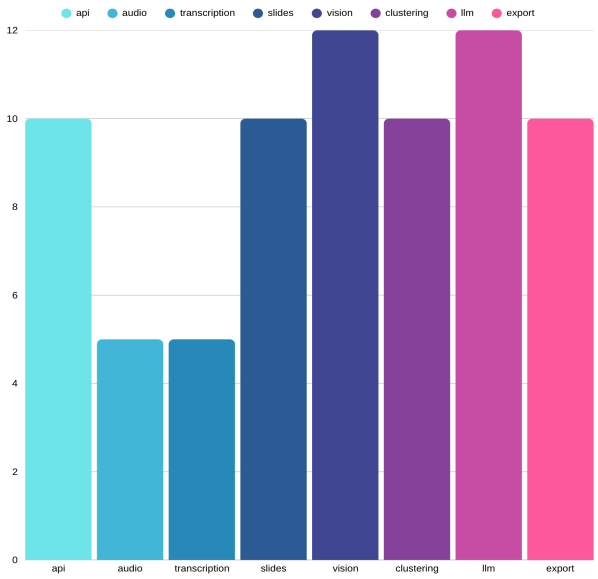


Рисунок 3.16 – Кількість тестових сценаріїв за бекенд модулями

### 3.4.2 Оцінка якості конспектування

В ході оцінки було використано сценарій з завантаженням презентації та без неї. Основними критеріями успішного проходження тесту були:

− Створення задачі після завантаження файлів;
− послідовне оновлення статусів обробки;
− відсутність помилок backend частини;
− наявність сформованого конспекту;
− якість сформованого конспекту;
− коректне відображення результату у frontend частині;
− можливість експорту сформованого конспекту.

За наяності презентації в тесті також перевірялось:

− якість обробки слайдів;
− створення PNG зображень слайдів;

− збереження ескізів у MinIO;

− формування часових міток для слайдів.

Такий підхід дозволив оцінити практичну придатність застосунку для розв’язання поставленої задачі. Результаті тестів наведено у таблиці 3.2.

Таблиця 3.2 – Результати тестів застосунку

| Довжина відео | 8хв | 8хв | 50хв | 50хв | 90хв | 90хв |
|---|---|---|---|---|---|---|
| Наявність презентації | так | ні | так | ні | так | ні |
| Створення задачі після завантаження файлів | Pass | Pass | Pass | Pass | Pass | Pass |
| Послідовне оновлення статусів обробки | Pass | Pass | Pass | Pass | Pass | Pass |
| Відсутність помилок backend частини | Pass | Pass | Pass | Pass | Pass | Pass |
| Наявність сформованого конспекту | Pass | Pass | Pass | Pass | Pass | Pass |
| Якість сформованого конспекту | Pass | Pass | Pass | Pass | Pass | Pass |
| Коректне відображення результату у frontend частині | Pass | Pass | Pass | Pass | Pass | Pass |
| Можливість експорту сформованого конспекту | Pass | Pass | Pass | Pass | Pass | Pass |
| Якість обробки слайдів | Pass | N/A | Pass | N/A | Pass | N/A |
| Створення PNG зображень слайдів | Pass | N/A | Pass | N/A | Pass | N/A |
| Збереження ескізів у MinIO | Pass | N/A | Pass | N/A | Pass | N/A |
| Формування часових міток для слайдів | Pass | N/A | Pass | N/A | Pass | N/A |

### 3.4.3 Оцінка швидкості

В ході тестування було виміряно швидкість етапів розроблено застосунку. Експерименти проводились на лекціях різної довжини, з презентацією та без. Тестування проводилося на локальній машині в CPU режимі, тобто без використання GPU для прискорення моделей. Комп'ютер для тестування має процесор M4, 16 Гб оперативної пам'яті та операційну систему MacOS. Результати тестів швидкості роботи наведені у таблиці 3.3.

Таблиця 3.3 – Результати тестів швидкості застосунку

| Довжина відео | 8хв | 8хв | 50хв | 50хв | 90хв | 90хв |
|---|---|---|---|---|---|---|
| Наявність презентації | так | ні | так | ні | так | ні |
| Витягнення аудіо | 0.4с | 0.3с | 3с | 3с | 6с | 6с |
| Розпізнавання мовлення | 2хв 28с | 2хв 31с | 14хв | 15хв | 29хв | 27хв |
| Розпізнавання слайдів | 3с | - | 43с | - | 1хв 12с | - |
| Синхронізація слайдів та відео | 27с | - | 4хв 30с | - | 8хв | - |
| Формування конспекту | 23с | 19с | 1хв 22с | 1хв 15с | 3хв 10с | 2хв 48с |
| Збереження | 1с | 1с | 4с | 4с | 7с | 7с |
| Загальний час | 3хв 22с | 2хв 51с | 20хв 42с | 16хв 22с | 41хв 35с | 29хв 1с |

# ВИСНОВОК

У рамках кваліфікаційної роботи було розроблено застосунок для автоматичного формування конспектів лекцій на основі відеозаписів та презентаційних матеріалів. У ході виконання роботи було проаналізовано проблему ручного конспектування лекцій, розглянуто сучасні підходи для вирішення цих проблем.

На основі проведеного аналізу було сформовано архітектуру системи, яка поєднує транскрипцію мовлення, обробку слайдів, візуальне зіставлення кадрів відео з презентацією та генерацію підсумкового Markdown-конспекту.

Під час розробки застосунку було реалізовано backend частину на основі FastAPI, Celery, Redis, PostgreSQL та MinIO, що дало змогу розділити швидкі HTTP-запити та довготривалу фонову обробку лекційних матеріалів. Для frontend частини використано React і TypeScript

Для розпізнавання мовлення використано faster-whisper з моделлю Whisper large-v3-turbo, для синхронізації слайдів із відео – CLIP ViT-B/32, а для генерації конспекту – Claude Sonnet 4.6.

У результаті було реалізовано повний pipeline обробки лекції. Система підтримує два основні сценарії роботи: обробку лише відеозапису лекції та обробку відео разом із презентацією. Якщо презентація відсутня, конспект формується на основі транскрипції, а якщо вона наявна, результат доповнюється структурою слайдів і часовими межами їх показу.

Після розробки було проведено функціональне тестування та оцінку швидкодії застосунку. У ході перевірки було підтверджено створення задач після завантаження файлів, послідовне оновлення статусів обробки та отримання конспекту. Для тестового відео тривалістю 8 хвилин загальний час обробки становив близько 2 хв 51 с без презентації та 3 хв 22 с із презентацією. Для довших лекцій основним фактором, що впливає на тривалість обробки, є етап розпізнавання мовлення, тоді як витягування аудіо, збереження результатів і експорт займають незначний час.

Отже, розроблений застосунок підтверджує практичну придатність запропонованого підходу для автоматизації створення навчальних конспектів. Його використання може зменшити час, який студент або викладач витрачає на ручне опрацювання відеозаписів лекцій, а також зробити навчальні матеріали більш структурованими та зручними для повторення.

У подальшому розвиток застосунку доцільно спрямувати на покращення точності синхронізації слайдів із відео, підтримку розпізнавання кількох мовців, розширення можливостей редагування конспекту та створення хмарної версії системи для стабільного доступу без залежності від локального комп'ютера користувача.

Підсумовуючи вищевикладене, можна стверджувати, що всі завдання, поставлені в межах кваліфікаційної роботи, були успішно виконані, а її головну мету досягнуто. Створена програма доводить, що поєднання мікросервісної архітектури та сучасних нейромережевих моделей є дієвим інструментом для оптимізації рутинних освітніх процесів. Запропоноване рішення є не лише технічно життєздатним, але й демонструє високий потенціал для подальшої інтеграції в системи дистанційного навчання (LMS) та платформи масових відкритих онлайн-курсів.

Впровадження подібних автоматизованих рішень є важливим кроком на шляху до загальної цифровізації освіти. Вони не лише підвищують продуктивність студентів та викладачів, але й сприяють інклюзивності навчання, роблячи інформацію більш структурованою та доступною для сприйняття. Отримані під час розробки та тестування результати мають виражену практичну цінність і можуть слугувати надійною базою для подальших досліджень у галузі освітніх технологій (EdTech) та прикладного використання штучного інтелекту.

Результати роботи апробовано у вигляді тези доповіді під час XI Міжнародної студентської конференції «Наука сьогодення: від досліджень до стратегічних рішень» (м. Умань, Україна, Молодіжна наукова ліга) [54].

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Beck K. M. (2014). Note taking effectiveness in the modern classroom. The Compass, vol. 1, № 1. Glenside, PA: Arcadia University, 9 с.

2. Stacy E. M., Cain J. (2015). Note-taking and Handouts in The Digital Age. American Journal of Pharmaceutical Education, vol. 79, № 7. Amsterdam: Elsevier BV, 107 с.

3. Sapena O., Onaindia E. (2026). Multimodal segmentation and labeling of lecture recordings for educational content retrieval. Neural Computing and Applications, vol. 38, № 7. Springer Science and Business Media LLC, 223 с.

4. Yakovleva O., Matúšová S., Liubchenko V., Maksimov H. (2025). Innovative solutions based on AI in education, on the example of generating multimodal lecture notes. Public Administration and Regional Development / Economics, Management and Marketing, vol. XXI, № 1. Bratislava: Bratislava University of Economics and Management, 140–163 с.

5. Sapena O., Onaindia E. (2022). Multimodal classification of teaching activities from university lecture recordings. *Applied Sciences*, vol. 12, № 9. Basel: MDPI AG, 4785 с.

6. Otter.ai. URL: https://otter.ai/ (дата звернення 18.04.2026).

7. Fireflies.ai. URL: https://fireflies.ai/ (дата звернення 18.04.2026).

8. Read AI. URL: https://www.read.ai/ (дата звернення 18.04.2026).

9. Tactiq. URL: https://tactiq.io/ (дата звернення 18.04.2026).

10. Notion AI. URL: https://www.notion.so/product/ai (дата звернення 18.04.2026).

11. NotebookLM. URL: https://notebooklm.google/ (дата звернення 18.04.2026).

12. ChatGPT. URL: https://chatgpt.com/ (дата звернення 18.04.2026).

13. Claude. URL: https://claude.ai/ (дата звернення 18.04.2026).

14. Eightify. URL: https://eightify.app/ (дата звернення 18.04.2026).

15. Mindgrasp AI. URL: https://mindgrasp.ai/ (дата звернення 18.04.2026).

16. Microsoft 365 Copilot. URL: https://www.microsoft.com/microsoft-365/copilot (дата звернення 18.04.2026).

17. Gemini Workspace. URL: https://workspace.google.com/solutions/ai/ (дата звернення 18.04.2026).

18. Radford A., Kim J. W., Xu T., Brockman G., McLeavey C., Sutskever I. (2023). Robust speech recognition via large-scale weak supervision. *Proceedings of the 40th International Conference on Machine Learning*. Honolulu, Hawaii, USA: PMLR, 28492–28518 с.

19. Google Cloud Speech-to-Text. URL: https://cloud.google.com/speech-to-text (дата звернення 20.04.2026).

20. Azure AI Speech. URL: https://azure.microsoft.com/en-us/products/ai-foundry/tools/speech (дата звернення 20.04.2026).

21. Deepgram. URL: https://deepgram.com (дата звернення 20.04.2026).

22. AssemblyAI. URL: https://www.assemblyai.com (дата звернення 20.04.2026).

23. PyPDF. URL: https://pypdf.readthedocs.io/en/stable/ (дата звернення 20.04.2026).

24. python-pptx. URL: https://python-pptx.readthedocs.io/en/latest/ (дата звернення 20.04.2026).

25. Smith R. (2007). An overview of the Tesseract OCR engine. *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Curitiba, Paraná, Brazil: IEEE, 629–633 с.

26. Google Vision API. URL: https://cloud.google.com/vision (дата звернення 21.04.2026).

27. Zhang J., Zhao Y., Saleh M., Liu P. (2020). PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization. *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 11328–11339 с.

28. OpenAI. GPT-4o Model. URL: https://platform.openai.com/docs/models/gpt-4o (дата звернення 21.04.2026).

29. Anthropic. Models overview. URL: https://docs.anthropic.com/en/docs/about-claude/models/overview (дата звернення 21.04.2026).

30. Google AI for Developers. Gemini models. URL: https://ai.google.dev/gemini-api/docs/models (дата звернення 21.04.2026).

31. Meta AI. Everything we announced at our first-ever LlamaCon. URL: https://ai.meta.com/blog/llamacon-llama-news/ (дата звернення 21.04.2026).

32. Mistral AI. Mistral Large 3. URL: https://docs.mistral.ai/models/mistral-large-3-25-12 (дата звернення 21.04.2026).

33. Qwen Team. Qwen3. URL: https://github.com/QwenLM/Qwen3 (дата звернення 21.04.2026).

34. Lin C.-Y. (2004). ROUGE: A package for automatic evaluation of summaries. *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*. Barcelona, Spain: Association for Computational Linguistics, 74–81 с.

35. Nallapati R., Zhou B., dos Santos C., Gulcehre C., Xiang B. (2016). Abstractive text summarization using sequence-to-sequence RNNs and beyond. *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*. Berlin, Germany: Association for Computational Linguistics, 280–290 с.

36. Gliwa B., Mochol I., Biesek M., Wawer A. (2019). SAMSum Corpus: A human-annotated dialogue dataset for abstractive summarization. *Proceedings of the 2nd Workshop on New Frontiers in Summarization*. Hong Kong : Association for Computational Linguistics, 70–79 с.

37. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O. (2024). Transforming image descriptions as a set of descriptors to construct classification features. *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 33, № 1. Yogyakarta: Institute of Advanced Engineering and Science, 113–125 с.

38. Yakovleva O., Matúšová S., Koshel V. (2025). Implementation of AI approaches in current tools for managing image collections to improve the search capabilities. *Grail of Science*, № 49. Vinnytsia–Vienna: European Scientific Platform, pp. 752–755 с.

39. Яковлева О. В., Кускова І. В. (2006). Дослідження результатів сегментації зображень методом матриць збігів. *Вісник Національного технічного університету «ХПІ»*, № 39. Харків: НТУ «ХПІ», 164–171 с.

40. Яковлева О. В., Панченко І. А. (2007). Застосування енергетичних характеристик Лавса для сегментації зображень. *Біоніка інтелекту: науково-технічний журнал*, № 2(67). Харків: ХНУРЕ, 94–98 с.

41. Яковлева Е. В., Нестерова Е. П. (2009). Сравнительный анализ методов характеристик Лавса и матриц совпадений в задачах сегментации текстурных изображений. *Прикладная радиоэлектроника: научно-технический журнал*. Харків: ХНУРЕ, т. 8, № 2, 181-187 с.

42. Yakovleva O., Nebeský Ľ., Liakhov P. (2023). Research methods of texture image analysis to solve the texture search problem. *Proceedings of the IV International Scientific and Practical Conference "The world of modern technologies and inventions"*. Vienna: International Science Group, 252-261 с.

43. Кобилін О., Вечірська І., Афанасьєв А. (2024). Аналіз існуючих моделей глибинного навчання в задачах обробки природної мови. *Information Technology: Computer Science, Software Engineering and Cyber Security*, № 3. Одеса: Гельветика, 63-76 с.

44. Python documentation. URL: https://docs.python.org/3/ (дата звернення 17.05.2026).

45. TypeScript documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення 17.05.2026).

46. React documentation. URL: https://react.dev (дата звернення 17.05.2026).

47. Tailwind CSS documentation. URL: https://v2.tailwindcss.com/docs (дата звернення 17.05.2026)

48. FastAPI documentation. URL: https://fastapi.tiangolo.com (дата звернення 18.05.2026)

49. Celery documentation. URL: https://docs.celeryq.dev/en/stable/ (дата звернення 20.05.2026)

50. Redis documentation. URL: https://redis.io/docs/latest/ (дата звернення 20.05.2026)

51. PostgreSQL documentation. URL: https://www.postgresql.org/docs/ (дата звернення 24.05.2026)

52. SQLAlchemy documentation. URL: https://docs.sqlalchemy.org/en/20/ (дата звернення 24.05.2026)

53. Alembic documentation. URL: https://alembic.sqlalchemy.org/en/latest/ (дата звернення 27.05.2026)

54. Шевченко І.С., Вечірська І.Д. (2026) Розробка вебзастосунку для автоматичного конспектування та аналізу навчальних матеріалів з використанням великих мовних моделей. *Тези одинадцятої міжнародної студентської наукової конференції «Наука сьогодення: від досліджень до стратегічних рішень» (26 червня 2026 року). Умань, С. 108-110*.