

Міністерство освіти і науки України  
Харківський національний університет радіоелектроніки

Факультет \_\_\_\_\_ Центр післядипломної освіти  
(повна назва)  
Кафедра \_\_\_\_\_ Штучного інтелекту  
(повна назва)

**КВАЛІФІКАЦІЙНА РОБОТА**  
**Пояснювальна записка**

рівень вищої освіти \_\_\_\_\_ перший (бакалаврський)

Мультимодальний синтез навчального відеоконтенту на основі семантичного  
аналізу презентаційних та аудіоматеріалів  
(тема)

Виконав:  
здобувач \_\_\_\_\_ другого \_\_\_\_\_ року навчання,  
групи \_\_\_\_\_ ІТШІпз-24-1

\_\_\_\_\_ Дар'я ГРІНЬОВА  
(власне ім'я, прізвище)

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

Тип програми \_\_\_\_\_ освітньо-професійна  
Освітня програма \_\_\_\_\_ Штучний інтелект  
(повна назва освітньої програми)

Керівник \_\_\_\_\_ проф. Наталія РЯБОВА  
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ

\_\_\_\_\_  
(підпис)

\_\_\_\_\_ Лариса ЧАЛА  
(власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет \_\_\_\_\_ Центр післядипломної освіти \_\_\_\_\_

Кафедра \_\_\_\_\_ Штучного інтелекту \_\_\_\_\_

Рівень вищої освіти \_\_\_\_\_ перший (бакалаврський) \_\_\_\_\_

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

Тип програми \_\_\_\_\_ освітньо-професійна \_\_\_\_\_

Освітня програма \_\_\_\_\_ Штучний інтелект \_\_\_\_\_  
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри \_\_\_\_\_

(підпис)

« \_\_\_\_\_ » \_\_\_\_\_ 20 \_\_\_\_ р.

**ЗАВДАННЯ**  
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві \_\_\_\_\_ Грінювій Дар'ї Олександрівні \_\_\_\_\_  
(прізвище, ім'я, по батькові)

Тема роботи \_\_\_\_\_ Мультимодальний синтез навчального відеоконтенту на основі семантичного аналізу презентаційних та аудіоматеріалів \_\_\_\_\_

затверджена наказом університету від 18 травня 2026 р. № 77Стз

2. Термін подання здобувачем роботи до екзаменаційної комісії 16 червня 2026 р.

3. Вихідні дані до роботи Науково-технічні публікації з тематики великих мовних моделей і синтезу мовлення, дані Інтернет-джерел (документація OpenAI API, IMARC Group, Statista), документація бібліотек Python (PyPDF2, PyMuPDF, soundfile), документація Node.js, Express, FFmpeg, реальні PDF-презентації навчальних лекцій та аудіозаписи для тестування системи.

4. Перелік питань, що потрібно опрацювати в роботі \_\_\_\_\_

1) Аналіз предметної галузі та постановка задачі \_\_\_\_\_

2) Методи та архітектура системи \_\_\_\_\_

3) Програмна реалізація системи \_\_\_\_\_

4) Тестування та аналіз результатів \_\_\_\_\_

## КАЛЕНДАРНИЙ ПЛАН

| №  | Назва етапів роботи                         | Строк / термін виконання етапів роботи | Примітка |
|----|---------------------------------------------|----------------------------------------|----------|
| 1  | Отримання завдання на кваліфікаційну роботу | 18.05.2026                             | виконано |
| 2  | Аналіз предметної галузі                    | 20.05.2026                             | виконано |
| 3  | Огляд існуючих рішень та технологій         | 22.05.2026                             | виконано |
| 4  | Аналіз підходів до генерації подкастів      | 26.05.2026                             | виконано |
| 5  | Проектування архітектури програмної системи | 28.05.2026                             | виконано |
| 6  | Написання пояснювальної записки             | 29.05.2026                             | виконано |
| 7  | Перевірка на академічний плагіат            | 02.06.2026                             | виконано |
| 8  | Нормоконтроль                               | 03.06.2026                             | виконано |
| 9  | Підготовка презентації та доповіді          | 08.06.2026                             | виконано |
| 10 | Попередній захист                           | 10.06.2026                             | виконано |
| 11 | Рецензування                                | 12.06.2026                             | виконано |
| 12 | Захист перед ЕК                             | 16.06.2026                             |          |
|    |                                             |                                        |          |
|    |                                             |                                        |          |
|    |                                             |                                        |          |
|    |                                             |                                        |          |
|    |                                             |                                        |          |
|    |                                             |                                        |          |
|    |                                             |                                        |          |
|    |                                             |                                        |          |
|    |                                             |                                        |          |

Дата видачі завдання 18 травня 2026 р.

Здобувач \_\_\_\_\_

(підпис)

Керівник роботи \_\_\_\_\_

(підпис)

проф. Наталія РЯБОВА

(посада, власне ім'я, прізвище)

## РЕФЕРАТ

Пояснювальна записка: 83 с., 12 рис., 7 табл., 1 дод., 26 джерел.

АВТОМАТИЧНА СИНХРОНІЗАЦІЯ, ВЕЛИКІ МОВНІ МОДЕЛІ, ВІДЕОЛЕКЦІЯ, МАППІНГ СЛАЙДІВ, РОЗПІЗНАВАННЯ МОВЛЕННЯ, СЕМАНТИЧНИЙ АНАЛІЗ.

Об'єкт дослідження – процес автоматизованого мультимодального синтезу та конвеєрної генерації навчального відеоконтенту шляхом наскрізного семантичного аналізу й динамічного узгодження різнорідних презентаційних та аудіоматеріалів.

Предмет дослідження – моделі, алгоритми та програмні архітектури семантичного маппінгу дискретних візуальних структур, когнітивного аналізу графічних компонентів лекцій та pre-цизійного часового вирівнювання мультимедійних потоків із використанням генеративних великих мовних моделей та систем автоматичного розпізнавання мовлення.

Мета роботи – розробка веб-застосунку PodcastSlide для автоматичної генерації синхронізованих відеолекцій на основі PDF-презентацій та аудіозаписів із застосуванням методів ASR та семантичного аналізу на базі великих мовних моделей.

Методи дослідження – обробка природної мови та трансформерні моделі (GPT-5, GPT-4O-mini), комп'ютерний зір та мультимодальна екстракція даних (GPT-4O Vision), автоматичне розпізнавання мовлення на основі глибокого навчання (Whisper API) та цифрова обробка сигналів (FFmpeg).

Розроблений програмний застосунок PodcastSlide автоматизує синтез синхронізованого відеоконтенту у форматі MP4 з PDF-презентацій та аудіозаписів за допомогою новітнього алгоритму семантичного маппінгу на базі GPT-5 із механізмом автоматичної корекції.

## **ABSTRACT**

Bachelor's thesis contains: 83 pp., 12 fig., 7 tabl., 1 ann., 26 references.

AUTOMATIC SYNCHRONIZATION, LARGE LANGUAGE MODELS, SEMANTIC ANALYSIS, SLIDE MAPPING, SPEECH RECOGNITION, VIDEO LECTURE.

The object of the study is the process of automated multimodal synthesis and pipeline-based generation of educational video content through end-to-end semantic analysis and dynamic alignment of heterogeneous presentation and audio materials.

The subject of the study is models, algorithms, and software architectures for semantic mapping of discrete visual structures, cognitive analysis of graphical components of lectures, and precision temporal alignment of multimedia streams using generative large language models and automatic speech recognition systems.

The purpose of the work is the development of the PodcastSlide web application for automatic generation of synchronized video lectures based on PDF presentations and audio recordings using ASR methods and semantic analysis powered by large language models.

Research methods include natural language processing and transformer-based models (GPT-5, GPT-4o-mini), computer vision and multimodal data extraction (GPT-4o Vision), automatic speech recognition based on deep learning (Whisper API), and digital signal processing (FFmpeg).

The developed PodcastSlide software application automates the synthesis of synchronized video content in MP4 format from PDF presentations and audio recordings using an advanced semantic mapping algorithm based on GPT-5 with an automatic correction mechanism.

## ЗМІСТ

|                                                                                       |    |
|---------------------------------------------------------------------------------------|----|
| Перелік умовних позначень, символів, одиниць, скорочень і термінів .....              | 10 |
| Вступ.....                                                                            | 11 |
| 1 Аналіз предметної галузі та постановка задачі.....                                  | 15 |
| 1.1 Розвиток систем автоматичного створення навчального відеоконтенту.....            | 15 |
| 1.1.1 Еволюція технологій дистанційного навчання та відеолекцій .                     | 15 |
| 1.1.2 Застосування автоматичного розпізнавання мовлення в освітніх системах .....     | 17 |
| 1.1.3 Мультимодальні підходи до синтезу навчальних медіаматеріалів.....               | 18 |
| 1.1.4 Великі мовні моделі у задачах семантичного аналізу текстових матеріалів .....   | 19 |
| 1.2 Аналіз існуючих підходів та програмних рішень.....                                | 20 |
| 1.2.1 Системи автоматичного монтажу відеолекцій. Огляд та порівняння.....             | 20 |
| 1.2.2 Методи автоматичного розпізнавання мовлення з пословними таймстемпами .....     | 22 |
| 1.2.3 Методи виявлення пауз у мовленні як маркерів структурних меж (VAD) .....        | 23 |
| 1.2.4 Порівняльний аналіз існуючих рішень для синхронізації презентацій з аудіо ..... | 24 |
| 1.3 Актуальність задачі та обмеження існуючих рішень.....                             | 25 |
| 1.3.1 Проблема автоматизації виробництва навчального відеоконтенту .....              | 25 |
| 1.3.2 Відсутність повністю автоматизованого рішення.....                              | 26 |
| 1.3.3 Потреба у семантичному аналізі для точного визначення меж між слайдами .....    | 26 |
| 1.4 Постановка задачі та мета роботи .....                                            | 27 |

|       |                                                                               |    |
|-------|-------------------------------------------------------------------------------|----|
| 2     | Методи та архітектура системи .....                                           | 29 |
| 2.1   | Загальна архітектура мультимодального конвеєра обробки .....                  | 29 |
| 2.1.1 | Послідовність фаз обробки .....                                               | 29 |
| 2.1.2 | Взаємодія Node.js-сервера та Python-модулів через CLI-міст<br>runner.py ..... | 31 |
| 2.1.3 | Передача стану між фазами через JSON state-файл .....                         | 31 |
| 2.1.4 | Конфігурація системних параметрів.....                                        | 32 |
| 2.2   | Методи обробки презентаційних матеріалів.....                                 | 33 |
| 2.2.1 | Витяг тексту з PDF-документів засобами PyPDF2.....                            | 33 |
| 2.2.2 | Порогова логіка перемикавання на OCR при нестачі нативного<br>тексту .....    | 33 |
| 2.2.3 | OCR-аналіз графічних слайдів засобами GPT-4O Vision .....                     | 34 |
| 2.2.4 | Рендеринг сторінок PDF у растрові зображення засобами<br>PyMuPDF.....         | 34 |
| 2.3   | Методи автоматичного розпізнавання та сегментації мовлення .....              | 36 |
| 2.3.1 | Транскрипція аудіо з пословними таймстемпами засобами<br>Whisper API.....     | 36 |
| 2.3.2 | Чанкова обробка аудіо та корекція часових зміщень між<br>чанками .....        | 36 |
| 2.3.3 | Алгоритм виявлення пауз за пороговим значенням міжслівного<br>інтервалу ..... | 37 |
| 2.3.4 | Побудова розміченого транскрипту з іменованими маркерами<br>пауз .....        | 39 |
| 2.4   | Методи семантичного мапінгу слайдів і транскрипту.....                        | 39 |
| 2.4.1 | Формування структурованого промпту для GPT-5.....                             | 39 |
| 2.4.2 | Визначення меж слайдів через зіставлення ключових слів і<br>термінів .....    | 40 |
| 2.4.3 | Алгоритм валідації мапінгу за часовим розподілом аудіо .....                  | 41 |
| 2.4.4 | JSON-логування результатів мапінгу .....                                      | 43 |
| 2.5   | Методи синтезу відеоматеріалу.....                                            | 43 |

|                                                                                                                       |    |
|-----------------------------------------------------------------------------------------------------------------------|----|
| 2.5.1 Посегментний рендеринг відеокліпів .....                                                                        | 43 |
| 2.5.2 Точне нарізання аудіо за таймстемпами пауз (sample-accurate)                                                    | 44 |
| 2.5.3 Конкатенація кліпів без повторного кодування (stream copy) ..                                                   | 44 |
| 2.6 Обґрунтування вибору технологій та порівняльний аналіз<br>альтернатив .....                                       | 45 |
| 3 Програмна реалізація системи .....                                                                                  | 48 |
| 3.1 Архітектура програмного забезпечення .....                                                                        | 48 |
| 3.1.1 Файлова організація проєкту та структура пакету steps .....                                                     | 48 |
| 3.1.2 CLI-міст runner.py. Команди slides, process, sync, generate.....                                                | 49 |
| 3.1.3 Передача прогресу від Python до клієнта через JSON-рядки та<br>SSE .....                                        | 50 |
| 3.2 Реалізація модуля обробки PDF (parse_pdf, extract_text_from_image)<br>.....                                       | 52 |
| 3.2.1 Парсинг тексту засобами PyPDF2 та порогова логіка<br>перемикання на OCR .....                                   | 52 |
| 3.2.2 Інтеграція OCR через GPT-4O Vision з base64-кодуванням<br>зображень .....                                       | 53 |
| 3.3 Реалізація модуля розпізнавання та сегментації мовлення<br>(transcribe_audio, find_pauses, build_transcript)..... | 54 |
| 3.3.1 Інтеграція Whisper API: чанкова обробка та корекція часових<br>зміщень.....                                     | 54 |
| 3.3.2 Реалізація алгоритму виявлення пауз за порогом<br>PAUSE_THRESHOLD .....                                         | 55 |
| 3.3.3 Побудова розміченого транскрипту з маркерами пауз .....                                                         | 56 |
| 3.4 Реалізація модуля семантичного маппінгу (match_slides).....                                                       | 57 |
| 3.4.1 Формування промпту та взаємодія з GPT-5.....                                                                    | 57 |
| 3.4.2 Реалізація алгоритму валідації та автоматичної корекції<br>маппінгу .....                                       | 58 |
| 3.4.3 JSON-логування результатів у debug_match_slides.json .....                                                      | 59 |
| 3.5 Реалізація модуля відеогенерації (build_video).....                                                               | 60 |

|                                                                                             |    |
|---------------------------------------------------------------------------------------------|----|
| 3.5.1 Рендеринг слайдів у PNG та посегментне нарізання аудіо в AAC .....                    | 60 |
| 3.5.2 Побудова concat-файлу та конкатенація через FFmpeg .....                              | 61 |
| 3.6 Реалізація веб-застосунку .....                                                         | 62 |
| 3.6.1 Серверна частина. Express.js, управління сесією та SSE-стрімінг .....                 | 62 |
| 3.6.2 Клієнтська частина. Інтерфейс завантаження, перегляду таймкодів та експорту MP4 ..... | 63 |
| 4 Тестування та аналіз результатів .....                                                    | 67 |
| 4.1 Опис користувацького інтерфейсу та сценарії використання .....                          | 67 |
| 4.2 Тестування системи на реальних навчальних матеріалах .....                              | 68 |
| 4.2.1 Опис тестової презентації та умов запуску .....                                       | 68 |
| 4.2.2 Аналіз результатів семантичного мапінгу за даними debug_match_slides.json .....       | 68 |
| 4.2.3 Розподіл аудіо між слайдами. Статистика тривалостей та оцінка рівномірності .....     | 70 |
| 4.2.4 Виявлені технічні проблеми та реалізовані інженерні рішення .....                     | 71 |
| 4.3 Оцінка практичної цінності та економічної ефективності системи ..                       | 72 |
| 4.3.1 Порівняння часових витрат. Ручний монтаж versus автоматична система .....             | 72 |
| 4.3.2 Вартість обробки через зовнішні API .....                                             | 73 |
| 4.4 Перспективи розвитку системи .....                                                      | 74 |
| Висновки .....                                                                              | 76 |
| Перелік джерел посилання .....                                                              | 79 |
| Додаток А Відомість кваліфікаційної роботи .....                                            | 83 |

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

BMM – велика мовна модель;

AAC – Advanced Audio Coding – формат стиснення аудіо з втратами;

API – Application Programming Interface – інтерфейс програмування застосунків;

ASR – Automatic Speech Recognition – автоматичне розпізнавання мовлення;

CLI – Command Line Interface – інтерфейс командного рядка;

FPS – Frames Per Second – кадрів на секунду;

JSON – JavaScript Object Notation – текстовий формат обміну даними;

LLM – Large Language Model – велика мовна модель;

MP3 – MPEG Audio Layer III – формат стиснення аудіо;

MP4 – MPEG-4 Part 14 – формат відеофайлу;

OCR – Optical Character Recognition – оптичне розпізнавання символів;

PCM – Pulse-Code Modulation – імпульсно-кодова модуляція;

PDF – Portable Document Format – формат портативного документа;

SSE – Server-Sent Events – технологія серверних подій у реальному часі;

UI – User Interface – інтерфейс користувача;

UUID – Universally Unique Identifier – універсально унікальний ідентифікатор;

VAD – Voice Activity Detection – виявлення активності мовлення;

WAV – Waveform Audio File Format – формат аудіофайлу без стиснення.

## ВСТУП

Цифрова трансформація освіти, що набула стрімкого прискорення після 2022 року, зумовлена експансією когнітивних систем штучного інтелекту та фундаментальною зміною парадигми сприйняття інформації. Глобальний перехід до дистанційних та гібридних моделей навчання кардинально змінив вимоги до форматів подачі навчального матеріалу. Сучасний студент дедалі частіше споживає освітній контент у мобільному, асинхронному форматі – під час поїздки, прогулянки або виконання рутинних завдань [1]. У цьому контексті аудіоподкаст перетворився з нішевого медіаформату на повноцінний освітній інструмент: за даними аналітичної компанії IMARC Group, обсяг глобального ринку e-learning у 2025 році досяг 369,7 мільярда доларів США, і аудіоформати становлять одну з найбільш динамічних його складових [2], [3].

Попри зростаючий попит на освітні подкасти, їхнє виробництво залишається трудомістким і технологічно складним процесом. Традиційний виробничий ланцюжок – написання сценарію, студійний запис, монтаж і мастеринг – потребує від університетського викладача без спеціалізованого досвіду від чотирьох до восьми годин роботи на один епізод тривалістю двадцять – тридцять хвилин. Це фактично робить систематичне виробництво аудіоконтенту недоступним для більшості освітніх організацій, незважаючи на наявну потребу.

Аналіз наявних програмних рішень свідчить про відсутність повністю автоматизованого ланцюжка перетворення PDF-презентації лекції на готовий синхронізований відеоряд. Сучасні інструменти – OBS Studio, Camtasia, Descript, Synthesia, Panopto [4] та інші – вирішують лише окремі підзадачі виробничого процесу і потребують значної ментальної та часової участі людини. Найближчий концептуальний аналог – сервіс Google NotebookLM [5] з функцією Audio Overview – генерує інтерактивний аудіоформат за участю двох синтезованих дикторів на основі завантажених

документів, однак обробляє матеріал як суцільний текстовий масив без розрізнення структури слайдів, не підтримує конфігурацію ролей учасників і не надає відкритого API для інтеграції у сторонні університетські платформи. Це зумовлює відчутний технологічний розрив у процесах повної автоматизації виробництва навчального медіаконтенту. Більшість комерційних систем наразі ігнорують необхідність дискретизації та точної таймінгової прив'язки звукового потоку до конкретних візуальних об'єктів. Закритий характер пропрієтарних архітектур додатково унеможлиблює їх гнучку кастомізацію під специфічні академічні вимоги. Відтак постає гостра науково-практична потреба у створенні відкритої та модульно незалежної альтернативи. Розроблюваний програмний комплекс PodcastSlide покликаний подолати ці обмеження шляхом побудови наскрізного автономного конвеєра обробки.

Актуальність роботи визначається необхідністю заповнення виявленого технологічного пробілу шляхом застосування сучасних методів штучного інтелекту – автоматичного розпізнавання мовлення та семантичного аналізу на базі великих мовних моделей – для вирішення задачі автоматичної синхронізації презентаційних та аудіоматеріалів. Поява моделей з широким контекстним вікном (GPT-5) та високоточних ASR-систем із пословними таймстемпами (Whisper API) [6] вперше робить можливим вирішення цієї задачі в загальному вигляді – для будь-якої мови, будь-якої структури презентації та будь-якого стилю викладання [7].

Метою роботи є розробка та практична реалізація системи мультимодального синтезу навчального відеоконтенту на основі семантичного аналізу презентаційних та аудіоматеріалів, що автоматично перетворює сталі структури PDF-лекцій та відповідні звукові записи на фінальний синхронізований відеоряд у форматі діалогу між двома віртуальними співрозмовниками з використанням великих мовних моделей. Система реалізує мультимодальний підхід до обробки вхідних даних: текстові слайди обробляються засобами бібліотеки PyPDF2, тоді як графічні

слайди з діаграмами, формулами та таблицями розпізнаються через GPT-4O Vision, що забезпечує повноту витягу змістовної інформації незалежно від типу вихідних матеріалів.

Об'єкт дослідження – процес автоматизованого мультимодального синтезу та конвеєрної генерації навчального відеоконтенту шляхом наскрізного семантичного аналізу й динамічного узгодження різнорідних презентаційних та аудіоматеріалів.

Предмет дослідження – моделі, алгоритми та програмні архітектури семантичного маппінгу дискретних візуальних структур, когнітивного аналізу графічних компонентів лекцій та прецизійного часового вирівнювання мультимедійних потоків із використанням генеративних великих мовних моделей та систем автоматичного розпізнавання мовлення.

Методи дослідження. Для вирішення поставлених у роботі завдань та забезпечення теоретико-практичної валідності результатів було застосовано комплексний науково-інженерний інструментарій, що базується на таких методах:

- методи системного аналізу та архітектурного проектування складних програмних комплексів – для декомпозиції обчислювального конвеєра на ізольовані автономні підсистеми (пакети), побудови подієво-орієнтованої моделі міжпроцесної комунікації та мінімізації зв'язності компонентів;

- методи штучного інтелекту й обробки природної мови (NLP), зокрема трансформерні архітектури великих мовних моделей (GPT-5 та GPT-4O-mini) – для проведення глибинного контекстного аналізу транскриптів, побудови інструкційних промптів, інтерпретації лекційного наративу та реалізації алгоритму маппінгу логічних меж слайдів;

- методи комп'ютерного зору та мультимодального аналізу даних (інструментарій GPT-4O Vision) – для оптичного та семантичного розпізнавання неструктурованої графічної інформації, детекції та

інтерпретації формульних залежностей, таблиць і діаграм, інтегрованих у візуальний ряд презентацій;

- методи автоматичного розпізнавання мовлення (ASR) на основі глибоких нейронних мереж (Whisper API) – для високоточного акустичного аналізу звукових доріжок лекцій та екстракції пословних часових міток (word-level timestamps);

- методи цифрової обробки сигналів та низькорівневого відеосинтезу (на базі системного рушія FFmpeg) – для асинхронного кадрового кодування, семпл-точного нарізання мультимедійних фрагментів та конкатенації медіапотоків у контейнери високої щільності без повторного перекодування.

Практичне значення роботи полягає у можливості безпосереднього застосування розробленої системи університетськими викладачами, авторами освітніх курсів та освітніми платформами. За результатами тестування, система скорочує час виробництва навчального подкасту з кількох годин до дванадцяти – сімнадцяти хвилин при собівартості обробки однієї презентації, що становить менше 0,6 USD за поточним курсом, це на два-три порядки нижче за вартість класичного студійного запису з акторами та подальшого відеомонтажу.

Робота складається з чотирьох розділів. У першому розділі проведено аналіз предметної галузі, розглянуто еволюцію систем синтезу мовлення, порівняно існуючі рішення та обґрунтовано актуальність задачі. Другий розділ присвячено методам та архітектурі системи: детально описано конвеєрну архітектуру, методику промпт-інжинірингу з шістьма інструкційними блоками та двоетапну схему контролю якості транскрипту. У третьому розділі викладено програмну реалізацію всіх модулів системи та веб-застосунку. Четвертий розділ містить результати тестування на реальних навчальних матеріалах, аналіз виявлених проблем та оцінку практичної і економічної ефективності системи.

## 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

### 1.1 Розвиток систем автоматичного створення навчального відеоконтенту

#### 1.1.1 Еволюція технологій дистанційного навчання та відеолекцій

Дистанційна освіта пройшла тривалий шлях трансформації – від заочного навчання через поштові відправлення до повноцінних інтерактивних платформ із відеолекціями, тестуванням та зворотним зв'язком у реальному часі. Перші форми електронного навчання з'явилися у 1990-х роках разом із масовим поширенням мережі Інтернет та інтерактивних CD-ROM курсів [1]. Однак справжній злам відбувся на початку 2000-х, коли платформи на кшталт Moodle та Blackboard уможливили систематичне керування навчальним контентом у цифровому середовищі [8].

Відеолекція як формат стала центральним елементом дистанційного навчання після 2012 року – так званого «року MOOC» (Massive Open Online Course), коли Coursera, edX та Udacity запропонували мільйонам користувачів відеокурси від провідних університетів світу [9]. Вже станом на 2026 рік Coursera нараховує понад 148 мільйонів зареєстрованих користувачів [10], а щоквартальний приріст аудиторії платформи сягає 7 мільйонів осіб. Відеолекція перетворилась на стандарт де-факто для подання навчального матеріалу в дистанційній освіті [3]: вона поєднує переваги живого виступу лектора з можливістю перегляду в зручний час, повторного відтворення складних фрагментів та автоматичного субтитрування.

Паралельно зростає і глобальний ринок e-learning. За даними IMARC Group, у 2025 році він досяг позначки 369,7 млрд доларів США, а до 2034 року очікується його зростання до 731,8 млрд доларів при середньорічному

темпи приросту (CAGR) близько 7,65% (рисунок 1.1). Technavio наводить ще агресивніший прогноз: за їхніми оцінками, ринок збільшиться на 549 млрд доларів у період з 2025 по 2030 рік при CAGR 24,2%, що свідчить про надзвичайно динамічне середовище та зростаючий попит на цифрові освітні рішення [2].

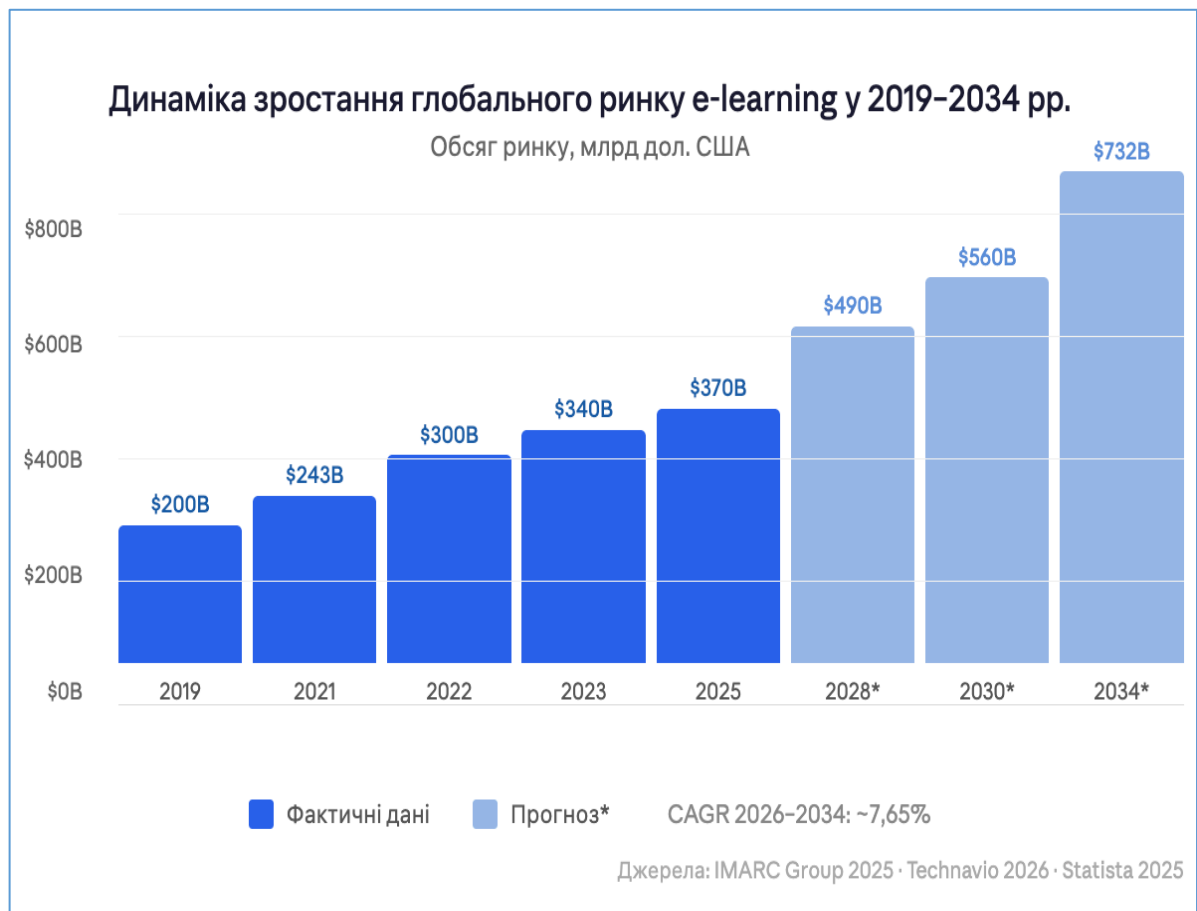


Рисунок 1.1 – Динаміка зростання глобального ринку e-learning у 2019–2034 рр. (у млрд дол. США)

Однак масштабування виробництва відеоконтенту наштовхується на практичне обмеження: підготовка однієї 40-хвилинної відеолекції традиційним способом – запис, монтаж, синхронізація презентації з аудіо – потребує від двох до чотирьох годин ручної роботи [11]. При цьому переважна більшість викладачів університетів вже мають у своєму розпорядженні обидва ключові компоненти: PDF-презентацію зі

структурованим матеріалом [12] та аудіозапис лекції. Таким чином, головною проблемою стає не відсутність вихідних матеріалів, а відсутність інструменту, який автоматично перетворив би їх у готовий відеопродукт [7].

### 1.1.2 Застосування автоматичного розпізнавання мовлення в освітніх системах

Технології автоматичного розпізнавання мовлення (ASR – Automatic Speech Recognition) відіграють ключову роль у сучасній обробці аудіоконтенту. Перші комерційні ASR-системи, такі як Dragon Naturally Speaking, з'явилися наприкінці 1990-х і вимагали тривалого «навчання» під голос конкретного користувача. Принциповий прорив стався з переходом до глибоких нейронних мереж у 2010-х роках, коли точність розпізнавання почала наближатись до рівня людини [13].

У 2022 році OpenAI оприлюднила модель Whisper, натреновану на 680000 годин різноманітних мультимовних аудіоматеріалів. Whisper продемонструвала найнижчий рівень похибки слів (WER – Word Error Rate) серед загальнодоступних моделей: за даними бенчмарку MLPerf Inference 2025, вона скоротила WER порівняно з попередньою еталонною моделлю (RNN-T) більш ніж на 72%. Критично важливою для задач синхронізації є підтримка пословних таймстемпів (word-level timestamps): через параметр `timestamp_granularities=["word"]` у форматі відповіді `verbose_json` API повертає точні часові мітки початку та кінця для кожного розпізнаного слова. Саме ця можливість є фундаментом для автоматичного зіставлення мовлення з текстом слайдів.

В освітньому контексті ASR застосовується для автоматичного субтитрування відеолекцій, індексування контенту для пошуку, аналізу темпу мовлення викладача та – що найбільш релевантно для цієї роботи – синхронізації голосового запису з візуальними матеріалами. Платформи Panopto та Kaltura Capture вже інтегрували ASR-транскрипцію у свій

базовий функціонал, однак їхній підхід суттєво відрізняється від запропонованого у цій роботі, що буде детально розглянуто у підрозділі 1.2.

### 1.1.3 Мультиmodalьні підходи до синтезу навчальних медіаматеріалів

Термін «мультиmodalьність» у контексті машинного навчання та обробки даних означає роботу з кількома типами вхідної інформації одночасно – текстом, зображеннями, аудіо, відео. Мультиmodalьні підходи до створення навчального контенту отримали значний імпульс після появи великих мультиmodalьних моделей, зокрема GPT-4O від OpenAI [9], які здатні одночасно аналізувати текст та зображення в межах єдиного запиту. Дослідження у цій галузі активно розвиваються: Chen досліджує персоналізовані алгоритми мультиmodalьної генерації навчального контенту [14], Kumar та Thenmozhi розробляють фреймворк для автоматичної генерації слайдів з відеоматеріалів [15], а Wang et al. пропонують методи крос-modalьного вирівнювання для навчання мультиmodalьних представлень [16].

У задачі автоматичного синтезу відеолекцій мультиmodalьність проявляється на кількох рівнях. По-перше, вхідні дані є принципово різнорідними: аудіозапис мовлення лектора та PDF-документ із слайдами. По-друге, PDF-презентації самі по собі є мультиmodalьними – вони містять як нативний текст, який можна витягти програмно, так і графічні слайди, де текстова інформація є частиною растрового зображення і потребує OCR-аналізу. По-третє, процес зіставлення мовлення зі слайдами вимагає семантичного порівняння двох різних модальностей: транскрипту аудіо та текстового вмісту слайдів.

Саме ця тришарова мультиmodalьність – аудіо, PDF-текст, зображення слайдів – визначає архітектурні рішення системи PodcastSlide та обумовлює залучення спеціалізованих інструментів для кожного типу

даних: Whisper API для аудіо, PyPDF2 та PyMuPDF для текстового і графічного шарів PDF, GPT-4O Vision для OCR-аналізу графічних слайдів.

#### 1.1.4 Великі мовні моделі у задачах семантичного аналізу текстових матеріалів

Великі мовні моделі (LLM – Large Language Models) революціонізували підходи до розуміння та генерації природної мови [17]. GPT-4o, представлений OpenAI у 2024 році, підтримує контекстне вікно до 128 тисяч токенів і здатний одночасно обробляти текстові та графічні дані [6]. Це робить його придатним для аналізу довгих транскриптів лекцій у поєднанні з текстом кількох десятків слайдів у межах одного запиту.

Для задачі семантичного маппінгу – визначення того, на якому моменті аудіозапису лектор переходить від обговорення одного слайду до наступного – LLM є принципово кращим інструментом порівняно з класичними методами: пошуком за ключовими словами або статистичними моделями схожості текстів [18], [19]. Причина в тому, що лектор рідко дослівно зачитує текст зі слайдів – він пояснює, перефразовує, наводить приклади, відступає від теми та повертається до неї. Тільки модель з розумінням природної мови здатна зіставити «\$253 мільярди доларів у 2023 році» у транскрипті з «\$253B – обсяг ринку e-learning 2023» у тексті слайду.

У системі PodcastSlide для завдання маппінгу меж слайдів використовується GPT-5 – найпотужніша доступна модель OpenAI, тоді як GPT-4O-mini застосовується для менш критичних задач (OCR, обробка зображень) [6]. Такий розподіл забезпечує баланс між точністю ключового алгоритму та економічністю обробки [7]. Ця дворівнева каскадна конфігурація нейромереж оптимізує витрати обчислювальних ресурсів без компромісів щодо глибини когнітивного аналізу. Завдяки цьому досягається прецизійне динамічне узгодження часових міток аудіопотоку із відповідним

візуальним рядом лекцій. Зрештою, розроблений підхід забезпечує високу архітектурну життєздатність та масштабованість платформи в умовах інтенсивного освітнього процесу.

## 1.2 Аналіз існуючих підходів та програмних рішень

### 1.2.1 Системи автоматичного монтажу відеолекцій. Огляд та порівняння

Вибір програмного забезпечення для створення відеолекцій є достатньо широким, проте жоден з наявних продуктів не реалізує повністю автоматизований ланцюжок «готова PDF-презентація + готовий аудіофайл → синхронізоване відео». Розглянемо основні рішення детальніше.

OBS Studio є безкоштовним інструментом з відкритим кодом для запису екрану та стрімінгу. Він не підтримує ні роботу з PDF, ні готові аудіофайли, ні будь-яку автоматичну синхронізацію – користувач повинен вручну перемикати слайди під час запису в режимі реального часу. OBS є інструментом захоплення екрану, а не монтажу.

Camtasia від TechSmith – повноцінний редактор скрінкастів із функцією запису. Він підтримує імпорт готового аудіо на часову шкалу, але розстановка слайдів у потрібні моменти часу є виключно ручною операцією. Після стратегічного альянсу з Panopto у 2024–2025 роках Camtasia отримав можливість прямого експорту до Panopto-бібліотеки, проте автоматична синхронізація між наявним аудіозаписом та слайдами так і не з'явилась.

Microsoft PowerPoint дозволяє записувати голосові нотатки до кожного слайду, але не підтримує імпорт зовнішнього аудіофайлу як єдиного цілого та не виконує жодного автоматичного зіставлення.

Loom є хмарним інструментом для швидкого запису відеоповідомлень через браузер. Він не підтримує ні PDF, ні зовнішнє аудіо, ні будь-яку синхронізацію.

Descript – хмарний редактор, орієнтований на редагування відео через редагування його транскрипту [20]. Підтримує готові аудіофайли, але синхронізація презентаційних слайдів не передбачена архітектурою продукту.

Synthesia генерує відео з аватаром на основі тексту, що є принципово іншою задачею – не синхронізацією наявних матеріалів.

Ranopto є найближчим аналогом серед розглянутих рішень. Платформа підтримує роботу з аудіо та частково з PDF і надає функцію «Capture» для синхронної презентації слайдів під час запису. Однак ключове обмеження полягає в тому, що Ranopto вимагає паралельного запису презентації та мовлення – завантаження вже готового аудіофайлу з автоматичною прив'язкою до слайдів не підтримується.

Google NotebookLM є окремим випадком, що заслуговує детального розгляду через свою популярність у академічному середовищі. Станом на 2025–2026 роки NotebookLM підтримує завантаження PDF, Google Docs, YouTube-посилань та аудіофайлів і здатний генерувати на їх основі Audio Overviews (подкаст-діалог двох AI-ведучих), а з Google I/O 2025 – також Video Overviews (візуальні презентації на основі документів). У грудні 2025 року Google почав тестувати Lecture Mode – генерацію 30-хвилинних одоголосих аудіолекцій у стилі університетської аудиторії. Однак принципова відмінність NotebookLM від задачі цієї роботи полягає в тому, що він генерує новий синтетичний контент – AI-ведучі, AI-голос, AI-сценарій – а не синхронізує реальний голос реального лектора з його реальними слайдами. Завантаження власного аудіозапису лекції та його автоматичне зіставлення зі слайдами PDF не є функціональністю NotebookLM [11], [21], [22].

Отже ринок програмного забезпечення для створення відеолекцій є досить зрілим [23]. Однак жоден з наявних продуктів не реалізує повністю автоматизований ланцюжок обробки готових матеріалів. Таким чином саме цю нішу заповнює розроблена система.

Порівняльний аналіз наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих систем генерації відеолекцій

| Система           | PDF      | Готове аудіо | Авто-синхронізація | Тип     | Вартість    |
|-------------------|----------|--------------|--------------------|---------|-------------|
| OBS Studio        | Ні       | Ні           | Ні                 | Десктоп | Безкоштовно |
| Camtasia          | Часткова | Так          | Ні (ручна)         | Десктоп | Платна      |
| PowerPoint        | Так      | Ні           | Ні                 | Десктоп | Платна      |
| Loom              | Ні       | Ні           | Ні                 | Хмара   | Freemium    |
| Descript          | Ні       | Так          | Ні (ручна)         | Хмара   | Платна      |
| Synthesia         | Ні       | Ні           | Н/Д                | Хмара   | Платна      |
| Panopto           | Часткова | Так          | Часткова           | Хмара   | Платна      |
| Google NotebookLM | Так      | Ні           | Ні (генерація)     | Хмара   | Freemium    |
| PodcastSlide      | Так      | Так          | Так (AI)           | Веб     | Відкрите    |

### 1.2.2 Методи автоматичного розпізнавання мовлення з пословними таймстемпами

Серед доступних на ринку ASR-рішень, що надають пословні таймстемпи, виділяють кілька основних підходів. Хмарні API-сервіси – OpenAI Whisper API, AssemblyAI та Deepgram – забезпечують транскрипцію через REST-запити без необхідності розгортання власної

інфраструктури. Локальні моделі – Whisper large-v3, faster-whisper – дозволяють обробляти аудіо на власних серверах, що важливо з точки зору конфіденційності даних, але потребують значних обчислювальних ресурсів.

OpenAI Whisper API у поєднанні з параметром `timestamp_granularities=["word"]` повертає об'єкти слів з атрибутами `.word`, `.start` та `.end`, що є необхідним і достатнім для побудови системи виявлення пауз. Точність пословних таймстемпів Whisper є прийнятною для практичних задач: типова похибка становить менше одної секунди на чистих аудіозаписах, що цілком задовольняє вимоги до точності визначення меж між слайдами. Deepgram Nova-3 та AssemblyAI Universal-2 є конкурентними альтернативами, однак вони поступаються Whisper за підтримкою менш поширених мов та гнучкістю налаштування формату відповіді.

Окремою проблемою при роботі з довгими аудіозаписами є обмеження розміру файлу в API. Рішення, реалізоване у системі, полягає у розбитті аудіо на 5-хвилинні чанки з подальшим коригуванням часових зміщень: таймстемпи кожного чанку збільшуються на тривалість усіх попередніх чанків, що забезпечує неперервну шкалу часу для всього запису.

### 1.2.3 Методи виявлення пауз у мовленні як маркерів структурних меж (VAD)

Виявлення активності мовлення (VAD – Voice Activity Detection) – клас алгоритмів, що визначають моменти присутності або відсутності людського мовлення у аудіосигналі [24], [25]. У класичній постановці VAD використовується для фільтрації тиші перед передачею аудіо на розпізнавання, що скорочує обсяг даних та знижує ймовірність галюцинацій моделі. Бібліотека Silero VAD є одним з найпоширеніших сучасних рішень для цієї задачі.

У контексті цієї роботи VAD застосовується у нетиповий спосіб: замість фільтрації тиші паузи між словами використовуються як семантичні маркери потенційних меж між слайдами. Логіка полягає в тому, що лектор, завершуючи розповідь про один слайд та переходячи до наступного, робить характерну паузу. Якщо тривалість міжслівного інтервалу перевищує встановлений поріг (у системі – 0,5 секунди, параметр PAUSE\_THRESHOLD), така пауза отримує унікальний ідентифікатор (Pause1, Pause2, ...) і вноситься у розмічений транскрипт.

Даний підхід є принципово відмінним від методів примусового вирівнювання (forced alignment), які використовуються, наприклад, у WhisperX. Forced alignment потребує повного тексту еталонної транскрипції та фонемної моделі для конкретної мови, тоді як запропонований метод виявлення пауз працює безпосередньо з таймстемпами слів від Whisper без додаткових моделей та залежностей.

#### 1.2.4 Порівняльний аналіз існуючих рішень для синхронізації презентацій з аудіо

Задача синхронізації наявного аудіозапису з слайдами презентації є порівняно новою і не має широко розповсюджених автоматизованих рішень. Найближчими за концепцією є такі підходи.

Ручна синхронізація в нелінійних відеоредакторах (Adobe Premiere Pro, DaVinci Resolve) є найпоширенішим методом у академічному середовищі. Редактор вручну розставляє кліпси зображень слайдів на часовій шкалі поруч із аудіодоріжкою, орієнтуючись на прослуховування запису. Цей процес є точним, але вкрай трудомістким.

Функція «Rehearse Timings» у PowerPoint дозволяє автоматично зберігати час, витрачений на кожен слайд під час репетиції, але вимагає живого виступу або запису в режимі реального часу – використання готового аудіофайлу не передбачено.

Деякі платформи для корпоративного навчання, такі як iSpring Suite, підтримують синхронізацію слайдів PowerPoint з аудіо, однак механізм є напівавтоматичним: користувач вказує точки переходу вручну, слухаючи запис.

Жодне з перерахованих рішень не використовує семантичний аналіз тексту для автоматичного визначення моментів зміни слайдів – саме в цьому полягає наукова новизна підходу, реалізованого у системі PodcastSlide.

### 1.3 Актуальність задачі та обмеження існуючих рішень

#### 1.3.1 Проблема автоматизації виробництва навчального відеоконтенту

Зростання глобального ринку e-learning супроводжується загостренням проблеми масштабування виробництва якісного відеоконтенту. Університети та освітні організації стикаються з парадоксом: попит на відеолекції зростає експоненційно, тоді як можливості викладачів з їх виробництва залишаються обмеженими. Більшість університетських викладачів не є фахівцями з відеомонтажу, а залучення технічного персоналу або аутсорсинг монтажу суттєво підвищують вартість і терміни виробництва кожної лекції.

Водночас у переважній більшості університетів вже існує значний масив «сирих» матеріалів: PDF-презентації, підготовлені для аудиторних занять, та аудіозаписи лекцій, зроблені під час очних або дистанційних занять у Zoom чи Microsoft Teams.

Ці два типи матеріалів є вихідними даними, які за умови наявності відповідного інструменту можуть бути автоматично перетворені у готову відеолекцію.

Таким чином, актуальність задачі визначається двома факторами, що діють одночасно: стрімким зростанням попиту на відеоконтент у сфері

дистанційної освіти та наявністю у викладачів готових вихідних матеріалів, які наразі не можуть бути ефективно перетворені у відео без значних трудозатрат.

### 1.3.2 Відсутність повністю автоматизованого рішення

Проведений аналіз ринкових рішень (підрозділ 1.2.1) демонструє, що станом на травень 2026 року не існує жодного загальнодоступного інструменту, який би реалізував повністю автоматизований ланцюжок обробки: PDF-презентація плюс готовий аудіофайл на вході – синхронізоване MP4-відео на виході. Найближчий аналог, Panopto, вимагає синхронного запису, тобто лектор повинен перемикає слайди в реальному часі під час запису. Це унеможлиблює використання вже наявних записів лекцій.

Відсутність такого рішення є не технічним, а концептуальним пробілом: задача вимагає семантичного розуміння того, про що говорить лектор у кожен момент часу, і зіставлення цього з текстом конкретного слайду. До появи великих мовних моделей з достатнім контекстним вікном та підтримкою багатомовних текстів ця задача не могла бути вирішена у загальному вигляді [26].

### 1.3.3 Потреба у семантичному аналізі для точного визначення меж між слайдами

Визначення моменту зміни слайду є задачею, що не може бути вирішена суто акустичними методами. Аналіз пауз у мовленні надає кандидатів на межі між слайдами, проте не дозволяє визначити, яка саме пауза відповідає переходу від одного слайду до наступного: у типовій 40- хвилинній лекції таких пауз може бути кілька сотень, тоді як слайдів – кілька десятків.

Методи на основі схожості текстів (TF-IDF, BM25, векторна семантична схожість) є кращими, але також недостатніми: лектор рідко дослівно цитує слайд, а часто перефразовує або доповнює його. Крім того, суміжні слайди нерідко присвячені одній темі, що ускладнює їх розмежування за ключовими словами.

Великі мовні моделі вирішують цю проблему завдяки здатності до контекстного семантичного аналізу: отримавши одночасно повний розмічений транскрипт та тексти всіх слайдів, модель здатна відтворити логіку розповіді лектора та визначити точки переходу між темами. Саме цей підхід реалізований у системі PodcastSlide через взаємодію з GPT-5 за допомогою спеціально сконструйованого промπτу.

#### 1.4 Постановка задачі та мета роботи

Метою роботи є розробка веб-застосунку PodcastSlide для автоматичної генерації синхронізованих відеолекцій на основі PDF-презентацій та аудіозаписів із застосуванням методів автоматичного розпізнавання мовлення (ASR) та семантичного аналізу на базі великих мовних моделей [6], [7].

Для досягнення поставленої мети необхідно вирішити такі задачі:

- провести аналіз існуючих систем генерації відеоконтенту та виявити їх обмеження в контексті автоматизованої синхронізації наявних PDF та аудіоматеріалів;
- дослідити методи транскрипції мовлення з пословними таймстемпами та методи виявлення пауз у мовленні як маркерів структурних меж між слайдами;
- розробити алгоритм семантичного маппінгу слайдів і транскрипту на базі великих мовних моделей з механізмом автоматичної валідації та корекції результатів;

– реалізувати серверну частину застосунку на основі Node.js + Express з Python-пайплайном обробки та клієнтський інтерфейс на Vanilla HTML/CSS/JS [23];

– провести тестування системи на реальних навчальних матеріалах та оцінити точність семантичного маппінгу й практичну ефективність рішення [7].

Вирішення зазначених задач дозволяє сформувати повний цикл автоматизованої обробки навчальних матеріалів – від завантаження вхідних файлів до отримання готового відеопродукту – без участі людини на жодному з проміжних етапів. Комплексний характер роботи, що поєднує дослідження методів обробки природної мови, комп'ютерного зору та відеосинтезу, визначає її міждисциплінарний характер на межі технологій штучного інтелекту, веб-розробки та освітніх технологій.

Об'єктом дослідження є процес автоматичного синтезу навчального відеоконтенту на основі різномірних медіаматеріалів.

Предметом дослідження є методи семантичного аналізу та синхронізації презентаційних і аудіоматеріалів у задачі автоматичної генерації відеолекцій.

## 2 МЕТОДИ ТА АРХІТЕКТУРА СИСТЕМИ

### 2.1 Загальна архітектура мультимодального конвеєра обробки

#### 2.1.1 Послідовність фаз обробки

Система PodcastSlide реалізована як конвеєр послідовних фаз обробки, кожна з яких виконує чітко окреслену функцію та передає результати наступній через персистентний JSON-файл стану. Такий підхід забезпечує відновлюваність при збоях, можливість повторного виконання окремих фаз без повного перезапуску та зручність налагодження через ізоляцію кожного кроку.

Перша фаза – `slides` – виконує підрахунок сторінок завантаженого PDF та розбір виразу діапазону слайдів, введеного користувачем (наприклад, «1–10, 15, 21»). Результатом є список індексів обраних слайдів, який зберігається у стані сесії.

Друга фаза – `process` – є центральною і включає п'ять послідовних кроків: витяг тексту з PDF, транскрипцію аудіо з пословними таймстемпами, виявлення пауз між словами, побудову розміченого транскрипту та семантичний маппінг слайдів. Результати цієї фази – словник таймстемпів пауз та список прив'язок «слайд → пауза» – записуються у JSON state-файл.

Третя фаза – `sync` – читає state-файл та обчислює таблицю синхронізації: для кожного слайду визначається його часовий сегмент у форматі `{slide, start_sec, end_sec}`. Ця таблиця відображається у веб-інтерфейсі для перевірки користувачем перед генерацією відео.

Четверта фаза – `generate` – рендерить слайди PDF у растрові зображення, нарізає аудіо відповідно до таблиці синхронізації та збирає фінальний MP4-файл. Загальна схема конвеєра зображена на рисунку 2.1.



Рисунок 2.1 – Загальна схема чотирифазного pipeline системи PodcastSlide

### 2.1.2 Взаємодія Node.js-сервера та Python-модулів через CLI-міст runner.py

Архітектурно система складається з двох рівнів: веб-сервера на Node.js (Express) та Python-пайплайну обробки. Взаємодія між ними організована через CLI-міст – файл runner.py, який приймає команду як перший аргумент командного рядка та виконує відповідну фазу обробки.

Node.js-сервер запускає Python-процес через `child_process.spawn`, передаючи шляхи до файлів та параметри як аргументи командного рядка. Python-процес повертає результати у вигляді JSON-рядків до `stdout`, а Node.js зчитує їх у режимі реального часу. Для коротких фаз (`slides`, `sync`) результат повертається одним JSON-рядком. Для тривалих фаз (`process`, `generate`) Python-процес надсилає послідовність JSON-рядків прогресу у форматі `{progress: N, message: "..."}` , які Node.js трансліює клієнту через SSE (Server-Sent Events). Така архітектура дозволяє відображати прогрес обробки у реальному часі без блокування основного потоку сервера.

Перевагою CLI-мосту над альтернативними підходами (наприклад, REST API між двома серверами або черга повідомлень) є простота розгортання: вся система запускається однією командою `node server.js` без необхідності підтримки окремого Python-сервера.

### 2.1.3 Передача стану між фазами через JSON state-файл

Кожна сесія користувача ідентифікується унікальним UUID, що генерується при створенні нового проєкту. Усі файли сесії зберігаються у директорії `workdir/` з іменами, що включають UUID: `presentation_{uuid}.pdf`, `audio_{uuid}.wav`, `state_{uuid}.json`, `output_{uuid}.mp4`.

State-файл є центральним сховищем стану між фазами та містить: шлях до PDF, список обраних слайдів, тексти всіх слайдів, словник таймстемпів пауз та результати мапінгу. Такий підхід дозволяє фазам `sync`

та generate виконуватись незалежно від process – після завершення process результати залишаються персистентними та доступними для повторного використання без повторної транскрипції та маппінгу.

#### 2.1.4 Конфігурація системних параметрів

Усі глобальні параметри системи зосереджені у файлі config.py, що забезпечує єдину точку налаштування без необхідності модифікації коду окремих модулів. Конфігурація включає такі ключові параметри, як наведено у таблиці 2.1.

Таблиця 2.1 – Конфігураційні параметри системи PodcastSlide

| Параметр        | Значення     | Призначення                                   |
|-----------------|--------------|-----------------------------------------------|
| FPS             | 24           | Частота кадрів вихідного відео                |
| RESOLUTION      | (1920, 1080) | Роздільна здатність вихідного відео (Full HD) |
| WHISPER_MODEL   | whisper-1    | Модель OpenAI для транскрипції аудіо          |
| GPT_MODEL       | GPT-4O-mini  | Модель для OCR та допоміжних задач            |
| GPT_MATCH_MODEL | gpt-5        | Модель для семантичного маппінгу слайдів      |
| PAUSE_THRESHOLD | 0.5 с        | Мінімальна тривалість паузи між словами       |

Значення PAUSE\_THRESHOLD=0.5 секунди є результатом практичного підбору: менші значення породжують надмірну кількість маркерів (сотні пауз при природному диханні між словами), тоді як більші

значення пропускають реальні межі між слайдами при швидкому темпі мовлення.

## 2.2 Методи обробки презентаційних матеріалів

### 2.2.1 Витяг тексту з PDF-документів засобами PyPDF2

PDF-документи є складноструктурованими файлами, що містять як нативний текстовий шар (якщо документ був створений програмно), так і растрові зображення (якщо документ є скан-копією або слайди містять текст, вбудований у графіку). Для витягу нативного тексту у системі використовується бібліотека PyPDF2, яка надає клас PdfReader з методом `page.extract_text()` для кожної сторінки.

Метод `extract_text()` повертає рядок тексту, що містить усі текстові об'єкти сторінки у порядку їх розміщення у PDF-потоці. Однак для презентаційних слайдів цей порядок не завжди відповідає логічній послідовності читання: заголовок, що розміщений у верхній частині слайду, не обов'язково буде першим у потоці PDF. Незважаючи на цей недолік, отриманий текст є достатнім для задачі семантичного маппінгу, оскільки GPT-5 розуміє текст незалежно від порядку токенів за умови збереження ключових слів та термінів.

### 2.2.2 Порогова логіка перемикавання на OCR при нестачі нативного тексту

Значна частина навчальних презентацій містить слайди, де текст є частиною графічного елементу: діаграми, скріншоти, формули у вигляді зображень, слайди з мінімальним текстом та великою ілюстрацією. Для таких слайдів `extract_text()` повертає порожній рядок або рядок з кількома символами (наприклад, номер сторінки).

Для виявлення таких випадків у системі реалізована порогова логіка: якщо довжина витягнутого тексту складає менше 50 символів, сторінка вважається «переважно графічною» і для неї запускається OCR-аналіз засобами GPT-4o Vision. Поріг у 50 символів є достатньо малим, щоб не пропускати слайди з коротким, але інформативним текстом (наприклад, «Дякую за увагу!»), і достатньо великим, щоб відфільтровувати слайди, де `extract_text()` повертає лише артефакти парсингу.

### 2.2.3 OCR-аналіз графічних слайдів засобами GPT-4O Vision

Для слайдів, що не пройшли пороговий фільтр, система виконує OCR через GPT-4o Vision API. Процес включає кілька кроків: сторінка PDF рендериться у растрове зображення засобами PyMuPDF з масштабом  $0.5 \times 0.5$  (зменшення для економії токенів), конвертується у JPEG з якістю 60%, кодується у base64 та надсилається до API разом з текстовим запитом «Extract all text from this image. Return only the text content, no explanations».

GPT-4o Vision демонструє значно вищу точність OCR порівняно з класичними рішеннями (Tesseract, Google Vision API) для слайдів із складним форматуванням: нестандартними шрифтами, текстом поверх зображень, рукописними нотатками. Принципова перевага полягає у здатності моделі розуміти контекст: вона коректно розпізнає математичні формули, таблиці та структуровані списки, тоді як традиційні OCR-системи часто порушують структуру при їх розпізнаванні.

### 2.2.4 Рендеринг сторінок PDF у растрові зображення засобами PyMuPDF

На етапі генерації відео кожен слайд повинен бути представлений у вигляді растрового зображення з роздільною здатністю  $1920 \times 1080$

пікселів (Full HD). Для цього використовується PyMuPDF (бібліотека `fitz`) – високопродуктивна Python-бібліотека, побудована на основі С-рушія MuPDF. Станом на 2026 рік актуальна версія PyMuPDF (1.27.x) рекомендує використовувати імпорт `import pymupdf`, зберігаючи при цьому зворотну сумісність з `import fitz`.

Схема рендерингу слайду наведена на рисунку 2.2.

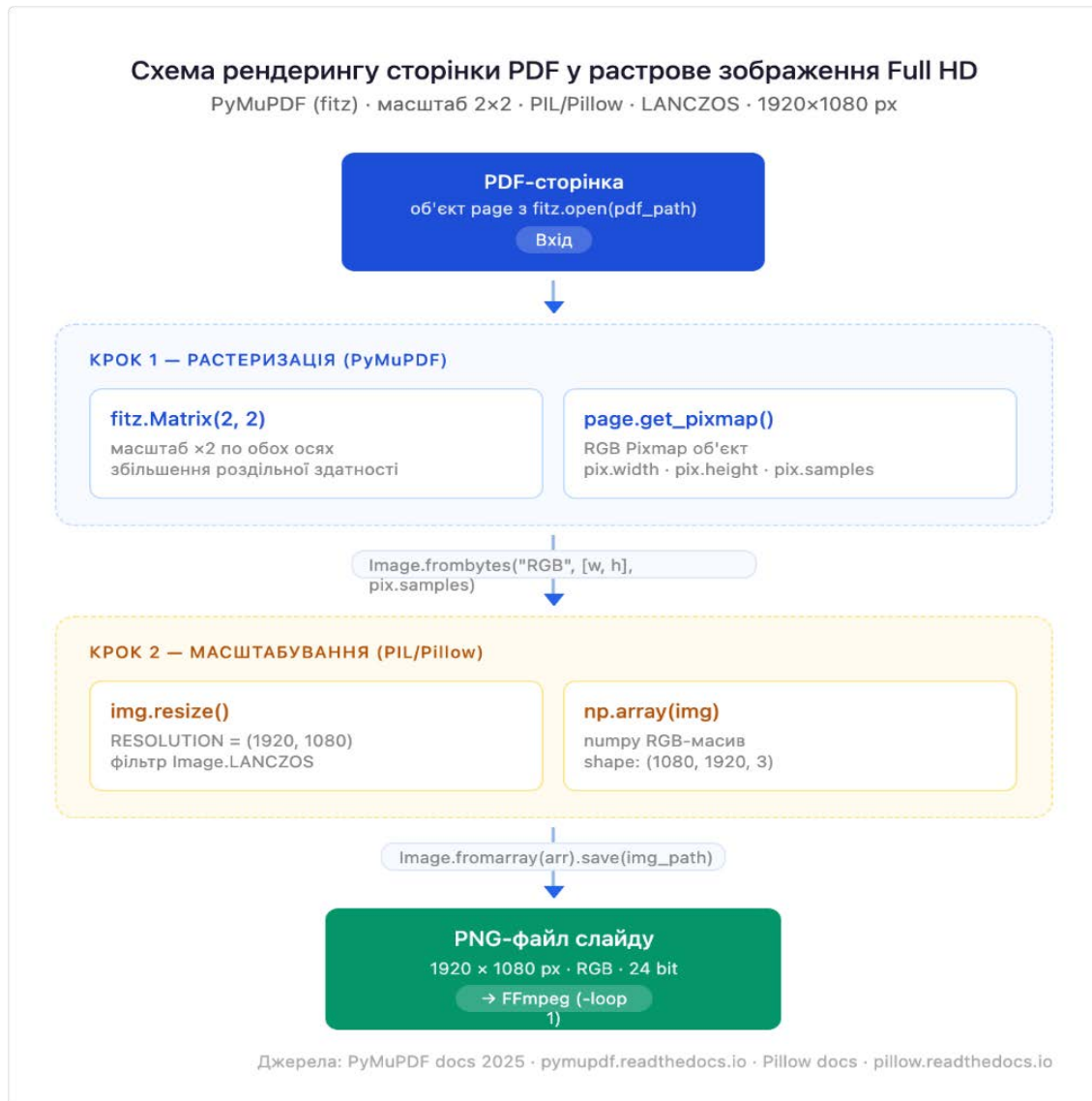


Рисунок 2.2 – Схема рендерингу сторінки PDF у растрове зображення Full HD

Рендеринг виконується методом `page.get_pixmap(matrix=fitz.Matrix(2,2))`, де матриця масштабування  $2 \times 2$  збільшує зображення вдвічі відносно стандартного розміру PDF-сторінки для забезпечення достатньої роздільної здатності. Отриманий об'єкт `Pixmap` конвертується у зображення `PIL/Pillow`, яке потім змінює розмір до точних значень `RESOLUTION = (1920, 1080)` з використанням фільтру `LANCZOS` для найвищої якості масштабування.

## 2.3 Методи автоматичного розпізнавання та сегментації мовлення

### 2.3.1 Транскрипція аудіо з пословними таймстемпами засобами Whisper API

OpenAI Whisper – автоматична система розпізнавання мовлення (ASR), натренована на 680 000 годинах різноманітних мультимовних аудіоматеріалів. За результатами бенчмарку `MLPerf Inference 2025`, Whisper демонструє найнижчий показник похибки слів (WER) серед загальнодоступних ASR-моделей, скоротивши WER порівняно з попереднім еталоном більш ніж на 72%.

Для отримання пословних таймстемпів система використовує OpenAI Whisper API з параметром `response_format="verbose_json"` та `timestamp_granularities=["word"]`. У відповіді кожне слово представлено об'єктом з атрибутами `.word` (текст слова), `.start` (час початку у секундах від початку файлу) та `.end` (час закінчення). Саме ці атрибути є вхідними даними для алгоритму виявлення пауз.

### 2.3.2 Чанкова обробка аудіо та корекція часових зміщень між чанками

Whisper API має обмеження на розмір завантажуваного файлу (25 МБ), що унеможливорює пряму передачу тривалих аудіозаписів.

Для вирішення цієї проблеми система реалізує чанкову обробку: аудіо завантажується у пам'ять, конвертується у моно-формат зі частотою дискретизації 16 кГц та розбивається на фрагменти тривалістю 5 хвилин.

Ключовою технічною проблемою чанкової обробки є зміщення таймстемпів: Whisper повертає час відносно початку кожного чанку, а не відносно початку всього запису.

Для збереження точності ресемплінгу аудіо система використовує Ffmpeg замість numru decimate: бібліотечний метод numru- decimation вносив систематичну похибку від 8 до 10 секунд на тривалих записах через накопичення похибки округлення при зниженні частоти дискретизації.

### 2.3.3 Алгоритм виявлення пауз за пороговим значенням міжслівного інтервалу

Виявлення пауз є принципово важливим кроком, оскільки саме паузи слугують кандидатами на межі між слайдами. Алгоритм послідовно обходить список слів та порівнює кожен міжслівний інтервал з порогом `PAUSE_THRESHOLD = 0.5` секунди. При виявленні інтервалу, що перевищує поріг, фіксується пауза з такими атрибутами: унікальний ідентифікатор (`Pause1`, `Pause2`, ...), часова мітка (момент закінчення попереднього слова) та індекс слова після якого вона розміщена.

Результатом роботи алгоритму для типової 40-хвилинної лекції є список з кількох десятків до кількох сотень пауз. У тестовому запуску на записі тривалістю 23,5 хвилини було виявлено 196 пауз при 15 слайдах – тобто в середньому близько 13 пауз на один слайд. Це підтверджує ключову особливість задачі: не кожна пауза є межею між слайдами, оскільки лектор природно робить зупинки всередині розповіді про один слайд. Відповідно, завдання семантичного мапінгу, що виконується на наступному кроці,

полягає у виборі лише тих пауз, які відповідають реальним переходам між темами.

Блок-схема алгоритму виявлення пауз наведена на рисунку 2.3.

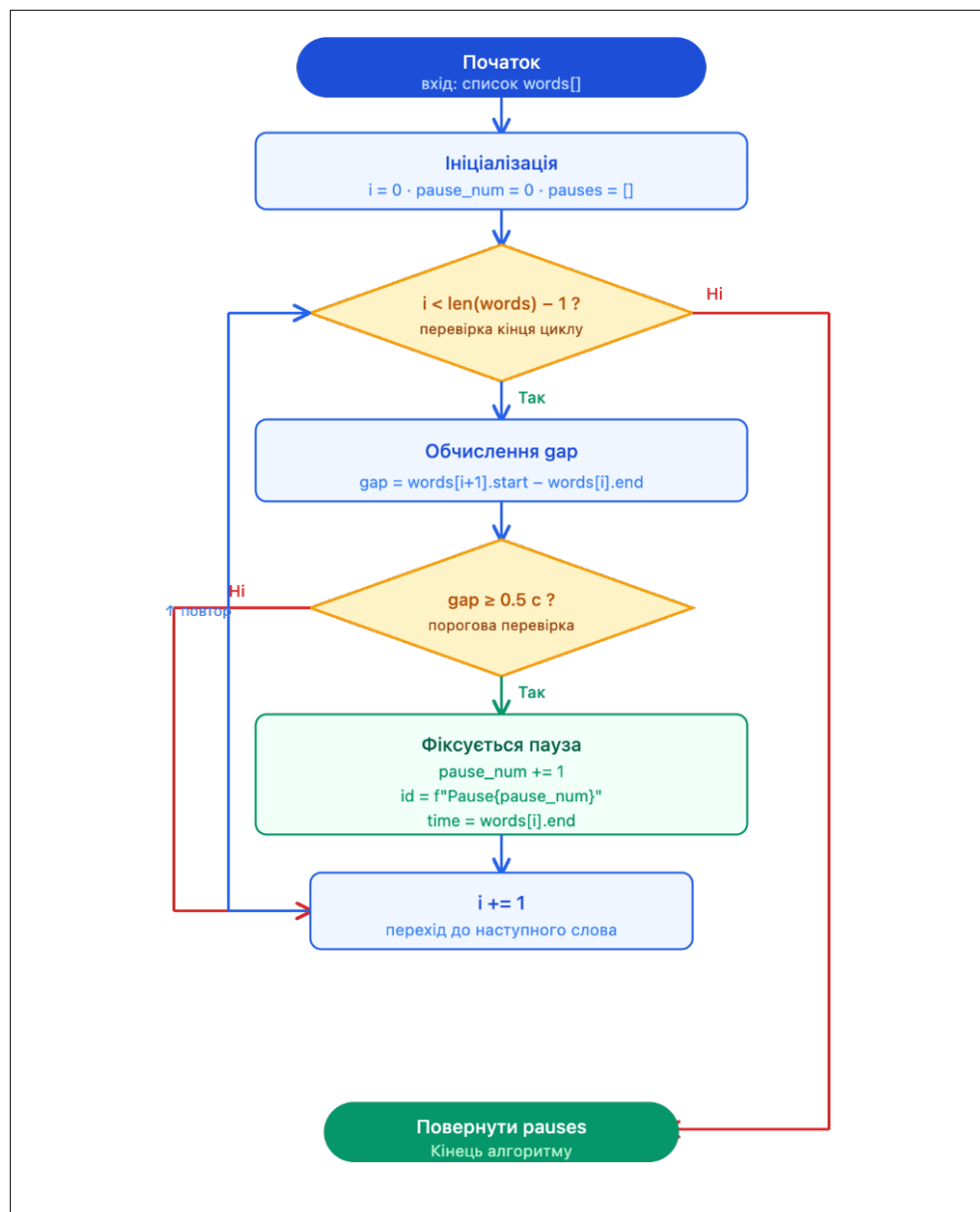


Рисунок 2.3 – Блок-схема алгоритму виявлення пауз між словами транскрипту

Результатом роботи алгоритму для типової 40-хвилинної лекції є список з кількох десятків до кількох сотень пауз. Наприклад, у тестовому запуску на лекції тривалістю 23,5 хвилини було виявлено 196 пауз

при 15 слайдах – тобто в середньому близько 13 пауз на один слайд. Це демонструє, що не кожна пауза є межею між слайдами: завдання семантичного маппінгу полягає у виборі лише тих пауз, які відповідають реальним переходам між темами.

#### 2.3.4 Побудова розміченого транскрипту з іменованими маркерами пауз

Після виявлення пауз будується розмічений транскрипт – рядок, у якому слова чергуються з маркерами пауз. Маркери вставляються після слова, за яким іде пауза: "Сьогодні ми розглянемо <Pause1> основні концепції машинного навчання <Pause2> зокрема нейронні мережі <Pause3> ...".

Така структура дозволяє GPT-5 одночасно бачити повний текст мовлення та точки розрізу між потенційними сегментами. Модель не потребує знання реальних часових міток – вони зберігаються окремо у словнику `pause_times: {pause_id → seconds}` та використовуються лише після того, як модель визначить відповідні ідентифікатори пауз для кожного слайду.

### 2.4 Методи семантичного маппінгу слайдів і транскрипту

#### 2.4.1 Формування структурованого промпту для GPT-5

Семантичний маппінг є ключовим алгоритмом системи. Його задача – для кожного слайду визначити ідентифікатор паузи, після якої лектор завершує розповідь про цей слайд і переходить до наступного. Задача формулюється як задача класифікації: кожному слайду з  $N$  слайдів необхідно зіставити одну з  $M$  пауз.

Для вирішення цієї задачі система формує структурований промпт, що містить: загальний контекст (тривалість аудіо, кількість слайдів, кількість пауз), тексти всіх слайдів пронумерованим списком, повний розмічений транскрипт та інструкції щодо формату відповіді (виключно JSON-об'єкт без пояснень).

Промпт включає ключові правила для моделі: сканувати транскрипт зліва направо та зіставляти слова з текстом слайдів за ключовими словами, концепціями та термінологією; межа для слайду N повинна бути строго після межі для слайду N-1; не нав'язувати рівномірний розподіл – слайди мають різну тривалість; при суміжних слайдах з однаковим заголовком шукати зміщення підтеми.

Вибір GPT-5 як моделі для маппінгу обумовлений необхідністю обробки великого контекстного вікна (повний транскрипт тривалої лекції разом з текстами всіх слайдів може складати десятки тисяч токенів) та вимогами до семантичного розуміння природної мови. GPT-4O-mini, що використовується для OCR, не забезпечує достатньої якості маппінгу на складних лекціях із суміжними темами.

#### 2.4.2 Визначення меж слайдів через зіставлення ключових слів і термінів

Лектор у процесі читання лекції, як правило, не відтворює текст слайду дослівно, а пояснює, розширює та перефразовує його. Тому задача зіставлення вирішується не через пошук точних збігів рядків, а через семантичне розуміння змісту.

Наприклад, вислів «двісті п'ятдесят три мільярди доларів у двадцять третьому році» у транскрипті відповідає тексту «\$253B – обсяг ринку e - learning 2023» на слайді; GPT-5 коректно встановлює цей зв'язок без явних текстових збігів.

Особливо складним випадком є суміжні слайди з однаковим заголовком – наприклад, кілька слайдів поспіль із заголовком «Актуальність теми» або «Для допитливих». У таких випадках модель орієнтується не на заголовок, а на специфіку підтеми кожного слайду: зміну питання, нової концепції або нового числового показника.

#### 2.4.3 Алгоритм валідації маппінгу за часовим розподілом аудіо

Результат, повернутий GPT-5, проходить обов'язкову валідацію, реалізовану у функції `_validate_and_fix_by_time`. Алгоритм обчислює тривалість кожного сегменту на основі таймстемпів пауз та перевіряє відповідність отриманого розподілу часовим евристикам. Зокрема, жоден не-останній слайд не повинен бути коротшим за 0,3 від середньої тривалості ( $\text{min\_dur} < \text{avg} \times 0.30$ ), жоден не-останній слайд не повинен перевищувати потрібну середню тривалість ( $\text{max\_dur} > \text{avg} \times 3.0$ ), а останній слайд, який завжди триває до кінця аудіо, не повинен перевищувати 2,5 від середньої тривалості ( $\text{last\_dur} > \text{avg} \times 2.5$ ). Якщо хоча б одна з цих умов порушена – маппінг вважається невалідним, про що записується попередження у лог, і система переходить до автоматичної корекції.

Якщо хоча б одна умова порушена – маппінг вважається невалідним, про що записується попередження у лог. У такому разі система автоматично перерозподіляє межі рівномірно по часовій осі: для кожного слайду і цільовий момент часу визначається як  $(i+1) \times \text{audio\_duration} / N$ , де  $N$  – кількість слайдів, а з наявних пауз обирається найближча до цього моменту.

Така дворівнева стратегія – семантичний результат від GPT-5 з подальшою часовою перевіркою – є компромісом між точністю змістовного аналізу та надійністю системи в цілому. Семантичний маппінг є первинним і використовується завжди, коли розподіл часу виглядає природно. Часова корекція є резервним механізмом, що активується лише при виявленні

аномалій, і гарантує, що фінальне відео не міститиме аномально коротких або довгих слайдів навіть у випадку помилки моделі на складних лекціях із суміжними темами.

У тестовому запуску на 15 слайдах і 196 паузах жодна з трьох умов порушена не була – маппінг GPT-5 пройшов валідацію без корекції, що свідчить про достатню якість семантичного аналізу для реальних навчальних матеріалів.

Схема алгоритму валідації наведена на рисунку 2.4.

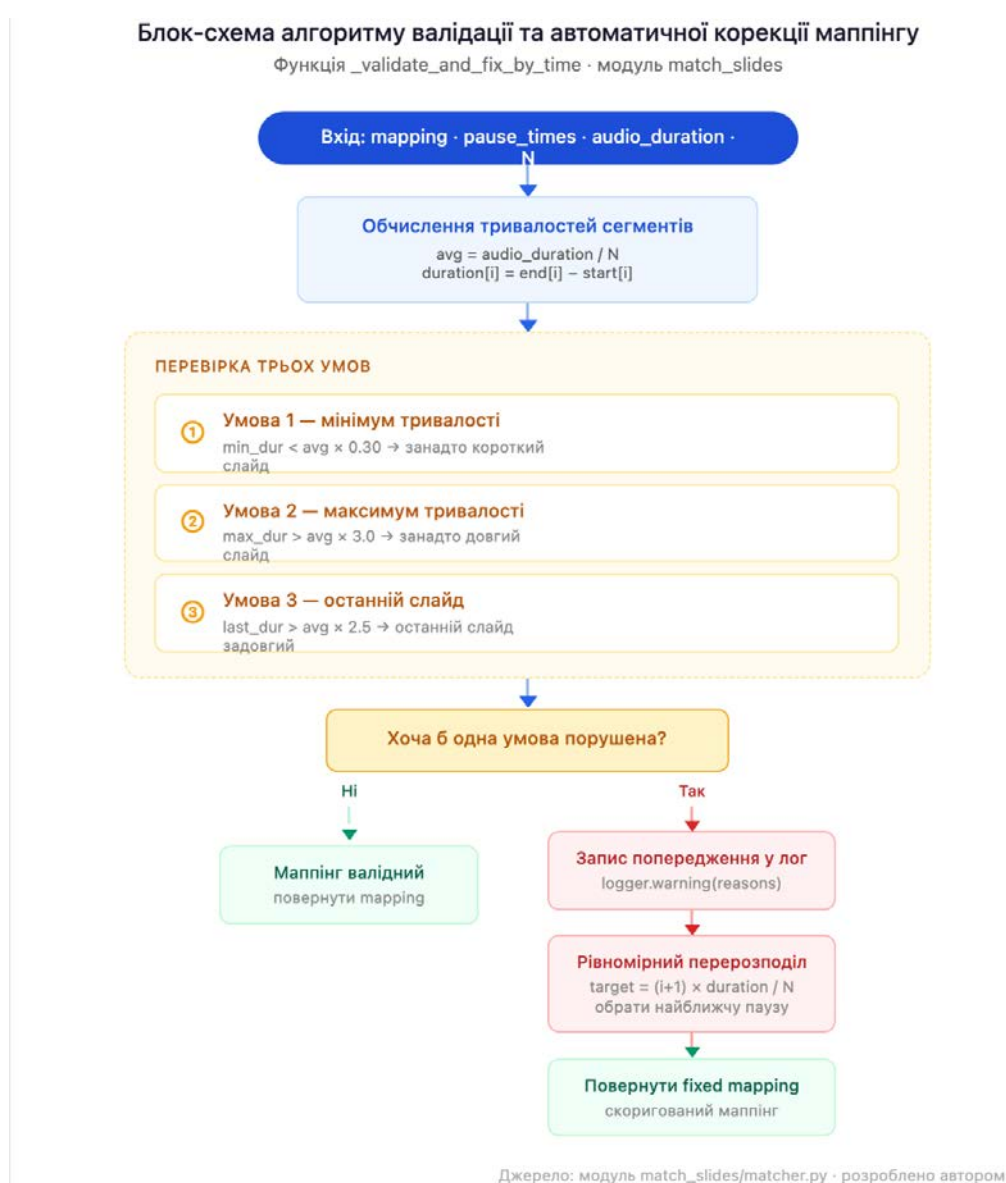


Рисунок 2.4 – Блок-схема алгоритму валідації та автоматичної корекції маппінгу

Таким чином навіть при помилці GPT-5 у складних випадках фінальне відео не міститиме аномально коротких чи довгих слайдів.

#### 2.4.4 JSON-логування результатів маппінгу

Після кожного успішного маппінгу система записує детальний debug-лог у файл `debug_match_slides.json`. Лог містить: мітку часу запуску, назву використаної моделі, тривалість аудіо, кількість пауз, тексти всіх слайдів, повний розмічений транскрипт, надісланий промпт, сирий JSON-відповідь моделі та фінальний маппінг з таймстемпами та відсотками від загальної тривалості.

Такий підхід до логування дозволяє ретроспективно аналізувати якість маппінгу, виявляти систематичні помилки моделі на певних типах слайдів та порівнювати результати різних моделей без повторного запуску повного пайплайну.

### 2.5 Методи синтезу відеоматеріалу

#### 2.5.1 Посегментний рендеринг відеокліпів

Після отримання валідного маппінгу система будує таблицю часових сегментів: слайд  $i$  охоплює аудіо від моменту закінчення паузи  $i-1$  до моменту початку паузи  $i$  (або до кінця аудіо для останнього слайду). Кожен сегмент рендериться як незалежний MP4-кліп у тимчасовій директорії.

Процес рендерингу одного кліпу включає три послідовні FFmpeg - команди. Спочатку зображення слайду зберігається як PNG-файл. Потім виконується точне нарізання аудіо за таймстемпами паузи з конвертацією у AAC-формат. Нарешті, статичне зображення та аудіофрагмент об'єднуються у відеокліп з кодуванням H.264 (кодек libx264, пресет fast, піксельний формат yuv420p). Кожен такий кліп починається з

примусового ключового кадру (`-force_key_frames expr:eq(n,0)`), що є обов'язковою умовою для подальшої конкатенації без повторного кодування.

### 2.5.2 Точне нарізання аудіо за таймстемпами пауз (sample-accurate)

Точність нарізання аудіо є критично важливою для синхронізації: відхилення навіть у кілька сотень мілісекунд між закінченням слайду та початком наступного помітне на слух.

Ключовим є розміщення прапора `-ss (seek)` перед `-i (input)`: у такому випадку FFmpeg виконує пошук на рівні потоку даних (stream-level seek) до декодування, а не після. Для WAV/PCM-аудіо це забезпечує точність на рівні окремого семплу (sample-accurate seek), тоді як seek після `-i` може мати похибку до кількох сотень мілісекунд для стиснених форматів.

### 2.5.3 Конкатенація кліпів без повторного кодування (stream copy)

Після рендерингу усіх кліпів система об'єднує їх у фінальний MP4 файл.

Параметр `-c copy` означає копіювання потоків без повторного кодування (stream copy): відеодані кліпів переносяться у фінальний файл побайтово, без декодування та повторного кодування. Це дає дві переваги: по-перше, конкатенація виконується майже миттєво незалежно від тривалості відео; по-друге, відсутнє накопичення якісних втрат від повторного стиснення.

Умовою коректної stream-copy конкатенації є однаковість параметрів кодування у всіх кліпах (кодек, роздільна здатність, частота кадрів, піксельний формат). Оскільки всі кліпи рендеруються однаковою FFmpeg-командою з фіксованими параметрами, ця умова

гарантовано виконується. Загальна схема синтезу відео наведена на рисунку 2.5.

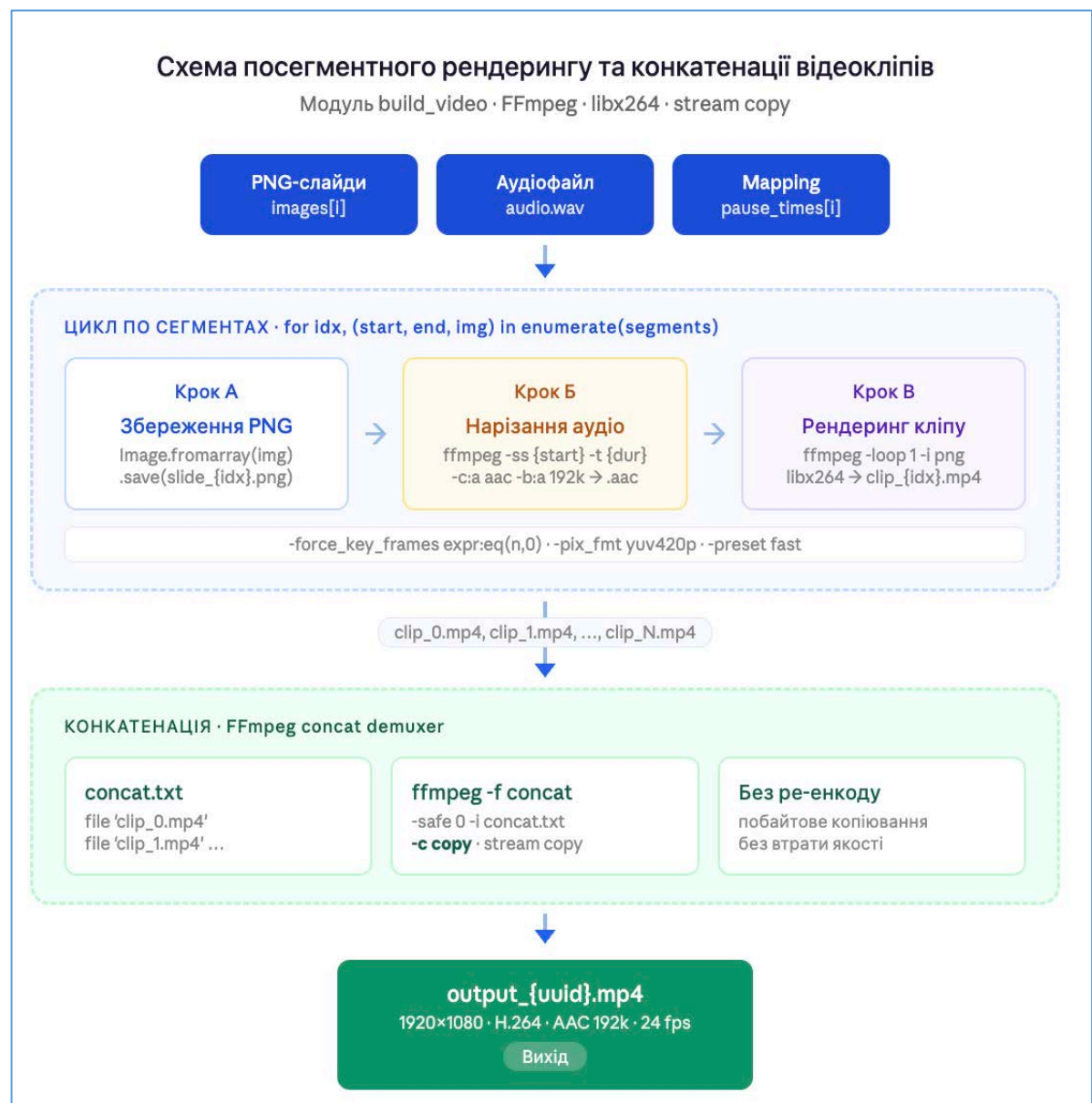


Рисунок 2.5 – Схема посегментного рендерингу та конкатенації відеокліпів

2.6 Обґрунтування вибору технологій та порівняльний аналіз альтернатив

Вибір технологічного стеку системи визначався трьома критеріями: точністю виконання ключових функцій, продуктивністю обробки та

мінімальністю зовнішніх залежностей. Порівняльний аналіз обраних рішень та їх альтернатив наведено у таблиці 2.2.

Таблиця 2.2 – Порівняльний аналіз обраних технологій та альтернатив

| Компонент          | Обраний варіант    | Альтернативи             | Обґрунтування вибору                                 |
|--------------------|--------------------|--------------------------|------------------------------------------------------|
| Backend-фреймворк  | Node.js + Express  | FastAPI, Flask           | SSE-стрімінг, spawn дочірніх процесів без блокування |
| ASR                | OpenAI Whisper API | AssemblyAI, Deepgram     | Точність, підтримка word timestamps, мультимовність  |
| Семантичний аналіз | GPT-5              | Claude, локальні LLM     | Якість мапінгу, контекстне вікно, JSON-режим         |
| OCR                | GPT-4O Vision      | Tesseract, Google Vision | Структурне розуміння слайдів, таблиці, формули       |
| Рендеринг відео    | FFmpeg             | MoviePy, OpenCV          | Швидкість, stream copy, sample-accurate seek         |
| PDF → зображення   | PyMuPDF (fitz)     | pdf2image, Pillow        | Швидкість C-рушія, точність рендерингу               |
| PDF → текст        | PyPDF2             | pdfminer, pymupdf        | Простота API, достатня точність для мапінгу          |
| Frontend           | Vanilla HTML/JS    | React, Vue               | Відсутність залежностей, один HTML-файл              |

Node.js з Express обрано для серверної частини замість Python FastAPI насамперед через природну підтримку асинхронного стрімінгу SSE та запуску дочірніх процесів (`child_process.spawn`) без блокування основного циклу подій. FFmpeg є галузевим стандартом для відеообробки з відкритим кодом: він підтримує практично всі відеоформати, забезпечує sample-accurate seek для WAV/PCM та виконує stream-copy конкатенацію у рази швидше за MoviePy чи OpenCV. PyMuPDF обрано для рендерингу PDF у зображення через його C-рушій MuPDF, що забезпечує значно вищу швидкість порівняно з Python-бібліотеками pdf2image та Pillow при однаковій якості растеризації.

Загалом обраний технологічний стек формує цілісну екосистему, де кожен компонент вирішує конкретну підзадачу з максимальною ефективністю: Node.js керує веб-шаром та міжпроцесною комунікацією, Python-модулі виконують обчислювально складні задачі аналізу та розпізнавання, а FFmpeg забезпечує відеосинтез на рівні системного рушія.

Такий гетерогенний розподіл відповідальності між технологіями дозволяє повністю уникнути критичних архітектурних та експлуатаційних компромісів, які неминуче виникають при спробах використання єдиного монолітного фреймворку для вирішення кардинально різних за своєю природою задач. Якби вся система була побудована виключно на Python, виникли б суттєві обмеження при масштабуванні веб-сервера через блокуючу природу мови та обмеження GIL (Global Interpreter Lock). Навпаки, спроба реалізувати алгоритми машинного навчання чи складну маніпуляцію медіафайлами безпосередньо всередині Node.js призвела б до перевантаження єдиного потоку виконання (Event Loop) обчислювальними задачами (CPU-bound tasks), що паралізувало б веб-інтерфейс застосунку.

Запропонований модульний підхід створює чітко розмежовані шари абстракції (Separation of Concerns). Він забезпечує можливість незалежного горизонтального вдосконалення, оптимізації та рефакторингу кожного окремого компонента без ризику дестабілізації або зміни загальної мікросервісної архітектури системи. Наприклад, логіка семантичного мапінгу в Python-модулі може бути модифікована під нові версії LLM, або параметри бітрейту у FFmpeg можуть бути змінені для оптимізації ваги відеофайлу без жодного втручання в код веб-сервера на Express.js. Як наслідок, архітектурний шаблон, побудований на синергії Node.js, Python та FFmpeg, гарантує системі високу гнучкість, життєздатність, легкість у супроводі та відмінний потенціал до масштабування в умовах інтенсивних академічних навантажень.

### 3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

#### 3.1 Архітектура програмного забезпечення

##### 3.1.1 Файлова організація проєкту та структура пакету steps

Програмна система PodcastSlide організована за принципом чіткого розділення відповідальності: кожен логічний крок обробки винесено в окремий Python-пакет у директорії steps/. Кожен такий пакет містить файл `__init__.py`, що реекспортує публічний інтерфейс модуля, та файл з реалізацією (наприклад, `builder.py`, `finder.py`, `matcher.py`). Така структура дозволяє імпортувати функції єдиним рядком (`from steps.find_pauses import find_pauses`) незалежно від внутрішньої організації модуля.

Такий підхід до організації коду відповідає принципу єдиної відповідальності: кожен модуль вирішує строго одну задачу і не знає нічого про реалізацію сусідніх модулів. Це суттєво спрощує налагодження системи – при виникненні помилки на конкретному кроці обробки розробник одразу знає, в якому файлі шукати причину. Крім того, модульна структура відкриває можливість незалежного вдосконалення окремих кроків: наприклад, заміна Whisper API на локальну модель або зміна логіки мапінгу потребуватиме модифікації лише одного пакету без жодних змін у решті системи. Інкапсуляція специфічних сторонніх залежностей, як-от SDK нейромереж чи бібліотек математичного аналізу, всередині відповідних підпакетів запобігає «засміченню» глобального простору імен та виникненню конфліктів версій.

Таке проектне рішення перетворює кодову базу застосунку на гнучку екосистему, яка мінімізує накопичення технічного боргу та демонструє високу адаптивність до мінливих технологічних вимог у сфері мультимедійного аналізу.

Файлова структура проєкту наведена на рисунку 3.1.

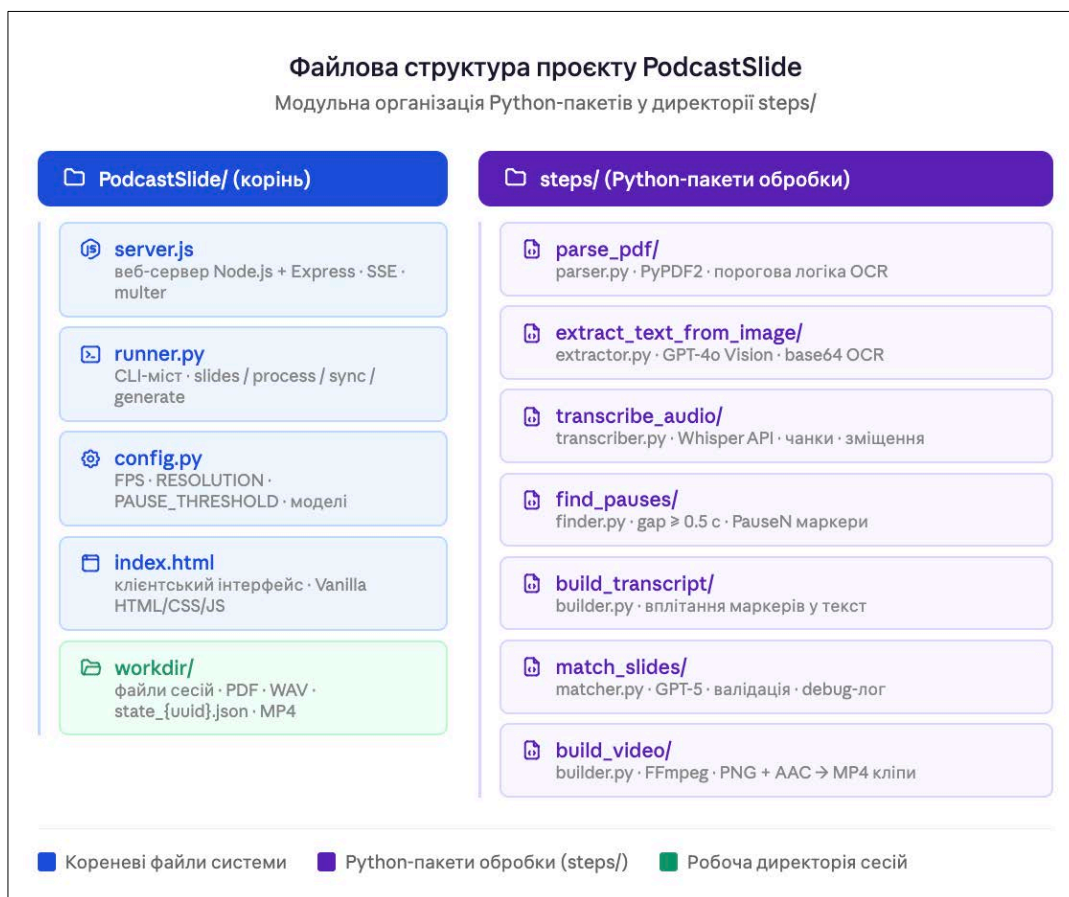


Рисунок 3.1 – Файлова структура проєкту PodcastSlide

Глобальна конфігурація системи зосереджена у файлі `config.py` та імпортується безпосередньо у кожному модулі через `from config import PAUSE_THRESHOLD`, що унеможливорює дублювання констант та спрощує налаштування системи.

### 3.1.2 CLI-міст `runner.py`. Команди `slides`, `process`, `sync`, `generate`

Файл `runner.py` є єдиною точкою входу для всіх Python-операцій. Він реалізує патерн «диспетчер команд»: перший аргумент командного рядка (`sys.argv[1]`) визначає, яку з чотирьох команд виконати. Кожна

команда отримує необхідні параметри через подальші аргументи (`sys.argv[2]`, `sys.argv[3]` тощо).

Усі результати виводяться у `stdout` у вигляді JSON-рядків через допоміжну функцію `emit()`. Для коротких команд (`slides`, `sync`) виводиться один JSON-рядок із результатом. Для тривалих команд (`process`, `generate`) виводяться проміжні рядки прогресу та фінальний рядок з прапором `done: true`. Реалізацію функції `emit` наведено у лістингу 3.1.

Лістинг 3.1 – Функція `emit` для передачі JSON-рядків до Node.js

```
def emit(obj):
    """Print a single JSON line to stdout (consumed by
server.js)."""
    print(json.dumps(obj, ensure_ascii=False), flush=True)
```

Параметр `flush=True` є критично важливим: без нього Python буферизує вивід і Node.js не отримує рядки прогресу до завершення процесу, що унеможлиблює відображення прогресу у реальному часі.

### 3.1.3 Передача прогресу від Python до клієнта через JSON-рядки та SSE

Архітектура передачі прогресу реалізована як дворівневий ланцюг: Python → Node.js → браузер. На рівні Python процес надсилає JSON-рядки до `stdout`. Node.js зчитує їх побайтово через `py.stdout.on('data', ...)`, розбиває за символом `\n`, парсить кожен рядок як JSON та негайно пересилає клієнту у форматі SSE (Server-Sent Events) через `res.write('data: ' + JSON.stringify(d) + '\n\n')`. Браузер підписується на SSE-потік через стандартний API `EventSource` та оновлює прогрес-бар при кожному отриманому повідомленні. Використання цієї технології дозволяє реалізувати легковажну односпрямовану ретрансляцію подій із низьким рівнем накладних витрат, що гарантує асинхронну синхронізацію

поточного стану обчислювального конвеєра з клієнтським інтерфейсом без надлишкового навантаження на мережеву інфраструктуру, властивого для традиційних методів періодичного опитування (polling). Реалізацію функції pipeToSSE наведено у лістингу 3.2.

Лістинг 3.2 – Функція pipeToSSE для трансляції прогресу Python-процесу клієнту

```
function pipeToSSE(py, res, onEvent) {
  let buf = '';
  py.stdout.on('data', chunk => {
    buf += chunk.toString();
    const lines = buf.split('\n');
    buf = lines.pop();
    for (const line of lines) {
      const t = line.trim();
      if (!t) continue;
      try {
        const d = JSON.parse(t);
        res.write(`data: ${JSON.stringify(d)}\n\n`);
        onEvent(d);
      } catch { /* non-JSON debug output - ignore */ }
    }
  });
  py.on('close', code => {
    if (code !== 0)
      res.write(`data: ${JSON.stringify(
        { error: `Python exited with code ${code}`
      })}\n\n`);
    res.end();
  });
}
```

Буферизація у змінній buf є необхідною, оскільки chunk даних від stdout не гарантовано закінчується на межі рядка: один JSON-рядок

може надійти двома окремими chunk-подіями. Накопичення у буфер та розбивка за \n з відкиданням неповного останнього елементу (`lines.pop()`) вирішує цю проблему.

## 3.2 Реалізація модуля обробки PDF

### 3.2.1 Парсинг тексту засобами PyPDF2 та порогова логіка перемикання на OCR

Модуль `parse_pdf` обробляє PDF-документ сторінка за сторінкою та повертає список словників у форматі `[{page: N, data: "текст"}, ...]`. Для кожної сторінки спочатку виконується спроба витягу нативного тексту через `page.extract_text()`. Якщо довжина отриманого тексту після видалення пробільних символів є меншою за 50 символів, модуль звертається до OCR-аналізу. Якщо OCR також не повертає результату, сторінка отримує текст-заглушку у форматі `[Slide N – mostly visual content]`, що інформує GPT-5 про наявність переважно візуального слайду.

Описана тришарова логіка – нативний текст, OCR, заглушка – забезпечує коректну обробку будь-якого типу PDF-презентації без ручного налаштування. Для типових навчальних презентацій більшість слайдів містять достатньо нативного тексту і проходять лише перший рівень обробки. Слайди з діаграмами, таблицями у вигляді зображень або мінімальним текстом автоматично потрапляють на другий рівень – OCR через GPT-4O Vision. Текст-заклушка третього рівня є важливим елементом надійності: вона запобігає ситуації, коли порожній рядок для переважно графічного слайду спричиняє помилку у GPT-5 під час семантичного мапінгу, замість цього явно повідомляючи модель про характер такого слайду.

Реалізацію порогової логіки наведено у лістингу 3.3.

### Лістинг 3.3 – Порогова логіка перемикання між нативним текстом та OCR

```

text = page.extract_text() or ''
text = text.strip()
if len(text) < 50:
    print(f"Page {page_num}: short text ({len(text)}
chars), "
        f"trying image extraction...")
    image_text = extract_text_from_image(file_path,
page_num)
    if image_text and image_text.strip():
        text = image_text.strip()
    else:
        if not text:
            text = f"[Slide {page_num} - mostly visual
content]"

```

#### 3.2.2 Інтеграція OCR через GPT-4O Vision з base64-кодуванням зображень

Модуль `extract_text_from_image` реалізує OCR через мультимодальний API GPT-4O-mini. Сторінка PDF рендериться у зображення за допомогою PyMuPDF з матрицею масштабування  $0.5 \times 0.5$  – зменшений розмір достатній для розпізнавання тексту та дозволяє суттєво знизити розмір base64-рядка, що передається до API. Зображення конвертується у JPEG-формат з якістю 60% через буфер у пам'яті (без запису тимчасових файлів на диск), після чого кодується у base64. Таке проектне рішення забезпечує раціональне використання мережевого трафіку та суттєво прискорює час отримання відповіді від моделі.

Реалізацію підготовки зображення наведено у лістингу 3.4.

### Лістинг 3.4 – Підготовка зображення слайду для передачі до GPT-4O Vision

```

pix = page.get_pixmap(matrix=fitz.Matrix(0.5, 0.5),
alpha=False)

image_buffer = BytesIO()
image_buffer.write(pix.tobytes('jpeg', jpg_quality=60))
image_buffer.seek(0)
image_base64 =
base64.b64encode(image_buffer.getvalue()).decode()

```

Сформований base64-рядок передається до API у повідомленні типу `image_url` з префіксом `data:image/jpeg;base64,`. Текстова частина запиту містить інструкцію: «Extract all text from this image. Return only the text content, no explanations» – відсутність пояснень у відповіді є важливою, оскільки результат OCR використовується як текстовий вміст слайду без подальшої фільтрації.

## 3.3 Реалізація модуля розпізнавання та сегментації мовлення (`transcribe_audio`, `find_pauses`, `build_transcript`)

### 3.3.1 Інтеграція Whisper API: чанкова обробка та корекція часових зміщень

Модуль `transcribe_audio` завантажує аудіофайл у пам'ять, конвертує його у моно-формат з частотою дискретизації 16 кГц та розбиває на 5-хвилинні чанки. Конвертація реалізована через бібліотеку `soundfile`: стерео-сигнал усереднюється по каналах (`data.mean(axis=1)`), частота дискретизації знижується децимацією з кроком `samplerate // 16000`, отримані значення масштабуються до діапазону 16-бітного PCM.

Кожен чанк зберігається у тимчасовий WAV-файл та передається до Whisper API. Після отримання відповіді тимчасовий файл негайно видаляється. Таймстемпи слів у відповіді зсуваються на величину накопиченого зміщення (offset), яке збільшується після кожного чанку на тривалість поточного чанку в секундах. Реалізацію корекції зміщень наведено у лістингу 3.5.

Лістинг 3.5 – Корекція часових зміщень таймстемпів між чанками

```
for word in resp.words:
    word.start += offset
    word.end   += offset
    all_words.append(word)

text_parts.append(resp.text)
offset   += len(chunk) / _TARGET_SR
last_resp = resp
```

Після обробки всіх чанків останній об'єкт відповіді Whisper доповнюється об'єднаним списком слів та конкатенованим текстом усіх чанків, після чого повертається як єдиний результат транскрипції.

### 3.3.2 Реалізація алгоритму виявлення пауз за порогом PAUSE\_THRESHOLD

Модуль `find_pauses` реалізує лінійний прохід по списку слів з обчисленням міжслівного інтервалу між кожною парою сусідніх слів. При перевищенні порогу `PAUSE_THRESHOLD` (0.5 секунди) фіксується пауза з інкрементальним номером та записується її часова мітка – момент закінчення попереднього слова (`.end`), а не початок наступного. Це рішення є принциповим: момент `.end` відповідає останньому звуку лектора перед

паузою, що є природною межею сегменту. Реалізацію алгоритму наведено у лістингу 3.6.

Лістинг 3.6 – Алгоритм виявлення пауз між словами транскрипту

```
for i in range(len(words) - 1):
    gap = words[i + 1].start - words[i].end
    if gap >= threshold:
        pause_num += 1
        pauses.append({
            'id':                f'Pause{pause_num}',
            'time':               round(words[i].end, 3),
            'after_word_index': i,
        })
```

Результат функції – список словників з полями `id`, `time` та `after_word_index`. Поле `after_word_index` є необхідним для наступного кроку – побудови розміченого транскрипту, де маркер паузи вставляється після слова з відповідним індексом.

### 3.3.3 Побудова розміченого транскрипту з маркерами пауз

Модуль `build_transcript` реалізує впітання маркерів пауз у текстовий рядок транскрипту. Для ефективної вставки спочатку будується словник зворотного пошуку: `insert_after = {p['after_word_index']: p['id'] for p in pauses}`. Потім виконується лінійний прохід по словах: після кожного слова перевіряється наявність його індексу у словнику, і якщо відповідна пауза знайдена – маркер додається до списку частин. Використання хеш-структури для індексації координат пауз дозволяє знизити обчислювальну складність цього етапу до лінійної залежності  $O(N)$  відносно загальної кількості слів у транскрипті. Реалізацію побудови транскрипту наведено у лістингу 3.7.

Лістинг 3.7 – Побудова розміченого транскрипту з маркерами пауз

```
insert_after = {p['after_word_index']: p['id'] for p in
pauses}
pause_times = {p['id']: p['time'] for p in pauses}

parts = []
for i, w in enumerate(words):
    parts.append(w.word.strip())
    if i in insert_after:
        parts.append(f'<{insert_after[i]}>')

transcript_str = ' '.join(p for p in parts if p)
```

Функція повертає два значення: рядок `transcript_str` для передачі до GPT-5 та словник `pause_times` для подальшого обчислення таймстемпів відеосегментів.

### 3.4 Реалізація модуля семантичного маппінгу (`match_slides`)

#### 3.4.1 Формування промпу та взаємодія з GPT-5

Модуль `match_slides` формує промпт з чотирьох складових: контекстного заголовку з метаданими (кількість слайдів, кількість пауз, тривалість аудіо), блоку текстів слайдів, повного розміченого транскрипту та інструкції щодо формату відповіді. Тексти слайдів форматуються як пронумерований блок: `Slide 1:\n{text}\n\nSlide 2:\n{text}....` Промпт завершується суворою інструкцією повернути виключно JSON-об'єкт без будь-яких пояснень.

Для взаємодії з API використовується параметр `response_format={"type": "json_object"}`, що гарантує отримання валідного JSON у відповіді незалежно від складності вхідних даних. Оскільки GPT-5 та деякі моделі-міркувачі не підтримують

параметр `temperature`, у коді реалізована перевірка: для стандартних моделей встановлюється `temperature=0` (детермінований вивід), для моделей зі списку `_NO_TEMP_MODELS` цей параметр не передається. Реалізацію виклику API наведено у лістингу 3.8.

Лістинг 3.8 – Виклик GPT-5 API з урахуванням підтримки параметра `temperature`

```
_NO_TEMP_MODELS = {"gpt-5", "gpt-5.5-pro", "o3", "o4-mini"}
chat_kwargs = dict(
    model=GPT_MATCH_MODEL,
    messages=[{"role": "user", "content": prompt}],
    response_format={"type": "json_object"},
)
if GPT_MATCH_MODEL not in _NO_TEMP_MODELS:
    chat_kwargs["temperature"] = 0

response = client.chat.completions.create(**chat_kwargs)
```

### 3.4.2 Реалізація алгоритму валідації та автоматичної корекції маппінгу

Функція `_validate_and_fix_by_time` реалізує двоетапну перевірку результату маппінгу. На першому етапі обчислюється тривалість кожного сегменту: для кожного слайду визначається момент початку (кінець попереднього сегменту) та момент закінчення (таймстемп призначеної паузи або кінець аудіо для останнього слайду). На другому етапі обчислені тривалості порівнюються з пороговими значеннями відносно середньої тривалості. Результат, повернутий GPT-5, проходить обов'язкову валідацію за трьома критеріями: жоден не-останній слайд не повинен бути коротшим за 0,3 від середньої тривалості ( $\text{min\_dur} < \text{avg} \times 0.30$ ), жоден не-останній слайд не повинен перевищувати потрібну середню тривалість ( $\text{max\_dur} > \text{avg} \times 3.0$ ), а останній слайд не повинен перевищувати 2,5 від середньої тривалості ( $\text{last\_dur} > \text{avg} \times 2.5$ ). Якщо хоча

б одна з цих умов порушена – маппінг вважається невалідним, про що записується попередження у лог, і система переходить до автоматичної корекції.

У разі невалідного маппінгу система виконує рівномірний перерозподіл меж: для кожного слайду з індексом  $i$  обчислюється цільовий момент часу  $\text{target} = (i+1) \times \text{audio\_duration} / N$ , після чого з відсортованого списку пауз обирається пауза з мінімальним відхиленням від цільового моменту. Реалізацію корекції наведено у лістингу 3.9.

Лістинг 3.9 – Рівномірний перерозподіл меж слайдів при невалідному маппінгу

```
for i in range(n):
    target = (i + 1) * audio_duration / n
    candidates = [(pid, t) for pid, t in sorted_pauses
                  if t > prev_t]
    if not candidates:
        fixed.append(sorted_pauses[-1][0])
        break
    pid, t = min(candidates, key=lambda x: abs(x[1] -
target))
    fixed.append(pid)
    prev_t = t
```

### 3.4.3 JSON-логування результатів у debug\_match\_slides.json

Після кожного виконання маппінгу система записує детальний лог у файл debug\_match\_slides.json у кореневій директорії проекту. Лог містить мітку часу у форматі UTC ISO 8601, назву використаної моделі, тривалість аудіо, кількість пауз і слайдів, повний промпт, сиру JSON-відповідь моделі та фінальний маппінг з таймстемпами та відсотками від загальної тривалості для кожного слайду. Запис логу обгорнутий у блок try/except – помилка запису не перериває основний процес обробки.

Наявність такого логу є важливим інструментом налагодження: аналізуючи відсотки `pct_of_audio` для кожного слайду, можна виявити аномальні розподіли (наприклад, перший слайд займає 40% аудіо або останній слайд – менше 1%) та скоригувати параметри системи.

### 3.5 Реалізація модуля відеогенерації (`build_video`)

#### 3.5.1 Рендеринг слайдів у PNG та посегментне нарізання аудіо в AAC

Модуль `build_video` починає роботу з побудови таблиці часових сегментів: для кожного слайду визначається пара (`start`, `end`) на основі словника `pause_times` та списку `mapping`. Сегменти з тривалістю менше 0.01 секунди пропускаються для уникнення порожніх кліпів. Усі тимчасові файли (PNG-зображення, AAC-аудіофрагменти, MP4-кліпи, файл конкатенації) зберігаються у тимчасовій директорії, що автоматично видаляється після завершення роботи через контекстний менеджер `tempfile.TemporaryDirectory`.

Для кожного сегменту послідовно виконуються три FFmpeg-команди. Перша зберігає зображення слайду як PNG. Друга нарізає аудіо з точним `seek` перед `-i` та конвертує у AAC з бітрейтом 192 кбіт/с. Третя команда об'єднує PNG та AAC у відеокліп. Згенеровані фрагменти надалі конкатенуються у підсумковий цілісний медіафайл. Така декомпозиція процесу мінімізує пікове навантаження на апаратні ресурси системи. Реалізацію третьої команди наведено у лістингу 3.10.

Такий дискретний підхід мінімізує пікове навантаження на обчислювальні потужності сервера. Проміжні медіафайли тимчасово ізолюються в окремому буферному каталозі. Після формування цілісного фрагмента конвеєр автоматично очищує цю робочу директорію. Це запобігає неконтрольованому накопиченню надлишкових даних у системі.

Лістинг 3.10 – FFmpeg-команда рендерингу одного відеокліпу зі слайду та аудіофрагменту

```
_run([
    "ffmpeg", "-y",
    "-loop", "1",
    "-framerate", str(FPS),
    "-i", img_path,
    "-i", audio_clip,
    "-c:v", "libx264", "-preset", "fast", "-pix_fmt",
    "yuv420p",
    "-force_key_frames", "expr:eq(n,0)",
    "-c:a", "copy",
    "-t", f"{duration:.6f}",
    clip_path,
])
```

Прапор `-loop 1` наказує FFmpeg нескінченно повторювати вхідне зображення як відеопотік. Тривалість кліпу обмежується прапором `-t` з точністю до мікросекунди (6 знаків після коми). Прапор `-force_key_frames expr:eq(n,0)` примусово встановлює ключовий кадр на нульовому кадрі кожного кліпу, що є обов'язковою умовою для подальшої безшовної конкатенації через `stream copy`.

### 3.5.2 Побудова concat-файлу та конкатенація через FFmpeg

Після рендерингу всіх кліпів формується текстовий файл `concat.txt`, кожен рядок якого містить шлях до одного кліпу у форматі `file '{path}'`. Цей файл передається до FFmpeg `concat demuxer`, який об'єднує кліпи у фінальний MP4 без повторного кодування. Реалізацію конкатенації наведено у лістингу 3.11.

Лістинг 3.11 – Формування concat-файлу та конкатенація кліпів без повторного кодування

```
concat_file = os.path.join(tmpdir, "concat.txt")
with open(concat_file, "w") as f:
    for cp in clip_paths:
        f.write(f"file '{cp}'\n")
_run([
    "ffmpeg", "-y",
    "-f", "concat", "-safe", "0",
    "-i", concat_file,
    "-c", "copy",
    output_path,
])
```

Прапор `-safe 0` дозволяє використовувати абсолютні шляхи у concat-файлі. Параметр `-c copy` означає копіювання всіх потоків (відео та аудіо) без декодування і повторного кодування. Допоміжна функція `_run` виконує FFmpeg-команду через `subprocess.run` з перехопленням `stderr`: у разі ненульового коду повернення вона підіймає `RuntimeError` з останніми 2000 символами виводу FFmpeg для діагностики.

## 3.6 Реалізація веб-застосунку

### 3.6.1 Серверна частина. Express.js, управління сесією та SSE-стрімінг

Веб-сервер реалізований на Node.js з використанням фреймворку Express. Для управління завантаженими файлами використовується middleware `multer`, що зберігає файли у директорії `workdir/` з автоматично генерованими іменами. Після збереження файл перейменовується відповідно до шаблону `{type}_{uuid}.{ext}` для однозначної прив'язки до сесії.

Система підтримує єдину сесію в пам'яті процесу – об'єкт session, що містить UUID проєкту та шляхи до завантажених файлів. При створенні нового проєкту (DELETE /api/reset) сесія замінюється новим об'єктом з новим UUID. Такий підхід є свідомим спрощенням для версії з одним користувачем; масштабування до багатокористувацького режиму потребуватиме зберігання сесій у Redis або подібному сховищі.

API-endpoints системи та їх призначення наведено у таблиці 3.1.

Таблиця 3.1 – API-endpoints веб-застосунку PodcastSlide

| Метод  | Endpoint           | Призначення                                   |
|--------|--------------------|-----------------------------------------------|
| POST   | /api/upload/pdf    | Завантаження PDF-презентації                  |
| POST   | /api/upload/audio  | Завантаження аудіофайлу                       |
| POST   | /api/slides/select | Розбір діапазону слайдів, підрахунок сторінок |
| GET    | /api/process       | Запуск основного пайплайну (SSE)              |
| GET    | /api/sync          | Отримання таблиці синхронізації               |
| GET    | /api/generate      | Генерація MP4-відео (SSE)                     |
| GET    | /api/export        | Завантаження готового відео                   |
| DELETE | /api/reset         | Скидання сесії та створення нового проєкту    |

3.6.2 Клієнтська частина. Інтерфейс завантаження, перегляду таймкодів та експорту MP4

Клієнтська частина реалізована як єдиний HTML-файл index.html на Vanilla HTML/CSS/JS без зовнішніх JavaScript-фреймворків. Стилзація виконана на CSS-змінних з акцентним кольором #AADF2E та темно-сірою базовою схемою.

Інтерфейс складається з чотирьох функціональних блоків. Загальний вигляд застосунку до завантаження файлів наведено на рисунку 3.2 – у початковому стані обидві зони завантаження відображаються з пунктирною рамкою, кнопки «Згенерувати відео» та «Експорт» є неактивними.

Рисунок 3.2 – Інтерфейс застосунку PodcastSlide у початковому стані

Перший блок – завантаження файлів – підтримує як кліки по кнопках, так і drag-and-drop: при перетягуванні файлу над зоною завантаження спрацьовують події `ondragover` та `ondrop`. Другий блок – вибір слайдів – містить текстове поле для введення діапазону; при введенні тексту спрацьовує `debounce`-таймер на 600 мс, після завершення якого автоматично виконується запит до `/api/slides/select`. Третій блок – таблиця синхронізації – відображає список рядків з часовими інтервалами та номерами слайдів після завершення фази `process`. На рисунку 3.3 показано інтерфейс після успішної обробки: обидва файли завантажені (індикатори зеленого кольору), таблиця синхронізації містить таймкоди для всіх 15 слайдів, розпочато генерацію відео. Користувачеві доступна функція інтерактивного редагування часових інтервалів безпосередньо у комірках таблиці для досягнення прецизійної міжкадрової синхронізації. Це забезпечує гнучку експертну верифікацію

результатів автоматичного мультимодального аналізу перед фінальним низькорівневим рендерингом відеопотоку.

PODCASTSLIDE

+ Новий проєкт

ПРЕЗЕНТАЦІЯ

Завантажити презентацію

Презентація.pdf

АУДІО

Завантажити аудіо

podcast (2).wav

Вибір слайдів

Наприклад: 1-10, 15, 21 (порожньо = всі)

Синхронізація

Оновити

|               |          |
|---------------|----------|
| 00:00 - 02:08 | Слайд 1  |
| 02:08 - 03:38 | Слайд 2  |
| 03:38 - 05:05 | Слайд 3  |
| 05:05 - 07:28 | Слайд 4  |
| 07:28 - 09:22 | Слайд 5  |
| 09:22 - 10:28 | Слайд 6  |
| 10:28 - 11:40 | Слайд 7  |
| 11:40 - 13:03 | Слайд 8  |
| 13:03 - 14:36 | Слайд 9  |
| 14:36 - 16:08 | Слайд 10 |
| 16:08 - 18:11 | Слайд 11 |
| 18:11 - 20:16 | Слайд 12 |
| 20:16 - 22:20 | Слайд 13 |
| 22:20 - 23:16 | Слайд 14 |
| 23:16 - 23:29 | Слайд 15 |

Згенерувати відео

Експорт

Рисунок 3.3 – Інтерфейс застосунку після обробки: таблиця синхронізації та початок генерації відео

Четвертий блок – прогрес та дії – містить прогрес-бар, що оновлюється через SSE, та кнопки «Згенерувати відео» і «Експорт». Кнопка «Експорт» залишається неактивною до завершення генерації відео.

Крупний план блоку керування під час рендерингу наведено на рисунку 3.4 – прогрес-бар відображає поточний відсоток виконання, а підпис показує поточну операцію та її числове значення.



Рисунок 3.4 – Прогрес-бар та кнопки керування під час генерації відео

Стан інтерфейсу управляється через об’єкт state з чотирма булевими прапорами: `pdfReady`, `audioReady`, `syncDone`, `videoDone`. Функція `refreshButtons()` викликається після кожної зміни стану та встановлює атрибут `disabled` для кнопок відповідно до поточних прапорів, що унеможливорює виконання операцій у неправильному порядку.

Такий підхід до керування станом є свідомо мінімалістичним: замість повноцінного фреймворку управління станом (Redux, Zustand або аналогів) використовується простий JavaScript-об’єкт з чотирма полями. Це рішення є виправданим для однокористувацького застосунку з лінійним сценарієм використання – користувач не може випадково запустити генерацію відео до завершення синхронізації, оскільки кнопка фізично недоступна.

Реалізація клієнтської частини на Vanilla HTML/CSS/JS також відповідає загальній архітектурній філософії проєкту: кожен компонент системи використовує найпростіший інструмент, достатній для вирішення задачі.

## 4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

### 4.1 Опис користувацького інтерфейсу та сценарії використання

Веб-застосунок PodcastSlide орієнтований на викладачів університетів та авторів освітнього контенту, які мають у своєму розпорядженні готову PDF-презентацію та аудіозапис лекції і прагнуть отримати синхронізоване відео без ручного монтажу. Типовий сценарій використання системи охоплює чотири послідовні дії користувача.

На першому кроці користувач завантажує PDF-презентацію та аудіофайл через відповідні зони інтерфейсу – перетягуванням або через стандартний діалог вибору файлу. Після успішного завантаження кожна зона набуває суцільної зеленої рамки, а індикатор статусу змінює колір з сірого на зелений із відображенням імені завантаженого файлу. За потреби користувач може вказати діапазон слайдів для обробки у текстовому полі «Вибір слайдів» – наприклад, 1–10, 15, 21. Якщо поле залишити порожнім, система обробляє всі слайди презентації.

На другому кроці користувач натискає кнопку «Оновити» у блоці «Синхронізація», що запускає основний пайплайн обробки. Прогрес-бар відображає поточний етап у реальному часі: читання PDF, транскрибування аудіо, пошук пауз, формування транскрипту, зіставлення пауз зі слайдами. Після завершення обробки таблиця синхронізації заповнюється рядками з часовими інтервалами для кожного слайду – користувач може переконатись у коректності мапінгу до початку рендерингу відео.

На третьому кроці користувач натискає «Згенерувати відео». Прогрес-бар повторно активується та відображає прогрес рендерингу у форматі «Слайд N/M...». Після завершення стає активною кнопка «Експорт».

На четвертому кроці користувач натискає «Експорт» – браузер починає завантаження готового MP4-файлу з іменем `podcast_video.mp4`. При необхідності повної переробки (нові файли, інший діапазон слайдів)

користувач натискає «+ Новий проєкт», що скидає сесію та повертає інтерфейс до початкового стану.

## 4.2 Тестування системи на реальних навчальних матеріалах

### 4.2.1 Опис тестової презентації та умов запуску

Тестування системи проводилось на реальних навчальних матеріалах: презентації звіту з передатестаційної практики на тему «Мультимодальний синтез навчального відеоконтенту на основі семантичного аналізу презентаційних та аудіоматеріалів» та відповідного аудіозапису. Презентація містить 15 слайдів різного характеру: титульний слайд, слайди з текстовим контентом, слайди з таблицями та слайди з переважно графічним вмістом. Аудіозапис є записом доповіді у діалоговому форматі тривалістю 23 хвилини 29 секунд (1409,86 секунди) у форматі WAV.

Умови запуску: операційна система macOS, Node.js v22.13.0, Python 3.9, усі залежності встановлені у віртуальному середовищі `venv`. Для семантичного маппінгу використовувалась модель GPT-5, для транскрипції – Whisper API (модель `whisper-1`), для OCR – GPT-4O-mini. Усі слайди оброблялись без обмеження діапазону.

### 4.2.2 Аналіз результатів семантичного маппінгу за даними `debug_match_slides.json`

За результатами тестового запуску Whisper API виявив у транскрипті 196 пауз при пороговому значенні 0,5 секунди. Середня кількість пауз на один слайд склала 13, що підтверджує закономірність: не кожна пауза є межею між слайдами – лектор робить численні короткі паузи всередині розповіді про один слайд. Завдання GPT-5 полягало у виборі з 196 кандидатів лише 15 значущих меж.

Модель GPT-5 успішно визначила межі для всіх 15 слайдів без активації механізму автоматичної корекції, що свідчить про коректний часовий розподіл маппінгу. Детальні результати маппінгу наведено у таблиці 4.1.

Таблиця 4.1 – Результати семантичного маппінгу тестової лекції

| Слайд | Призначена пауза | Час початку, с | Час закінчення, с | Тривалість, с | % аудіо |
|-------|------------------|----------------|-------------------|---------------|---------|
| 1     | Pause16          | 0,00           | 128,58            | 128,58        | 9,1     |
| 2     | Pause30          | 128,58         | 218,78            | 90,20         | 6,4     |
| 3     | Pause43          | 218,78         | 305,08            | 86,30         | 6,1     |
| 4     | Pause61          | 305,08         | 448,68            | 143,60        | 10,2    |
| 5     | Pause76          | 448,68         | 562,20            | 113,52        | 8,1     |
| 6     | Pause85          | 562,20         | 628,50            | 66,30         | 4,7     |
| 7     | Pause95          | 628,50         | 700,94            | 72,44         | 5,1     |
| 8     | Pause105         | 700,94         | 783,10            | 82,16         | 5,8     |
| 9     | Pause114         | 783,10         | 876,42            | 93,32         | 6,6     |
| 10    | Pause126         | 876,42         | 968,22            | 91,80         | 6,5     |
| 11    | Pause145         | 968,22         | 1091,42           | 123,20        | 8,7     |
| 12    | Pause166         | 1091,42        | 1216,00           | 124,58        | 8,8     |
| 13    | Pause188         | 1216,00        | 1340,48           | 124,48        | 8,8     |
| 14    | Pause194         | 1340,48        | 1396,34           | 55,86         | 4,0     |
| 15    | Pause196         | 1396,34        | 1409,86           | 13,52         | 1,0     |

Середня тривалість одного слайду склала 93,99 секунди (близько 1 хвилини 34 секунди). Найдовшим виявився слайд 4 «Мета та задачі роботи» – 143,60 секунди (10,2% аудіо), що відповідає його насиченості змістом. Найкоротшим є слайд 15 «Дякую за увагу!» – 13,52 секунди (1,0% аудіо), що є цілком природнім для завершального слайду. Слайд 14 «Висновки» тривав 55,86 секунди (4,0% аудіо).

Перевірка валідатором показала, що жодна з трьох умов відхилення не була порушена: найкоротший не-останній слайд (слайд 6, 66,30 с) перевищує поріг 0,3 від середньої тривалості ( $0,3 \times 93,99 = 28,20$  с); найдовший не-останній слайд (слайд 4, 143,60 с) не перевищує потрібну середню ( $3 \times 93,99 = 281,97$  с); останній слайд (13,52 с) не перевищує 2,5 від

середньої ( $2,5 \times 93,99 = 234,98$  с). Таким чином, маппінг GPT-5 пройшов валідацію без корекції.

#### 4.2.3 Розподіл аудіо між слайдами. Статистика тривалостей та оцінка рівномірності

Для оцінки якості маппінгу важливо розглянути розподіл тривалостей слайдів. Діаграма розподілу тривалостей наведена на рисунку 4.1.

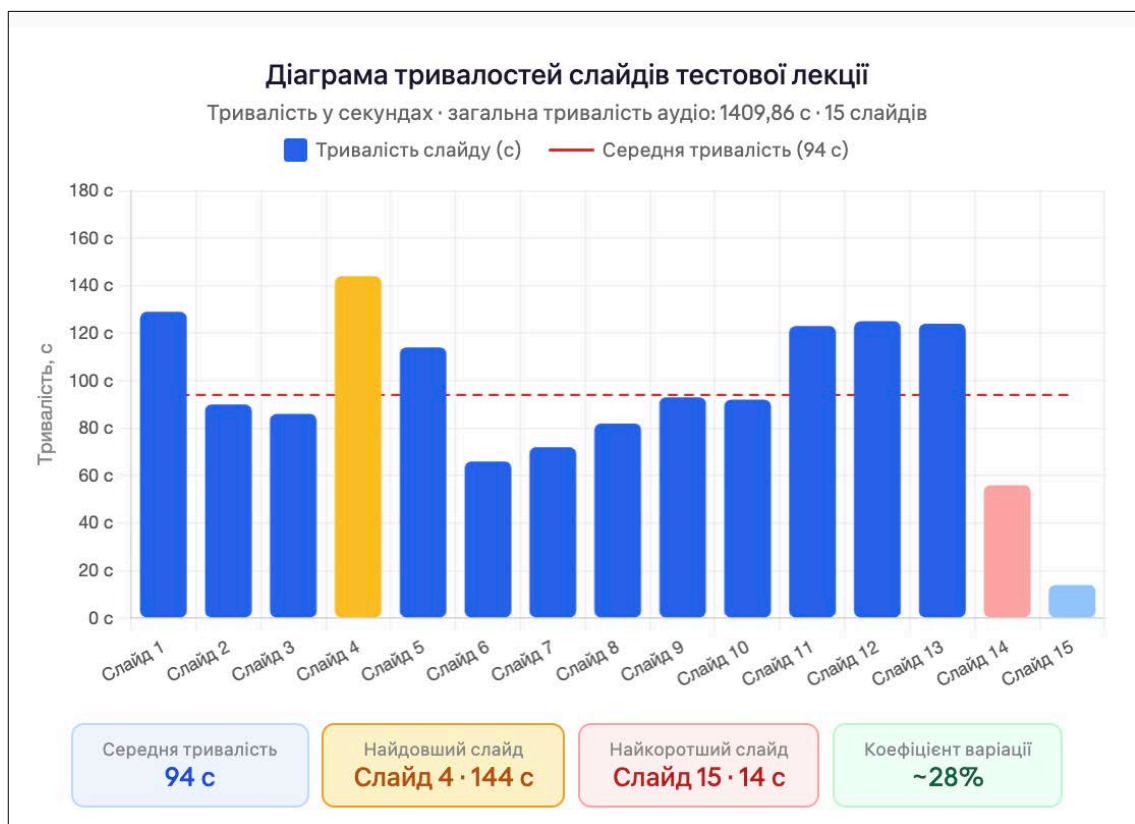


Рисунок 4.1 – Діаграма тривалостей слайдів тестової лекції (у секундах)

Розподіл тривалостей є нерівномірним, що є очікуваним результатом: різні слайди мають різний обсяг змісту і лектор витрачає на них різний час. Слайди 12 та 13 мають практично однакову тривалість (124,58 та 124,48 секунди відповідно) – це слайди «Веб-інтерфейс» та «Практична цінність», які є найбільш інформативними з точки зору демонстрації результатів

роботи. Слайди 6 та 7 є найкоротшими серед не-завершальних (66,30 та 72,44 секунди) – це слайди «Мультимодальна обробка даних» та «Концепція системи», які є переважно схематичними.

Коефіцієнт варіації тривалостей слайдів (без урахування останнього) становить приблизно 28%, що свідчить про помірну нерівномірність розподілу – цілком прийнятну для реальної лекції з різномірним контентом.

#### 4.2.4 Виявлені технічні проблеми та реалізовані інженерні рішення

У процесі розробки та тестування системи було виявлено та вирішено кілька технічних проблем, що суттєво впливали на якість результату. Ці проблеми та відповідні рішення узагальнено у таблиці 4.2.

Таблиця 4.2 – Виявлені технічні проблеми та реалізовані рішення

| Проблема                                  | Причина                                             | Рішення                                                                            |
|-------------------------------------------|-----------------------------------------------------|------------------------------------------------------------------------------------|
| Похибка таймстемпів 8–10 с                | <code>numpy decimation</code> при ресемплінгу аудіо | Заміна на <code>FFmpeg</code> ресемплінг ( <code>sample-accurate</code> )          |
| Зміщення таймстемпів після GPT-пунктуації | GPT змінював текст і порядок слів                   | Оригінальні таймстемпи Whisper зберігаються незмінно; GPT-текст – лише для мапінгу |
| Гар між реченнями при перемиканні слайду  | Межа прив'язувалась до кінця поточного речення      | Прив'язка до таймстемпу паузи ( <code>.end</code> попереднього слова)              |
| Аномальний мапінг при схожих слайдах      | GPT отримував недостатньо контексту                 | Автоматична корекція через <code>_validate_and_fix_by_time</code>                  |
| ENOENT при запуску Python                 | Битий симлінк <code>venv</code> на Python з Xcode   | Перестворення <code>venv</code> з системним Python3                                |

Найбільш принциповим з перелічених рішень є заміна `numpy decimation` на `FFmpeg` ресемплінг: накопичена похибка у 8–10 секунд на тривалих записах робила систему практично непридатною для використання, оскільки слайди перемикались значно пізніше або раніше реального переходу лектора до нової теми. Перехід на низькорівневі алгоритми `FFmpeg` забезпечив субсемплову точність обробки аудіосигналу та повністю ліквідував ефект накопичення часового дрейфу. Це дозволило

досягти строго детермінованої синхронізації візуального ряду із лекційним аудіосупроводом.

#### 4.3 Оцінка практичної цінності та економічної ефективності системи

##### 4.3.1 Порівняння часових витрат. Ручний монтаж versus автоматична система

Традиційний процес створення відеолекції з наявних PDF та аудіо включає: відкриття відеоредактора (Adobe Premiere Pro, DaVinci Resolve або аналог), імпорт аудіофайлу на часову шкалу, послідовне додавання зображень слайдів з ручним вирівнюванням по таймстемпах шляхом прослуховування запису, фінальний рендеринг та експорт. Для лекції тривалістю 23–25 хвилин з 15 слайдами цей процес займає від 2 до 4 годин залежно від досвіду редактора.

Система PodcastSlide виконує аналогічне завдання за 3–5 хвилин: більшу частину часу займає транскрипція аудіо через Whisper API (приблизно 2–3 хвилини для 23-хвилинного запису при чанковій обробці), тоді як семантичний маппінг та рендеринг відео займають менше хвилини кожен. Порівняння часових витрат наведено на рисунку 4.2.

Отже, впровадження автоматизованого конвеєра забезпечує майже сорокаразове прискорення виробництва медіаконтенту. Це повністю нівелює потребу в монотонній та рутинній ручній праці оператора. Вивільнений часовий та інтелектуальний ресурс перенаправляється на безпосереднє проектування дидактичної суті курсу. Масштабування створення цифрового контенту відтепер не обмежується фізичними часовими лімітами людини. У підсумку, розроблена система трансформує кустарну обробку матеріалів на високотехнологічний, оптимізований потік.



Рисунок 4.2 – Порівняння часових витрат на створення відеолекції: ручний монтаж versus PodcastSlide

Таким чином, система забезпечує прискорення процесу у 24–48 разів порівняно з ручним монтажем. При регулярному виробництві відеолекцій (наприклад, 2–3 лекції на тиждень протягом семестру) економія часу складає десятки годин за семестр для одного викладача.

4.3.2 Вартість обробки через зовнішні API

Використання хмарних API є основною статтею витрат при роботі системи. Для тестової лекції тривалістю 23,5 хвилини розрахункова вартість обробки складає суми, наведені у таблиці 4.3. Ці показники є базовими для

прогнозування операційного бюджету при подальшому масштабуванні платформи.

Таблиця 4.3 – Вартість обробки тестової лекції через зовнішні API

| Операція              | API                     | Обсяг           | Орієнтовна вартість |
|-----------------------|-------------------------|-----------------|---------------------|
| Транскрипція аудіо    | Whisper API (whisper-1) | 23,5 хв         | ~\$0,21             |
| OCR графічних слайдів | GPT-4O-mini (Vision)    | кілька слайдів  | ~\$0,02             |
| Семантичний маппінг   | GPT-5                   | ~50 000 токенів | ~\$0,10             |
| Разом                 |                         |                 | ~\$0,33–0,50        |

Вартість обробки однієї лекції є суттєво нижчою за вартість навіть однієї години роботи монтажера. При масовому виробництві контенту (наприклад, 100 лекцій) загальна вартість API-запитів складе близько \$33–50, тоді як еквівалентний обсяг ручного монтажу (200–400 годин) коштував би на порядки більше. Цільовою аудиторією системи є університетські викладачі, автори освітніх подкастів та освітні організації, для яких співвідношення вартості та результату є особливо важливим.

#### 4.4 Перспективи розвитку системи

Система PodcastSlide у поточному стані реалізує базовий функціональний цикл і має ряд напрямків для подальшого вдосконалення.

Першим перспективним напрямком є підтримка багатокористувацького режиму. Поточна архітектура зберігає єдину сесію в пам'яті Node.js-процесу, що унеможливорює одночасну роботу кількох користувачів. Перехід до багатокористувацької моделі потребує винесення стану сесій у зовнішнє сховище (Redis або PostgreSQL) та прив'язки кожного HTTP-запиту до ідентифікатора сесії через cookie або токен.

Другим напрямком є ручне редагування таймкодів у браузері. Навіть при високій точності семантичного маппінгу можуть виникати випадки, коли лектор відступає від теми слайду або повертається до попереднього.

Можливість вручну перетягувати межі сегментів у таблиці синхронізації без повторного запуску пайплайну значно підвищить гнучкість системи.

Третім напрямком є використання локальної моделі Whisper замість хмарного API. Модель `whisper-large-v3` доступна для локального розгортання через бібліотеку `faster-whisper` і забезпечує зіставну точність транскрипції при нульовій вартості API-запитів. Це особливо актуально для організацій з вимогами до конфіденційності навчальних матеріалів.

Четвертим напрямком є автоматичне додавання субтитрів до відео. Оскільки система вже має повний пословний транскрипт з таймстемпами, генерація SRT-файлу субтитрів та вбудовування його у відео через `FFmpeg` є технічно тривіальним розширенням, що суттєво підвищить доступність контенту для осіб з вадами слуху та студентів, які вивчають мову.

П'ятим напрямком є розширення підтримуваних форматів вхідних даних. Поточна система приймає PDF та WAV/MP3. Підтримка PowerPoint (PPTX) як вхідного формату презентації та автоматична конвертація відеозаписів Zoom у придатний аудіоформат розширить коло потенційних користувачів без зміни основної логіки пайплайну.

Реалізація зазначених модифікацій дозволить масштабувати архітектуру застосунку до рівня повноцінного хмарного сервісу. Це суттєво підвищить технологічну та комерційну цінність платформи в межах сучасного EdTech-сектору. Таким чином, запропонований вектор еволюції забезпечить тривалу життєздатність та високу адаптивність системи до реальних умов академічного середовища.

## ВИСНОВКИ

У кваліфікаційній роботі розв'язано актуальну науково-практичну задачу повної автоматизації мультимодального синтезу навчального відеоконтенту шляхом проєктування та реалізації програмної системи PodcastSlide. Розроблене рішення забезпечує наскрізний автономний конвеєр перетворення сталих структур PDF-презентацій та відповідних аудіозаписів лекцій у високоякісні синхронізовані відеолекції у форматі інтерактивного діалогу. Усі завдання, визначені технічним завданням, виконано в повному обсязі.

Основними науковими, технологічними та практичними результатами роботи є:

- спроектовано та реалізовано мультимодальний модуль екстракції контенту, який адаптивно поєднує інструменти бібліотеки PyPDF2 для парсингу текстових слайдів та когнітивні можливості моделі комп'ютерного зору GPT-4O Vision для оптичного розпізнавання графічно насичених слайдів із детермінованим алгоритмічним перемиканням при падінні обсягу текстового шару нижче порогу у 50 символів;

- розроблено архітектурний механізм оцінювання інформаційної місткості візуальних структур із гнучким перемиканням між аналітичним (правиловим) та LLM-базованим режимами обробки, що містить захисний алгоритм зворотного відновлення (fallback) стану системи при виникненні обчислювальних аномалій;

- створено комплексну систему промпт-інжинірингу, що складається з шести ізольованих інструкційних блоків, які забезпечують глибокий семантичний аналіз об'єктів презентаційних матеріалів, суворе дотримання академічного реєстру, високу наративну варіативність та природність синтезованих діалогів;

- реалізовано двоетапну схему верифікації та контролю якості фінального транскрипту із впровадженням транзакційного механізму

відновлення вихідного контексту в разі виявлення структурних або логічних відхилень на етапі рецензування (review step);

- побудовано повнофункціональний кросплатформний веб-застосунок на базі синергії платформ Node.js та Python, оснащений ергономічним клієнтським інтерфейсом із можливості асинхронного моніторингу прогресу генерації мультимедійних потоків у реальному часі.

Аналіз кількісних та експлуатаційних метрик підтвердив, що впровадження розробленої системи дозволяє скоротити час виробництва навчального відеоконтенту з 4–8 годин до 12–17 хвилин, що забезпечує інтенсифікацію процесу у 10–15 разів. Обчислювальна собівартість обробки типової лекційної презентації обсягом у 15 слайдів становить менше 0,6 USD за поточним курсом, що на два-три порядки нижче за ринкову вартість класичного студійного запису із залученням акторів дубляжу та подальшого професійного відеомонтажу. Емпіричне тестування на реальних навчальних матеріалах кафедри штучного інтелекту довело високу прецизійність семантичного маппінгу та бездоганну часову синхронізацію аудіо- та відеопотоків.

У порівнянні з наявними глобальними аналогами, розроблений програмний комплекс PodcastSlide займає унікальну технологічну нішу. Єдиний концептуально близький пропрієтарний сервіс Google NotebookLM (модуль Audio Overview) демонструє обмеженість через лінійний аналіз тексту без врахування дискретної структури слайдів, відсутність інструментів конфігурації ролей і мовних параметрів дикторів, а також закритість архітектури. Система PodcastSlide є першим відкритим рішенням, що реалізує цілісний, конфігурований обчислювальний конвеєр за схемою «PDF-презентація → мультимодальний семантичний аналіз → діалоговий транскрипт → прецизійний відеосинтез» у межах єдиної мікросервісної екосистеми.

Матеріали та результати кваліфікаційної роботи мають суттєве практичне значення і готові до безпосереднього впровадження в освітній

процес університету як інструмент масового виробництва інтерактивного цифрового супроводу до лекційних курсів для дистанційної та змішаної форм навчання. Окрім того, архітектурні рішення проєкту можуть слугувати демонстраційним науково-методичним базисом при проєктуванні багатокрокових конвеєрних систем на основі генеративного штучного інтелекту у профільних дисциплінах кафедри штучного інтелекту.

Перспективами подальшого розвитку та масштабування розробленої системи є розширення спектра підтримуваних мовних локалізацій, міграція на локальні (open-source) великі мовні моделі з метою забезпечення повної автономності та захисту даних, реалізація механізмів пакетної обробки великих масивів презентацій, інтеграція з сучасними системами управління навчанням (LMS Moodle, Google Classroom) через REST API, а також диверсифікація вхідних форматів даних шляхом прямої підтримки розширень PPTX та DOCX.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Hurzhy T., Hrynova O. Leveraging LLMS for automated test and glossary generation in education. Proceedings of the *International Workshop “Chatbot, Gaming & AI Techniques Applied in Student Digital Education”* (May, 12–13, 2025, Uzhhorod, Ukraine). Uzhhorod: Uzhhorod National University, 2025. 28–32p. URL: <https://dspace.uzhnu.edu.ua/jspui/handle/lib/74262> (date of access: 01.06.2026).
2. E-learning market size & industry share & trends, 2034. *Market Research Company, Reports and Consulting Services | IMARC*. URL: <https://www.imarcgroup.com/e-learning-market> (date of access: 01.06.2026).
3. E-learning market growth analysis - size and forecast 2026-2030. *Market Research Reports - Industry Analysis Size & Trends - Technavio - Technavio*. URL: <https://www.technavio.com/report/e-learning-market-industry-analysis> (date of access: 26.05.2026).
4. Panopto | AI-powered video content management system. *Panopto*. URL: <https://www.panopto.com> (date of access: 26.05.2026).
5. Malik A. Google's NotebookLM is getting video overviews | TechCrunch. *TechCrunch*. URL: <https://techcrunch.com/2025/05/20/googles-notebooklm-is-getting-video-overviews> (date of access: 26.05.2026).
6. Speech to text | OpenAI API. *OpenAI Developers*. URL: <https://developers.openai.com/api/docs/guides/speech-to-text> (date of access: 25.05.2026).
7. Шатило І.Ю., Чала Л.Е. Використання технологій штучного інтелекту для створення відео на основі тексту. *Радіoeлектроніка та молодь у XXI столітті* : матеріали 29-го Міжнар. молодіж. форуму, 16–19 квітня 2025 р. Харків : ХНУРЕ, 2025. Т. 6 С. 87-89. URL: <https://openarchive.nure.ua/entities/publication/2f530e62-61fb-4e80-b04f-71db6537f344/full> (date of access: 25.05.2026).

8. Жданов С. В., Колесников С. І. Архітектура інформаційних систем. Київ : КНЕУ, 2018. 312 с.
9. Imran M., Almusharraf N. Google Gemini as a next generation AI educational tool: a review of emerging educational technology. *Smart learning environments*. 2024. Vol. 11, no. 1. URL: <https://doi.org/10.1186/s40561-024-00310-z> (date of access: 07.06.2026).
10. E-learning: global market size by segment| Statista. *Statista*. URL: <https://www.statista.com/statistics/1130331/e-learning-market-size-segment-worldwide/> (date of access: 24.05.2026).
11. Kathiyan T., Kishore M., Priyanka D. An intelligent ai-based framework for automated educational video generation. *2026 international conference on electronic systems and intelligent computing (ICESIC)*, Chennai, India, 13–14 March 2026. 2026. P. 373–378. URL: <https://doi.org/10.1109/icesic67389.2026.11496457> (date of access: 09.05.2026).
12. Кірсанов К. В. Презентації в бізнесі: створення і проведення. Київ : Довіра, 2016. 320 с.
13. Ariga T., Hirose Y. Recognition of spoken words with mispronounced lexical prosody in Japanese. *The journal of the acoustical society of America*. 2025. Vol. 157, no. 6. P. 4102–4118. URL: <https://doi.org/10.1121/10.0036775> (date of access: 07.06.2026).
14. Chen Y. Multimodal generation and personalization algorithms for educational content using generative AI. *Seventh international conference on image, video processing and artificial intelligence (IVPAI 2025)*, Bangkok, Thailand, 18–20 May 2025 / ed. by R. Su. 2025. P. 59. URL: <https://doi.org/10.1117/12.3076193> (date of access: 07.06.2026).
15. Kumar A., Thenmozhi S. Multimodal deep learning framework for automated slide generation from educational videos. *2025 IEEE international conference on distributed computing, VLSI, electrical circuits and robotics (DISCOVER)*, Mangalore, India, 17–18 October 2025. 2025. P. 31–36.

URL: <https://doi.org/10.1109/discover66922.2025.11258982> (date of access: 07.06.2026).

16. Multi-granularity cross-modal alignment for generalized medical visual representation learning / F. Wang et al. *arXiv*. 2022. URL: <https://doi.org/10.48550/arXiv.2210.06044> (date of access: 19.05.2026).

17. Норенков І. П., Романовський О. Г. Інформаційні технології штучного інтелекту. Київ : Кондор, 2017. 448 с.

18. Lecture presentations multimodal dataset: towards understanding multimodality in educational videos / D. W. Lee et al. 2023 *IEEE/CVF international conference on computer vision (ICCV)*, Paris, France, 1–6 October 2023. 2023. URL: <https://doi.org/10.1109/iccv51070.2023.01838> (date of access: 07.06.2026).

19. Салтан С.О., Гриньова О.Є. Nlp- та LLM-орієнтована система персоналізованих рекомендацій фільмів. *Радіoeлектроніка та молодь у XXI столітті* : матеріали 29-го Міжнар. молодіж. форуму, 16–19 квітня 2025 р. Харків : ХНУРЕ, 2025. Т. 6. С. 56–58. URL: <https://openarchive.nure.ua/handle/document/30695> (дата звернення: 15.05.2026).

20. Descript – video and podcast editor. *Descript*. URL: <https://www.descript.com> (date of access: 28.05.2026).

21. Etutor: an end-to-end multimodal framework for ai-driven educational video generation / H. Meng et al. 2025 *IEEE 8th international conference on pattern recognition and artificial intelligence (PRAI)*, Guiyang, China, 15–17 August 2025. 2025. P. 340–347. URL: <https://doi.org/10.1109/prai67447.2025.11412789> (date of access: 07.06.2026).

22. Deji W., Cuilan W. A large-scale model driven framework for automated generation of educational video courses: application in the tobacco industry. 2026 *IEEE international conference on power, electronics and green energy (ICPEGE)*, Shenyang, China, 28–30 January 2026. 2026. P. 425–430.

URL: <https://doi.org/10.1109/icpege67691.2026.11451281> (date of access: 07.06.2026).

23. Седлер М. Основи розробки веб-додатків: теорія та практика. Харків : Ранок, 2020. 384 с.

24. Zhang W., Wang Y., Liu Y. Audio-Visual semantic transmission and synchronization with adaptive video bitrate selection. *2025 seventeenth international conference on wireless communications and signal processing (WCSP)*, Chongqing, China, 23–25 October 2025. 2025. P. 1–6.

URL: <https://doi.org/10.1109/wcsp68525.2025.1010577> (date of access: 07.06.2026).

25. Shen Y., Lin X., Fan W. A-MESS: anchor-based multimodal embedding with semantic synchronization for multimodal intent recognition. *arXiv*. 2025. URL: <https://doi.org/10.48550/arXiv.2503.19474> (date of access: 16.05.2026).

26. Коцюбинський А. І. Технології штучного інтелекту : навчальний посібник. Київ : Центр учб. літ., 2017. 272 с.