

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)

Мультимодальна генерація розмовного аудіоконтенту за даними семантичного
аналізу об'єктів презентаційних матеріалів
(тема)

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

_____ Руслан ГРІНЬОВ
(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 _____ Гріньову Руслану Олеговичу _____
(прізвище, ім'я, по батькові)

1. Тема роботи _____ Мультимодальна генерація розмовного аудіоконтенту за даними семантичного аналізу об'єктів презентаційних матеріалів _____

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, pydub), реальні PDF-презентації навчальних лекцій 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	02.05.2026	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: 100 с., 18 рис., 6 табл., 1 дод., 27 джерел.

АУДІОПОДКАСТ, ВЕЛИКІ МОВНІ МОДЕЛІ, ГЕНЕРАЦІЯ ДІАЛОГОВИХ ПАТЕРНІВ ПРОМПТ-ІНЖИНІРИНГ, ІНТЕЛЕКТУАЛЬНА СИСТЕМА, СЕМАНТИЧНА ТРАНСФОРМАЦІЯ КОНТЕНТУ, СИНТЕЗ ПРИРОДНОГО МОВЛЕННЯ.

Об'єкт дослідження – процес автоматизованої семантичної трансформації статичного навчального контенту в динамічний розмовний аудіоформат.

Предмет дослідження – інтелектуальні методи промпт-інжинірингу та архітектурної оркестрації великих мовних моделей для генерації імерсивних діалогових транскриптів.

Мета роботи – розробка та практична реалізація інтелектуальної програмної системи для автономного перетворення візуально-текстових матеріалів (PDF-презентацій) на високоякісні аудіоподкасти із застосуванням генеративних ШІ-технологій.

Методи дослідження – системний аналіз (дослідження архітектур LLM та систем TTS), інженерний (проектування конвеєра обробки даних та багаторівневої системи промпт-інжинірингу), експериментальний (валідація згенерованих аудіоматеріалів на предмет когнітивного навантаження та природності звучання).

Розроблено систему PodcastGen. Впровадження 6-блокової конвеєрної архітектури промптів та двоетапного контролю якості дозволило нівелювати галюцинації і «механічність» ШІ. Система радикально пришвидшує створення імерсивного освітнього контенту, генеруючи природні комунікативні патерни.

ABSTRACT

Bachelor's thesis contains: 100 pp., 18 fig., 6 tabl., 1 ann., 27 references.

AUDIO PODCAST, DIALOGUE PATTERN GENERATION, INTELLIGENT SYSTEM, LARGE LANGUAGE MODELS, NATURAL SPEECH SYNTHESIS, PROMPT ENGINEERING, SEMANTIC CONTENT TRANSFORMATION.

Object of research – the process of automated semantic transformation of static educational content into a dynamic conversational audio format.

Subject of research – intelligent methods of prompt engineering and architectural orchestration of large language models for generating immersive dialogue transcripts.

Purpose of the study – the development and practical implementation of an intelligent software system for the autonomous conversion of visual-textual materials (PDF presentations) into high-quality audio podcasts utilizing generative AI technologies.

Research methods – systems analysis (investigation of LLM architectures and TTS systems), engineering methods (design of a data processing pipeline and a multi-level prompt engineering system), and experimental evaluation (validation of generated audio materials regarding cognitive load and naturalness of sound).

Results – the PodcastGen system has been developed. The implementation of a 6-block prompt pipeline architecture and two-stage quality control successfully neutralized AI hallucinations and «mechanical» sound. The system radically accelerates the creation of immersive educational content by generating natural communicative patterns.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	9
Вступ.....	10
1 Аналіз предметної галузі та постановка задачі.....	13
1.1 Розвиток систем автоматичної генерації аудіоконтенту	13
1.1.1 Еволюція аудіальної дидактики. Від лінійного мовлення до інтелектуальних систем генерації	13
1.1.2 Еволюція систем синтезу мовлення. Від акустичної симуляції до когнітивної емпатії.....	15
1.1.3 Мультимодальні підходи до обробки тексту та зображень	18
1.1.4 Великі мовні моделі у задачах генерації діалогів.....	19
1.2 Аналіз існуючих підходів та інструментів	21
1.2.1 Архітектурний аналіз систем синтезу мовлення: від SSML-детермінізму до семантичної просодики.....	21
1.2.2 Архітектурний аналіз платформ генерації контенту. Від лінійного монтажу до закритих екосистем	23
1.2.3 Порівняльний аналіз систем генерації навчального аудіоконтенту	24
1.2.4 Архітектурні стратегії генерації діалогових сценаріїв. Від примітивних викликів до Stateful-оркестрації	26
1.3 Актуальність задачі та архітектурні бар'єри існуючих рішень	28
1.3.1 Криза масштабованості освітнього контенту. Проблема семантичного розриву	28
1.3.2 Відсутність повністю автоматизованого ланцюжка PDF → аудіодіалог	29
1.3.3 Потреба у семантичному аналізі презентаційних матеріалів....	30
1.4 Постановка задачі та мета роботи	31
2 Методи та архітектура системи	33
2.1 Загальна архітектура мультимодальної системи	33

2.1.1 Pipeline обробки: PDF – текст – оцінка – діалог – аудіо	33
2.1.2 Рівні обробки даних та взаємодія модулів	34
2.1.3 Інтеграція LLM та TTS у єдиний конвеєр	36
2.2 Методи обробки PDF-презентацій	36
2.2.1 Витяг тексту з PDF-документів (PyPDF2).....	36
2.2.2 OCR-обробка графічних слайдів через GPT-4o Vision	38
2.2.3 Автоматична оцінка інформаційної насиченості слайду.....	39
2.3 Методи генерації діалогового контенту	41
2.3.1 Семантичний аналіз тексту слайдів як основа діалогу	41
2.3.2 Prompt engineering для генерації академічного діалогу	42
2.3.3 Контекстне моделювання між слайдами (history_of_conversation)	45
2.3.4 Форматні пресети діалогу (formal / dialog / simplified)	46
2.3.5 Постобробка та багаторівневий контроль якості транскрипту .	46
2.3.6 Обробка спеціальних слайдів (титульний та завершальний)	48
2.4 Методи синтезу мовлення та формування аудіо	49
2.4.1 Text-to-Speech через OpenAI TTS-1-HD	49
2.4.2 Вибір голосів залежно від статі та ролі співрозмовника	50
2.4.3 Обробка та злиття PCM-сегментів через pydub.....	50
2.4.4 Управління паузами між репліками та ресемплінг	52
2.5 Обґрунтування вибору технологій	52
3 Програмна реалізація системи	55
3.1 Архітектура програмного забезпечення	55
3.1.1 Структура модулів та файлова організація проєкту.....	55
3.1.2 Організація pipeline та передача даних між кроками.....	56
3.1.3 Конфігурація форматних пресетів та параметрів запуску.....	57
3.2 Реалізація модуля обробки PDF	59
3.2.1 Парсинг тексту: PyPDF2 та порогова логіка	59
3.2.2 Інтеграція OCR через GPT-4o Vision	60
3.2.3 Реалізація оцінки складності слайду.....	61

3.3 Реалізація генерації діалогів	63
3.3.1 Формування системного та користувацького пром프트у	63
3.3.2 Обробка імен та ролей співрозмовників.....	64
3.3.3 Генерація та вставка вступної фрази	65
3.3.4 Реалізація постобробки (_review_transcript).....	67
3.3.5 Система JSON-логування для налагодження.....	68
3.4 Реалізація синтезу мовлення.....	70
3.4.1 Інтеграція OpenAI TTS API (PCM формат).....	70
3.4.2 Парсинг XML-тегів та фільтрація сегментів.....	71
3.4.3 Формування фінального WAV через pydub	71
3.5 Реалізація веб-застосунку.....	73
3.5.1 Серверна частина: Node.js + Express, job queue	73
3.5.2 API endpoints та обробка файлів (multer)	74
3.5.3 Відстеження прогресу генерації в реальному часі	75
3.5.4 Клієнтська частина: інтерфейс та вбудований плеєр	76
4 Тестування та аналіз результатів.....	84
4.1 Опис користувацького інтерфейсу та сценарії використання.....	84
4.2 Тестування системи на реальних навчальних матеріалах	85
4.2.1 Опис тестових презентацій та умов запуску	85
4.2.2 Аналіз якості транскриптів	86
4.2.3 Виявлені проблеми та інженерні рішення.....	88
4.3 Оцінка практичної цінності та економічної ефективності	90
4.3.1 Порівняння часових витрат: ручний монтаж versus система	90
4.3.2 Вартість обробки через API	91
4.4 Перспективи розвитку системи	92
Висновки	94
Перелік джерел посилання	96
Додаток А Відомість кваліфікаційної роботи	100

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

III – Штучний інтелект;

API – Application Programming Interface – прикладний програмний інтерфейс;

GPT – Generative Pre-trained Transformer – генеративний попередньо натренований трансформатор;

JSON – JavaScript Object Notation – текстовий формат обміну даними на основі синтаксису JavaScript;

LLM – Large Language Model – велика мовна модель;

LMS – Learning Management System – система управління навчанням;

NLP – Natural Language Processing – обробка природної мови;

OCR – Optical Character Recognition – технологія оптичного розпізнавання символів;

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

PDF – Portable Document Format – портативний формат документів;

SSML – Speech Synthesis Markup Language – мова розмітки для синтезу мовлення;

TTS – Text-to-Speech – технологія синтезу мовлення з тексту;

WAV – Waveform Audio File Format – формат нестиснених аудіофайлів;

XML – Extensible Markup Language – розширювана мова розмітки.

ВСТУП

Епоха глобальної цифровізації освіти, яка отримала стрімке прискорення після 2020 року, остаточно деконструювала класичні формати передачі знань. Сучасний здобувач освіти існує в режимі безперервного, мобільного та асинхронного споживання інформації. У цій парадигмі аудіоподкаст еволюціонував із нішевого медіа у потужний дидактичний інструмент. Макроекономічні індикатори глобального сектора EdTech демонструють безпрецедентну динаміку: згідно з аналітичними зведеннями IMARC Group, станом на 2025 рік капіталізація світового ринку електронного навчання сягнула позначки у 369,7 мільярда доларів США. Структурний аналіз цього зростання вказує на те, що драйвером новітньої освітньої парадигми стають генеративні аудіотехнології, які трансформують статичну інформацію у глибоко імерсивний акустичний досвід [1], [2]. Однак на шляху до масового масштабування цього формату стоїть фундаментальний бар'єр технологічної та часової складності. Традиційний виробничий цикл (написання сценарію, студійний запис, монтаж і мастеринг) вимагає від автора курсу від чотирьох до восьми годин роботи на один епізод тривалістю двадцять-тридцять хвилин [3]. Існуючі ж базові системи синтезу мовлення (TTS) здатні лише механічно зачитувати текст, продукуючи «сухий», позбавлений емпатії звук, що критично знижує когнітивне залучення слухача.

Критичний аналіз існуючого програмного ландшафту засвідчує глибокий дефіцит архітектурно завершених рішень, здатних здійснювати наскрізну когнітивно-семантичну трансформацію первинних навчальних матеріалів. Відомі інструменти, такі як OBS Studio, Camtasia або Descript, функціонують у застарілій парадигмі лінійного або напівавтоматичного медіаредагування; вони вирішують виключно локальні, утилітарні завдання вокалізації чи монтажу, будучи повністю відірваними від процесів інтелектуального аналізу контексту [4,], [5].

Єдиним релевантним контурним аналогом на ринку є функціонал Audio Overview платформи Google NotebookLM, який демонструє здатність до генерації двоголосої дискусії. Однак спроба реалізувати автономний синтез подкастів у межах цього сервісу виявила системні недоліки базових монолітних підходів до LLM-генерації. Сервіс здійснює грубу гомогенізацію вхідних даних: алгоритм нівелює логіко-композиційну дискретність презентаційних слайдів, зводячи їх, разом із графічними інваріантами, до недиференційованого текстового масиву.

Для розробника інтелектуальних систем критичним недоліком є повна закритість архітектури промпт-оркестрації. Відсутність доступу до системних інструкцій (system prompts) виключає можливість гнучкого моделювання комунікативних стратегій, налаштування тональності, емпатії чи темпоритму віртуальних мовців. Крім того, пропрієтарна природа сервісу та відсутність відкритого прикладного програмного інтерфейсу (API) роблять неможливим його розгортання в незалежних університетських LMS-екосистемах.

Актуальність даної роботи визначається необхідністю розв'язання задачі вищого порядку у сфері генеративного штучного інтелекту: семантичної трансформації статичного контенту у живий діалоговий патерн. На відміну від систем аудіовізуальної синхронізації та механічного озвучення, тут головним інженерним викликом є надання синтезованому мовленню людяності – здатності ШІ розуміти дидактичний контекст, дискутувати та акцентувати увагу на складних концептах. Підставою для виконання роботи є, з одного боку, доступність сучасних хмарних API великих мовних моделей і систем синтезу мовлення [6], [7], а з іншого – відсутність готових рішень, що поєднують ці технології у єдиний конфігурований конвеєр для освітніх цілей. Це вимагає розробки складної оркестрації промптів, здатної перетворити сухий академічний PDF-файл на інтелектуальний аудіоспектакль.

Метою роботи є розробка та практична реалізація програмної системи PodcastGen – інтелектуального конвеєра, що автоматично перетворює навчальні PDF-презентації на високоякісні аудіоподкасти у форматі емпатичного діалогу. Система реалізує глибокий мультимодальний підхід – текстові масиви обробляються інструментами PyPDF2, тоді як графічні примітиви (діаграми, формули, таблиці) семантично декодуються через GPT-4o Vision [8], що гарантує абсолютну повноту екстракції знань.

Запропоновані у роботі підходи до мультимодальної обробки контенту розвивають ідеї, викладені у працях співробітників кафедри ШІ, ХНУРЕ [9]. Практичне значення роботи полягає у радикальній демократизації процесу створення освітнього аудіоконтенту. Розроблена архітектура скорочує час виробництва повноцінного подкасту з кількох годин до дванадцяти-сімнадцяти хвилин. При цьому собівартість генерації одного епізоду становить близько п'ятдесяти семи центів, що на кілька порядків дешевше за класичний студійний запис, повністю зберігаючи при цьому ілюзію живого людського спілкування.

Структура роботи. Пояснювальна записка складається з чотирьох розділів. У першому розділі досліджено еволюцію генеративних систем та обґрунтовано проблематику машинного синтезу діалогів. Другий розділ розкриває архітектурне ядро – спроектований конвеєр обробки даних, унікальну 6-блокову систему промпт-інжинірингу та алгоритми боротьби з галюцинаціями LLM. У третьому розділі наведено програмну реалізацію бекенд-модулів та веб-інтерфейсу. Четвертий розділ містить результати експериментальної валідації системи, оцінку акустичної природності згенерованих патернів та аналіз економічної ефективності розробки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Розвиток систем автоматичної генерації аудіоконтенту

1.1.1 Еволюція аудіальної дидактики. Від лінійного мовлення до інтелектуальних систем генерації

Протягом останніх двох десятиліть парадигма споживання освітньої інформації зазнала фундаментальної трансформації. Цифровий аудіоконтент подолав шлях від нішевого формату передачі даних до домінуючого інструменту асинхронного навчання. Якщо на етапі свого зародження освітній подкастинг сприймався виключно як засіб одностороннього, механічного мовлення, то сьогодні він набув ознак складної дидактичної екосистеми. У сучасних реаліях здобувач освіти потребує не просто озвученого тексту, а імерсивного інтелектуального середовища, здатного забезпечити ефект живого діалогу (формат «listen- while-commuting»), де технологія слугує своєрідним інструментом гуманізації освітнього процесу, скорочуючи когнітивну дистанцію між автором та слухачем.

Масштаб цієї трансформації беззаперечно підтверджується макроекономічними індикаторами. За аналітичними зведеннями IMARC Group, у 2025 році обсяг глобального ринку e-learning досяг позначки 369,7 мільярда доларів США. Прогнозується, що до 2034 року цей показник майже подвоїться, сягнувши 731,8 мільярда доларів із середньорічним темпом зростання (CAGR) на рівні 7,65% [1]. За оцінками експертів Statista, лише в період 2025–2030 років індустрія згенерує додаткові сотні мільярдів доларів капіталізації [2] (рисунок 1.1). Станом на 2026 рік ринок впевнено наближається до 400 мільярдів доларів, що концептуально змінює вимоги до швидкості та обсягів масштабування навчальних матеріалів.

Динаміка зростання глобального ринку e-learning у 2019–2026 рр. (млрд дол. США)



Рисунок 1.1 – Динаміка зростання глобального ринку e-learning у
2019– 2026 рр. (млрд дол. США)

Проте на шляху до масового впровадження аудіальних форматів в освітній процес постає фундаментальний технологічний бар'єр – критична диспропорція між експоненційним попитом на мобільний контент та архаїчною, лінійною парадигмою його виробництва. У сучасних умовах, коли навчальні програми потребують безперервного оновлення, швидкість доставки знань стає визначальним фактором, який класичні медіатехнології забезпечити не здатні.

Традиційний виробничий цикл створення єдиного тридцятихвилинного освітнього епізоду вимагає від чотирьох до восьми годин висококваліфікованої праці [3]. Цей процес є глибоко фрагментованим і обтяженим когнітивними та технічними втратами на кожному етапі. Спочатку автор курсу змушений здійснювати ручну семантичну редукцію – трансформувати щільний академічний матеріал у розмовний сценарій, що вимагає специфічних навичок сторітелінгу та адаптації тексту для сприйняття на слух. Наступний етап – студійний

запис – жорстко прив'язаний до акустичних умов, наявності апаратної бази та фізичного стану диктора. Фінальна стадія, яка охоплює багаторівневий нелінійний монтаж, очищення від звукових артефактів, нормалізацію гучності та мастеринг, остаточно перетворює викладача на технічного оператора, відриваючи його від безпосередньо науково-методичної діяльності.

Така архітектура виробництва породжує тотальну операційну неефективність. Вона фактично унеможлиблює систематичну адаптацію динамічних університетських курсів до аудіоформату, адже будь-яка зміна в навчальній програмі вимагає повного перезапису відповідних сегментів. Це робить підтримку актуальності контенту економічно та ресурсно нерентабельною.

Саме ця криза масштабування стала головним каталізатором для парадигмального зсуву: переходу від ручного медіавиробництва до проєктування інтелектуальних систем автономної генерації подкастів. Вирішення цієї проблеми лежить у площині застосування великих мовних моделей (LLM), які здатні взяти на себе не лише етап вокалізації, але й найскладніший когнітивний процес – автономну семантичну трансформацію статичного матеріалу в природний, емпатичний діалоговий патерн. Це дозволяє перетворити створення подкасту з багатоденної технічної рутини на алгоритмічну оркестрацію, де штучний інтелект виступає не механічним ретранслятором, а повноцінним комунікативним партнером.

1.1.2 Еволюція систем синтезу мовлення. Від акустичної симуляції до когнітивної емпатії

Синтез мовлення (Text-to-Speech, TTS) є однією з найраніших галузей застосування штучного інтелекту і пройшов кілька принципово різних технологічних етапів. Перші системи TTS, що з'явилися у 1970–80-х роках,

базувались на конкатенативному синтезі – методі зчеплення попередньо записаних одиниць мовлення (фонем, диграфів, складів). Незважаючи на відносну зрозумілість отриманого мовлення, конкатенативний синтез страждав від неприродних переходів між елементами та різко обмеженої просодичної варіативності.

Наступним етапом стали параметричні методи на основі прихованих марківських моделей (Hidden Markov Model, HMM), що дозволяли статистично описувати акустичні характеристики мовлення й генерувати більш плавну мелодику. Однак і цей підхід виявляв характерну «роботизованість», через яку синтетичне мовлення легко відрізнялось від природного.

Кардинальний прорив стався у 2016–2018 роках із впровадженням неймережових архітектур – зокрема моделей (DeepMind, 2016) [10], Tacotron (Google, 2017) [11], а також FastSpeech [12]. Ці системи вперше продемонстрували якість синтезу, наближену до природного мовлення людини, завдяки кінцево-до-кінцевому (end-to-end) навчанню на великих корпусах аудіозаписів (рисунок 1.2).



Рисунок 1.2 – Еволюція архітектур систем синтезу мовлення

Компанія OpenAI запропонувала власний TTS-стек: початково сервіс підтримував дві моделі – `tts-1` та `tts-1-hd`, що забезпечують дев'ять унікальних голосів та контроль над швидкістю відтворення [6]. У березні 2025 року OpenAI представила нову модель `gpt-4o-mini-tts` із тринадцятьма голосами та принципово новою функцією керованої просодики (`steerable prosody`) [7]. Саме модель `tts-1-hd` задіяна у системі PodcastGen, що розглядається в даній роботі, оскільки повертає аудіо у форматі PCM – «сирих» аудіоданих без заголовків контейнера, що суттєво спрощує пряме зчеплення сегментів засобами бібліотеки `pydub`.

Окремо слід відзначити проблему читання числових значень у системах TTS. Більшість нейромережевих синтезаторів мовлення, отримуючи на вхід цифровий запис числа, відтворюють його неприродно або некоректно у контексті мовлення. Ця проблема особливо актуальна для освітнього контенту з великою кількістю статистичних даних, формул та числових показників. Одним із архітектурних рішень, реалізованих у розробленій системі, є обов'язкова передача всіх чисел у текстовому вигляді ще на етапі генерації транскрипту, що повністю усуває зазначений артефакт.

У контексті розробки системи PodcastGen ключовим інструментарієм є генеративний стек компанії OpenAI. Використана в архітектурі модель `tts-1-hd` забезпечує не просто високоякісну вокалізацію, а імітацію живого комунікативного акту. Вибір саме цієї моделі зумовлений глибокими інженерними вимогами: вона здатна повертати аудіо у форматі PCM (Pulse-Code Modulation) – у вигляді масиву сирих акустичних даних без контейнерних заголовків. Це дозволяє системі PodcastGen програмно обходити обмеження стандартних аудіоформатів та забезпечувати безшовне, мілісекундне зчеплення діалогових реплік на рівні бекенду за допомогою алгоритмів бібліотеки `pydub`.

Окремим архітектурним викликом у мультимодальних системах є проблема контекстної текстової нормалізації (Contextual Text Normalization). Більшість сучасних TTS-моделей, отримуючи на вхід

цифровий запис (характерний для презентацій зі статистикою, датами та формулами), продукують акустичні артефакти або руйнують природний темпоритм диктора. У системі PodcastGen цю проблему вирішено концептуально інакше: не на рівні постобробки аудіо, а безпосередньо у ядрі семантичного промпт-інжинірингу. Розроблений конвеєр інструкцій примусово переводить усі числові та символічні матриці у фонетично-текстовий формат (вербалізацію) ще на етапі генерації транскрипту LLM. Це гарантує абсолютну лексичну чистоту вхідних даних для моделі TTS та повністю нівелює ризик руйнування ілюзії людського спілкування.

1.1.3 Мультимодальні підходи до обробки тексту та зображень

Мультимодальність у сучасних системах штучного інтелекту означає не просто одночасну обробку даних різних типів (текст, зображення, аудіо), а здатність моделі встановлювати глибокі семантичні інваріанти між ними. Для задачі автоматичної трансформації презентаційних матеріалів цей підхід є критичним, оскільки навчальні слайди за своєю природою є гетерогенними середовищами, де графічні примітиви (діаграми, схеми, таблиці) несуть самостійне, часто першочергове когнітивне навантаження [13], [14].

Застосування традиційних методів оптичного розпізнавання символів (OCR) – зокрема Tesseract або ABBYY – у цьому контексті демонструє тотальну операційну неспроможність. Архітектури OCR розроблялися для суцільних, лінійно-структурованих документів; при спробі обробити слайд із нелінійним розташуванням тексту, нашаруванням поверх графіки або векторними елементами, OCR-система руйнує просторову та логічну ієрархію контенту. Отриманий на виході незв'язний масив символів є непридатним для подальшого когнітивного аналізу великими мовними моделями.

Злам парадигми стався із появою Transformer-архітектур, інтегрованих з візуальними кодувальниками (Vision Encoders), що дозволило перейти до парадигми End-to-End Multimodal Understanding [15]. Використана у системі PodcastGen модель GPT-4o з підтримкою Vision-режиму не просто «розпізнає текст» на зображеннях. Вона здійснює цілісне семантичне декодування: модель одночасно аналізує візуальні та текстові характеристики, встановлюючи зв'язки між, наприклад, елементами схеми та підписами до них.

Вибір саме цієї архітектури для обробки графічних слайдів є обґрунтованим інженерним компромісом. Незважаючи на вищу обчислювальну складність та собівартість генерації, мультимодальний аналіз є єдиним способом гарантувати повноту екстракції дидактичних знань. Там, де традиційний парсинг засобами PyPDF2 повертає менше п'ятдесяти символів (ігноруючи семантику графіків та формул), GPT-4o Vision забезпечує безшовне відновлення когнітивної структури матеріалу, що є залізобетонним фундаментом для подальшої LLM-генерації діалогу.

1.1.4 Великі мовні моделі у задачах генерації діалогів

Великі мовні моделі (Large Language Models, LLM) еволюціонували від простих систем автодоповнення до потужних когнітивних механізмів, здатних вирішувати задачі складної семантичної трансформації. Провідні архітектури класу Gemini 1.5 (Google DeepMind) [16], Claude 3 (Anthropic), GPT-4o (OpenAI) та LLaMA 3 (Meta) об'єднані здатністю до глибокого контекстуального розуміння та генерації зв'язного тексту за складними багатовекторними інструкціями (промптами). Саме ця властивість робить їх фундаментальним інструментарієм для проєктування систем автоматизованого створення освітнього контенту [17], [18].

Проте генерація полілогічного (зокрема діалогового) дискурсу висуває до LLM екстремальні вимоги щодо когерентності, утримання системного контексту (context window constraint) та збереження консистентності персонажів (Role-Playing consistency) впродовж тривалих сесій. Дослідження у сфері конwersаційних агентів доводять, що автономна генерація якісного діалогу неможлива без складної інженерної оркестрації: якість вихідного патерну прямо пропорційна архітектурі системного промпту (System Prompt), який детермінує ролі, когнітивний реєстр, заборонені теми (guardrails) та метрики емпатії [19], [20]. В освітньому контексті критичним стає інженерне керування дидактичним розподілом: ШІ має не просто переказувати факти, а моделювати сократівський діалог, де ролі (наприклад, лектора-експерта та зацікавленого студента) слугують інструментами підтримання природної динаміки розмови [21].

Для вирішення цих задач в ядрі системи PodcastGen розгорнуто модель gpt-4o-mini. Вибір цієї моделі обґрунтовано результатами архітектурного трейд-оффу, вона забезпечує ідеальну здатність до строгого дотримання жорстких форматів інструкцій (зокрема генерації JSON-структур), необхідних для академічного контенту, демонструючи при цьому експоненційно нижчу вартість інференсу за токен порівняно з флагманською моделлю GPT-4o [6], [7].

Важливо акцентувати, що в парадигмі розробленої системи PodcastGen сама мовна модель виступає лише базовим процесором. Визначальним фактором когнітивної якості згенерованого подкасту (відсутності галюцинацій та механічності) є спроектована система багаторівневого промпт-інжинірингу – жорсткий конвеєр інструкцій, обмежень і форматних вимог, що скеровують генерацію на кожному етапі трансформації вхідного PDF-матеріалу [22], [23]. Такий підхід перетворює розрізнені API-запити на алгоритмічний моноліт, який функціонує як семантичний запобіжник проти деградації діалогу до рівня машинного шаблону. Завдяки цій архітектурній жорсткості технологія долає бар'єр

штучності, генеруючи не просто озвучений текст, а стан когнітивної емпатії та глибокої дидактичної взаємодії. Відтак, перехід від ситуативного керування моделлю до проєктування детермінованого конвеєра, здатного автономно утримувати контекстну цілісність, становить фундаментальний науково-інженерний здобуток цього дослідження.

1.2 Аналіз існуючих підходів та інструментів

1.2.1 Архітектурний аналіз систем синтезу мовлення: від SSML-детермінізму до семантичної просодики

Сучасний ландшафт систем машинного синтезу мовлення (TTS) концептуально розділений на хмарні неймережеві екосистеми та локальні open-source рішення. Кожен із цих класів пропонує власний архітектурний компроміс між обчислювальною автономністю, якістю когнітивного залучення та методами керування генерацією.

В екосистемі корпоративних хмарних рішень, наприклад таких як Google Cloud TTS, Microsoft Azure Cognitive Services, Amazon Polly історично домінує парадигма детермінованого керування просодикою за допомогою мови розмітки SSML (Speech Synthesis Markup Language). Ці платформи (зокрема моделі WaveNet та Neural2) забезпечують високу акустичну якість, проте їхня архітектура вимагає жорсткого ручного програмування пауз, інтонацій та емоційних акцентів через XML-подібні теги [10], [24]. У контексті розробки повністю автономних систем, де сценарій генерується великою мовною моделлю у реальному часі, залежність від SSML стає критичним алгоритмічним вузьким місцем, оскільки змушує будувати складні додаткові парсери для перекладу семантики емоцій у технічні теги.

Парадигмальний зсув у цьому напрямку відбувся з релізом генеративного стеку OpenAI, який здійснив перехід від SSML-детермінізму

до семантично керованої просодики (steerable prosody). Станом на травень 2026 року екосистема пропонує моделі класу tts-1, tts-1-hd та найновішу gpt-4o-mini-tts (реліз березня 2025 року). Інноваційність останньої полягає у здатності моделі налаштовувати тон, емпатію та емоційне забарвлення виключно через контекстні текстові інструкції, що ідеально інтегрується в архітектуру промпт-інжинірингу без використання жодних мов розмітки.

Втім, для безпосереднього аудіо-ядра системи PodcastGen залучено модель tts-1-hd. Цей вибір продиктовано не лише високою акустичною деталізацією, а фундаментальною інженерною необхідністю: модель здатна повертати результати інференсу у форматі PCM (Pulse-Code Modulation). На відміну від стандартних MP3-контейнерів, PCM є масивом «сирих» аудіоданих без метазаголовків. Це дозволяє архітектурі PodcastGen здійснювати безшовне, потокове зчеплення діалогових реплік (на рівні байтів засобами бібліотеки pydub), повністю уникаючи ресурсомістких циклів транскодування та декодування на сервері.

Аналізуючи альтернативний вектор розвитку – відкриті локальні системи (на кшталт Coqui TTS або Kokoro) – слід визнати їхню незалежність від зовнішніх API. Зокрема, компактна архітектура Kokoro (з об'ємом у вісімдесят два мільйони параметрів) демонструє конкурентну якість для англomовного середовища. Однак для розв'язання задачі мультимовної освітньої генерації (зокрема підтримки розмовної української мови з урахуванням академічної термінології) локальні моделі виявляють критичний дефіцит навчальних корпусів. Вони не здатні забезпечити той рівень zero-shot емоційної виразності та лексичної чистоти, який необхідний для утримання уваги слухача навчального подкасту, що зумовлює потребу в розробці нових підходів до генерації мовлення, здатних адаптуватися до контексту, стилю та комунікативних особливостей освітнього контенту.

1.2.2 Архітектурний аналіз платформ генерації контенту. Від лінійного монтажу до закритих екосистем

Сучасний ринок програмного забезпечення для створення аудіальної дидактики можна стратифікувати на три архітектурні класи, кожен з яких розв'язує вузький спектр завдань і володіє критичними обмеженнями щодо наскрізної автоматизації навчального процесу.

До першого класу належать системи лінійного та напівавтоматичного аудіомонтажу (Descript, Adobe Podcast, Audacity). Попри інтеграцію локальних ШІ-модулів (наприклад, алгоритмів шумозаглушення чи клонування голосу Overdub), їхня базова архітектура залишається жорстко прив'язаною до ручного виробничого циклу. Вони функціонують виключно як інструменти постобробки, вимагаючи від лектора попередньо згенерованого аудіозапису або повністю готового текстового сценарію. Ці платформи не містять вбудованих механізмів семантичного аналізу первинних документів, що залишає їх поза межами генеративного конвеєра.

Другий клас формують спеціалізовані нейромережеві TTS-платформи (ElevenLabs, Murf AI, Lovo). З інженерної точки зору, ці рішення є високоефективними акустичними вокодерами, що забезпечують якісну просодіку та зручний інтерфейс синтезу. Проте вони повністю позбавлені когнітивного рушія: такі платформи не здатні самостійно здійснювати екстракцію знань із презентаційних матеріалів чи автономно моделювати діалоговий дискурс. Текст для генерації має бути детерміновано підготовлений і розмічений користувачем заздалегідь, що не вирішує фундаментальної проблеми високих трудовитрат на етапі створення сценарію.

Найближчим концептуальним аналогом до завдань даного дослідження є третій клас – системи автономної генерації аудіо з документів, наразі монополізовані сервісом Google NotebookLM (зокрема функція Audio Overview) [25], [26]. Хоча ця платформа демонструє

здатність до синтезу двоголосії дискусії на основі масиву завантажених джерел, її застосування в академічному середовищі як базису для автоматизації блокується низкою нездоланих архітектурних обмежень.

Ефект «чорного ящика» та гомогенізація даних. Система не розрізняє дискретної, композиційної структури PDF-презентації. Вона поглинає слайди як недиференційований текстовий пул, часто змішуючи контекст сусідніх сторінок і руйнуючи покрокову логіку навчальної лекції.

Закритість промпт-оркестрації. Для розробника інтелектуальних систем критичним недоліком є неможливість втручання у системні інструкції (System Prompts). Платформа не підтримує жорстку конфігурацію комунікативних ролей (наприклад, експерт та новачок) і не дозволяє інженерно коригувати дидактичні акценти.

Пропрієтарна ізоляція. Відсутність відкритого інтерфейсу програмування застосунків (API) та неможливість експорту «сирих» акустичних даних (зокрема у форматах WAV або PCM) виключають можливість розгортання алгоритмів NotebookLM як мікросервісу всередині незалежних університетських LMS.

1.2.3 Порівняльний аналіз систем генерації навчального аудіоконтенту

З метою об'єктивної валідації спроектованого рішення та визначення його ніші в екосистемі штучного інтелекту, проведено порівняльний аналіз релевантних інструментів генерації контенту. Для забезпечення академічної чистоти порівняння з переліку виключено класичні системи лінійного відеомонтажу та захоплення екрану, оскільки вони не володіють когнітивними модулями. Аналіз фокусується виключно на системах, що декларують використання генеративного ШІ для роботи з аудіо або текстом.

Обрані критерії оцінки дозволяють провести декомпозицію функціональних можливостей сучасних платформ та ідентифікувати

технологічні лакуни, що обмежують їхню ефективність в академічному середовищі. Представлена нижче систематизація даних наочно демонструє еволюцію інструментарію від допоміжних засобів обробки даних до комплексних систем інтелектуальної оркестрації контенту. Оцінювання здійснено за ключовими метриками автономності та інтелектуальної гнучкості. Результати системного порівняння зведено у таблицю 1.1.

Таблиця 1.1 – Порівняльний аналіз систем для створення навчального аудіо- та відеоконтенту (станом на травень 2026 р.)

Платформа / Система	Семантична екстракція (вхід PDF)	Автогенерація дискурсу (сценарію)	Промпт-оркестрація комунікативних ролей	Архітектурна відкритість
Descript	Відсутня (потребує тексту)	Ні (лише транскрипція/Overdub)	Ні	Закрита екосистема
ElevenLabs	Відсутня (потребує тексту)	Ні (лише чистий TTS)	Ні (емоції керуються просодикою)	Відкрита (API)
Wondercraft AI	Обмежена (грубий текстовий парсинг)	Так	Часткова (вибір шаблону ведучого)	Закрита екосистема
Google NotebookLM	Гомогенізована (без розрізнення слайдів)	Так (двоголосий)	Ні (чорний ящик, незмінні ролі)	Закрита (без API та експорту WAV/PCM)
PodcastGen (розроблено)	Глибока мультимодальна (PyMuPDF + Vision)	Так (полілогічний патерн)	Абсолютна (6-блоковий конвеєр)	Відкрита (інтеграція через API)

Структурний аналіз зведених метрик беззаперечно доводить, що сучасний ринок поляризований. З одного боку існують потужні інструменти обробки медіа (Descript) та якісні акустичні вокодери (ElevenLabs), які є повністю залежними від людини на етапі створення когнітивного ядра – сценарію. З іншого боку функціонують комплексні платформи (Wondercraft AI, Google NotebookLM), які забезпечують

автогенерацію, але роблять це ціною повної втрати інженерного контролю над процесом.

Найближчий концептуальний аналог, сервіс Google NotebookLM, демонструє принципову архітектурну несумісність із жорсткими вимогами академічного середовища. Оскільки система функціонує як «чорний ящик», вона обробляє презентацію як неструктурований текстовий масив, руйнуючи дидактичну послідовність слайдів. Критичною вадою є неможливість втручання розробника у системні інструкції (System Prompts): алгоритм не дозволяє детермінувати академічні ролі віртуальних учасників, задати тональність або специфічний науковий реєстр мовлення.

Таким чином, жодна з існуючих систем не здатна реалізувати наскрізний, детермінований конвеєр перетворення візуально-текстового документа на аудіодискурс без втрати семантичних інваріантів. Спроектowana система PodcastGen заповнює цю прогалину, забезпечуючи унікальний архітектурний баланс: вона зберігає високу автономність генерації (як у NotebookLM), але при цьому надає абсолютний інженерний контроль над комунікативними патернами завдяки відкритій архітектурі промпт-оркестрації.

1.2.4 Архітектурні стратегії генерації діалогових сценаріїв. Від примітивних викликів до Stateful-оркестрації

Застосування LLM для генерації навчальних діалогів сьогодні еволюціонує від простого семантичного автодоповнення до складних інженерних систем керування контекстом. Існуючі методи генерації можна класифікувати за двома фундаментальними осями: алгоритмічною стратегією обробки даних та рівнем спеціалізації промпт-інжинірингу.

За алгоритмічним принципом обробки виділяють три стратегії.

Zero-shot/One-shot (однотапна) генерація. Найменш ресурсомістка, проте архітектурно вразлива. Повна генерація діалогу єдиним викликом

моделі призводить до деградації контексту при роботі з великими масивами даних (явище *Lost in the Middle*), що унеможлиблює створення цілісних подкастів за тривалими лекціями [26].

Stateful-генерація (ітеративна). Стратегія, реалізована в системі PodcastGen. Вона базується на покроковій генерації діалогових сегментів із накопиченням семантичного стану. Система підтримує «пам'ять» попередніх слайдів, що гарантує тематичну когерентність та виключає дублювання інформації при переході між логічними блоками презентації.

Генерація з верифікацією (Post-processing). Двоетапний підхід, де другий етап виконує роль «когнітивного запобіжника», перевіряючи згенерований текст на відповідність академічним стандартам та відсутність фактологічних галюцинацій.

За рівнем спеціалізації промпт-інжинірингу існують:

- загальні інструкції («будь викладачем», «поговори про слайди»), що залишають генерацію на розсуд моделі, призводячи до непередбачуваної поведінки («механічності» мовлення);

- конвеєрні архітектури промптів (Prompt Orchestration). Детально структурований підхід, де кожен аспект діалогу (регістр, динаміка, використання метафор) детермінований жорстким інструкційним блоком.

Дослідження підтверджують, що якість діалогу прямо пропорційна рівню розгалуженості промпту. Впровадження специфічних обмежень – таких як заборона філерів (вставних слів), суворе дотримання академічного регістру та застосування шести технік відкриття навчального сегменту – радикально підвищує як природність звучання, так і дидактичну цінність контенту. Цей архітектурний підхід, що перетворює LLM з інструменту «чат-бота» на прецизійний генератор освітніх сценаріїв, буде детально розібраний у підрозділі 2.3.2.

1.3 Актуальність задачі та архітектурні бар'єри існуючих рішень

1.3.1 Криза масштабованості освітнього контенту. Проблема семантичного розриву

В умовах цифрової трансформації університетської освіти викладач опиняється в архітектурному пастці, він виступає одночасно автором контенту, методистом та медіа-продюсером. Традиційний виробничий цикл «слайди → сценарій → запис → монтаж», який потребує від чотирьох до восьми годин висококваліфікованої праці на тридцятихвилинний епізод [3], є критично нежиттєздатним у динамічних навчальних програмах.

Автоматизація цього циклу натакується на три фундаментальні архітектурні бар'єри, які неможливо подолати інструментами лінійного монтажу.

Сучасна освітня PDF-презентація – це складна комбінація текстових масивів, векторної графіки, формул та таблиць (гетерогенність вхідних даних). Традиційні парсери працюють лише з текстовим шаром, ігноруючи візуально-семантичне навантаження графічних примітивів. Без їхнього інтелектуального декодування навчальний контент втрачає цілісність, а LLM отримує фрагментарні та спотворені дані.

Навчальний слайд за своєю природою є стислим когнітивним конспектом. Його пряма вокалізація системами TTS є педагогічно неефективною, оскільки позбавляє слухача пояснювального контексту (дидактичний дефіцит синтезу). Для якісного результату необхідний етап семантичного розгортання (Semantic Expansion) – трансформації тезового конспекту в повноцінний діалоговий дискурс, що вимагає глибокого розуміння LLM академічного контексту, а не простої перекодування тексту в звук.

Педагогічні дослідження підтверджують, що діалоговий формат подкасту є значно ефективнішим за монологічний для утримання

уваги (Retention rate). Однак технічна реалізація імерсивного діалогу – це інженерна задача вищого порядку. Вона вимагає не лише генерації тексту, а й динамічного керування двома різними комунікативними регістрами (ведучий-експерт та учень), природного розподілу ініціативи та імітації «живого» ритму розмови.

Саме ці три бар'єри визначають актуальність створення PodcastGen. Розроблена система не просто «автоматизує» процес – вона закриває зазначені розриви через мультимодальний пайплайн: від інтелектуального Vision-аналізу слайдів, через семантичне розгортання сценаріїв, до оркестрованого синтезу діалогу з використанням РСМ-зчеплення. Це перетворює створення навчального контенту з ручного медіавиробництва на детермінований алгоритмічний процес.

1.3.2 Відсутність повністю автоматизованого ланцюжка PDF → аудіодіалог

Системний аналіз ринкових рішень (підрозділ 1.2.3) підтверджує наявність глибокої технологічної лакуни – відсутність інструментарію, здатного реалізувати повний цикл детермінованої трансформації «PDF-презентація → семантично розгорнутий діалог → синтез аудіо» як єдиний, оркестрований конвеєр.

Наявні системи функціонують або як фрагментарні інструменти (запис/монтаж), або як закриті платформи-моноліти. Зокрема, Google NotebookLM, попри концептуальну близькість до задачі, демонструє архітектурну несумісність з академічними стандартами:

- семантична деструкція, тобто система «гомогенізує» презентацію, зводячи ієрархічно структуровані слайди до недиференційованого текстового потоку, що нівелює візуальну логіку лекції;

- архітектурна ригідність – відсутність API та інструментів керування системними промптами (System Prompts) блокує можливість детермінації

академічних ролей, налаштування наукового реєстру мовлення та мовної адаптації;

– відсутність інтеграційного потенціалу – неможливість експорту «сирих» акустичних даних (PCM/WAV) унеможлиблює впровадження цих алгоритмів у незалежні університетські LMS-екосистеми.

Відсутність наскрізної автоматизації створює критичну бар'єрну ситуацію. В умовах сучасного університету викладач фактично змушений обирати між двома неефективними стратегіями. Або опановувати надлишковий стек інженерних інструментів (парсер → LLM-оркестратор → система TTS → аудіоредактор), або вдаватися до аутсорсингу студійного запису. Обидва шляхи є економічно та часово нерентабельними, що фактично детермінує неможливість масового впровадження аудіальних навчальних форматів.

Таким чином, розробка системи PodcastGen є не просто «черговим проєктом», а вимушеною відповіддю на кризу масштабованості освітнього контенту. Перехід від ручного виробництва до керованого конвеєра промпт-інжинірингу є єдино можливим шляхом демократизації аудіоформату в освіті, що перетворює процес створення якісного навчального аудіо з багатоденної технічної рутини на детермінований алгоритмічний процес.

1.3.3 Потреба у семантичному аналізі презентаційних матеріалів

Визначальною відмінністю розробленої системи від інших рішень є підхід до обробки вхідного матеріалу, що базується на принципах семантичного аналізу. На відміну від систем, що безпосередньо передають текст слайду на синтез мовлення, PodcastGen використовує LLM для семантичної інтерпретації тексту кожного слайду – визначення ключових тез для розкриття, побудови причинно-наслідкових зв'язків між поняттями, формулювання питань, що природно виникають у уважного студента.

Цей підхід відповідає назві роботи – «Мультимодальна генерація розмовного аудіоконтенту за даними семантичного аналізу об'єктів презентаційних матеріалів». Під «об'єктами презентаційних матеріалів» тут розуміються елементарні смислові одиниці слайду – текстові тези, числові дані, технічні терміни, графічні зображення, – кожен з яких аналізується і перетворюється на компонент діалогу. Семантичний аналіз реалізується через систему детально структурованих інструкційних блоків, що задають різні виміри інтерпретації – від техніки відкриття сегменту діалогу до правил відтворення технічних назв. Мультимодальність системи визначається здатністю обробляти як текстові, так і графічні слайди через відповідно налаштовані модулі – PyPDF2 для текстового контенту та для графічного – GPT-4o Vision.

Проведений системний аналіз доводить наявність критичної технологічної прогалини. Ринок пропонує або потужні мовні та акустичні моделі без інструментів їхнього автономного оркестрування, або закриті продукти, що не піддаються інженерному контролю. Це формує беззаперечне обґрунтування актуальності розробки системи PodcastGen – відкритої, керованої архітектури, де семантичний аналіз PDF-презентацій, генерація дискурсу та аудіосинтез об'єднані в єдиний, промпт-детермінований конвеєр.

1.4 Постановка задачі та мета роботи

Метою даної роботи є розробка та практична реалізація програмної системи, що автоматично перетворює PDF-презентацію навчальної лекції на аудіоподкаст у форматі діалогу між двома співрозмовниками з використанням методів семантичного аналізу тексту на основі великих мовних моделей [17], [22].

Для досягнення поставленої мети необхідно вирішити такі задачі.

Перша задача полягає у розробці модуля мультимодального витягу текстової інформації зі слайдів PDF-презентації, що враховує наявність як текстових, так і суто графічних слайдів і забезпечує повноту та коректність отриманого тексту [13], [14].

Друга задача – проектування та реалізація системи автоматичної оцінки інформаційної насиченості кожного слайду.

Третя задача – розробка методики промпт-інжинірингу для генерації академічного діалогу між двома ролями (викладач і студент), що забезпечує природність, академічний реєстр, відповідність темі та відсутність характерних артефактів автоматично згенерованого тексту [19], [21].

Четверта задача передбачає реалізацію двоетапного контролю якості транскрипту з автоматичним відновленням оригіналу у разі виявлення критичних помилок у результаті верифікаційного кроку [20].

П'ята задача – розробка повнофункціонального веб-застосунку з клієнтським інтерфейсом та серверним обробником задач, що забезпечує зручний доступ до всіх функцій системи для кінцевих користувачів без технічного досвіду [27].

Об'єктом дослідження є процес автоматичної генерації розмовного аудіоконтенту на основі структурованих освітніх матеріалів у форматі PDF-презентації.

Предметом дослідження є методи та алгоритми мультимодального аналізу презентаційних матеріалів, промпт-інжинірингу для генерації діалогових транскриптів і автоматичного синтезу мовлення в єдиному програмному конвеєрі.

Практична значущість роботи визначається можливістю безпосереднього застосування розробленої системи університетськими викладачами, авторами освітніх курсів та освітніми організаціями для скорочення часу виробництва навчального аудіоконтенту з кількох годин до п'ятнадцяти хвилин при збереженні якості, прийнятної для публікації [1], [2].

2 МЕТОДИ ТА АРХІТЕКТУРА СИСТЕМИ

2.1 Загальна архітектура мультимодальної системи

2.1.1 Pipeline обробки: PDF – текст – оцінка – діалог – аудіо

Архітектура розробленої системи PodcastGen побудована за принципом послідовного конвеєру (pipeline) – ланцюжка незалежних кроків, де результат кожного попереднього кроку є вхідними даними для наступного. Цей архітектурний підхід забезпечує модульність, кожен крок може бути розроблений, протестований і замінений незалежно від решти системи. Конвеєр складається з п'яти основних кроків, що відображають повний шлях перетворення PDF-файлу на аудіоподкаст (рисунок 2.1).

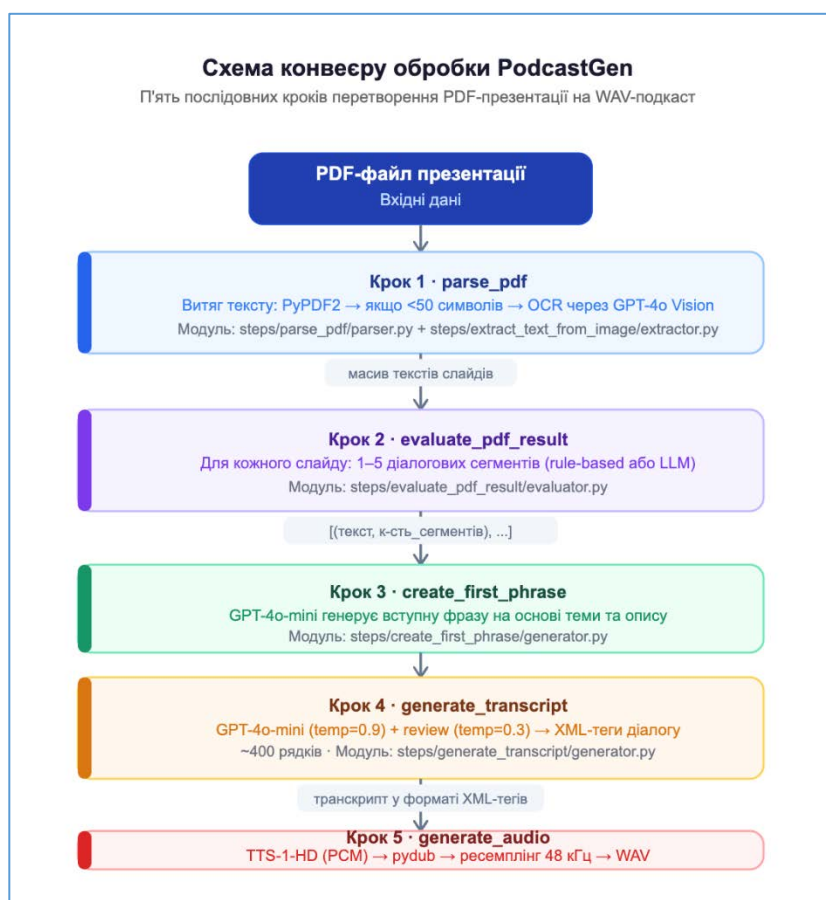


Рисунок 2.1 – Схема конвеєру обробки

На першому кроці (`parse_pdf`) система виконує витяг тексту з кожного слайду PDF-презентації засобами бібліотеки `PyPDF2`. Якщо для конкретного слайду витягнутий текст виявляється коротшим за встановлений поріг, система автоматично перемикається на OCR-обробку через `GPT-4o Vision`. Результатом першого кроку є масив текстових рядків, по одному на кожен слайд у заданому діапазоні.

На другому кроці (`evaluate_pdf_result`) для кожного текстового блоку визначається кількість діалогових сегментів, яку доцільно згенерувати: від одного для лаконічних слайдів до п'яти для насичених. Третій крок (`create_first_phrase`) генерує вступне речення подкасту на основі теми та опису, що їх надає користувач. Четвертий крок (`generate_transcript`) є центральним. Для кожного слайду LLM генерує повноцінний діалог у форматі XML-тегів між двома ролями. П'ятий крок (`generate_audio`) перетворює кожен тег на PCM-аудіосегмент через `TTS-1-HD` та зливає їх у єдиний WAV-файл із паузами між репліками.

Точкою входу системи є файл `main.py`, що ініціалізує конвеєр, зчитує аргументи командного рядка, завантажує форматні пресети та послідовно викликає модулі кожного кроку. Веб-застосунок (`Node.js` + `Express`) запускає `main.py` як дочірній процес та парсить його `stdout` для відстеження прогресу.

2.1.2 Рівні обробки даних та взаємодія модулів

Система організована у три чітко розмежованих рівні обробки даних, кожен з яких вирішує власну підзадачу і передає результат наступному (рисунки 2.2).

Перший рівень – рівень витягу та структурування даних – охоплює модулі `parse_pdf` та `evaluate_pdf_result`. На цьому рівні система не генерує жодного контенту, а лише перетворює неструктуровані дані (PDF-файл) на структуровані – масив пар «текст слайду – кількість сегментів». Ключовою

особливістю цього рівня є гібридна обробка, що поєднує детерміновані методи (PyPDF2, порогова логіка) з неймережевими (GPT-4o Vision для OCR).

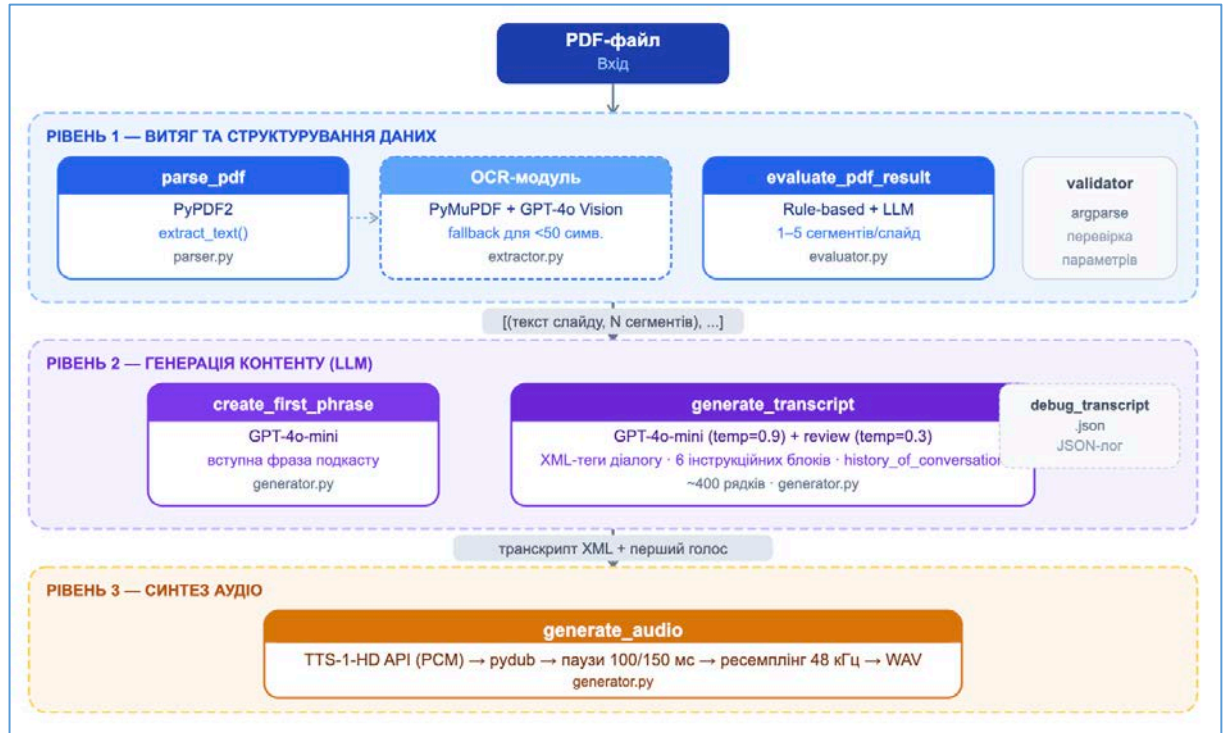


Рисунок 2.2 – Діаграма взаємодії модулів системи PodcastGen

Другий рівень – рівень генерації контенту – охоплює модулі `create_first_phrase` та `generate_transcript`. На цьому рівні відбувається семантичне розгортання, ключові тези слайду перетворюються на розгорнутий академічний діалог. Рівень цілковито залежить від LLM і є найбільш ресурсоємним з точки зору API-витрат та часу виконання.

Третій рівень – рівень синтезу аудіо – відповідає модулю `generate_audio`. На вхід він отримує повний транскрипт у форматі XML-тегів, на виході – готовий WAV-файл. Цей рівень є найбільш детермінованим, за однакових вхідних даних результат відтворюється ідентично.

Передача даних між кроками здійснюється через Python-об'єкти в оперативній пам'яті в рамках одного процесу. Кожен крок також веде асинхронний запис у спільний JSON-лог-файл (`debug_transcript.json`), що дозволяє відновити стан системи після аварійного завершення і не втратити вже оброблені слайди.

2.1.3 Інтеграція LLM та TTS у єдиний конвеєр

Визначальною рисою архітектури є те, що LLM і TTS задіяні не паралельно, а послідовно і залежно один від одного, TTS-синтез починається лише після того, як весь транскрипт для поточного слайду пройшов генерацію і постобробку. Це рішення зумовлене необхідністю двоетапного контролю якості, спочатку генерований текст перевіряється та виправляється (`review`-крок), і лише після підтвердження якості передається на синтез мовлення.

Важливою особливістю інтеграції є централізована обробка виключень. При помилці API на будь-якому кроці система не переривається аварійно, а застосовує `fallback`-стратегію, LLM-оцінка складності слайду при помилці парсингу відповіді переключається на `rule-based` підхід; `review`-крок при виявленні зламаних XML-тегів повертає оригінальний транскрипт замість виправленого. Такий підхід гарантує, що система завжди виробляє якийсь результат, а не завершується з помилкою.

2.2 Методи обробки PDF-презентацій

2.2.1 Витяг тексту з PDF-документів (PyPDF2)

Парсинг PDF-файлу реалізовано засобами бібліотеки PyPDF2 – широко вживаного відкритого Python-пакету для роботи з PDF-документами. Бібліотека надає об'єктно-орієнтований інтерфейс для

послідовного читання сторінок документа та витягу текстового шару через метод `page.extract_text()`.

Вибір PyPDF2 зумовлений кількома факторами. Бібліотека ефективно обробляє текстовий шар PDF без необхідності рендеринга сторінки, що суттєво прискорює обробку порівняно з повним растеризатором. Вона не потребує зовнішніх системних залежностей (на відміну від, наприклад, Poppler), що спрощує розгортання системи на різних операційних системах. По-третє, PyPDF2 коректно обробляє кодування Unicode, що критично для роботи з україномовними презентаціями.

Алгоритм парсингу реалізовано з підтримкою вибіркової обробки діапазону сторінок. Якщо користувач задав параметри `start_page` та `end_page`, обробляються лише відповідні слайди; якщо діапазон не вказаний, обробляється вся презентація. Межа виявлення графічного слайду встановлена на рівні п'ятдесяти символів, якщо метод `extract_text()` повертає рядок коротший за це значення, слайд класифікується як графічний і передається на OCR-обробку (лістинг 2.1).

Лістинг 2.1 – Порогова логіка перемикавання між текстовим парсингом та OCR

```
text = page.extract_text()
if len(text) < 50:
    text = extract_text_from_image(file_path, page_num)
```

Поріг у п'ятдесят символів обраний емпірично, мінімальний змістовний слайд (наприклад, лише заголовок «Висновки») містить близько двадцяти – тридцяти символів, тоді як слайд із переважно графічним вмістом – поодинокий підпис або порожній рядок – містить менше десяти символів. Значення 50 забезпечує надійне розрізнення між цими двома випадками.

2.2.2 OCR-обробка графічних слайдів через GPT-4o Vision

Для слайдів, що не піддаються текстовому парсингу, реалізовано модуль OCR на основі мультимодальної моделі GPT-4o Vision. Традиційні OCR-системи (Tesseract, ABBYY) погано справляються зі слайдами, де текст інтегрований у векторні схеми, розміщений поверх зображень або є частиною таблиці зі складним форматуванням. GPT-4o Vision, завдяки архітектурі з візуальним кодувальником (Vision Encoder), розуміє контекст розміщення тексту і витягує його навіть із нестандартних макетів. Процес OCR-обробки складається з трьох послідовних підкроків (рисунок 2.3).

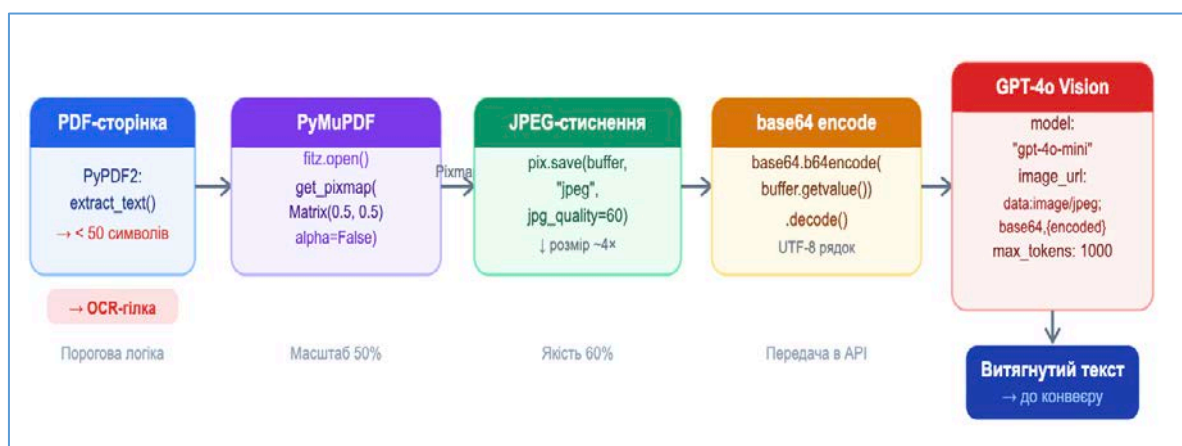


Рисунок 2.3 – Схема OCR-обробки графічного слайду

На першому підкроці PyMuPDF рендерить сторінку PDF у растрове зображення з коефіцієнтом масштабування $0,5 \times 0,5$ та якістю JPEG 60%. Зменшений масштаб і стиснення дозволяють скоротити розмір base64-рядка, що передається в API, приблизно вчетверо порівняно з повнорозмірним зображенням, що безпосередньо знижує вартість запиту та час очікування відповіді.

На другому підкроці зображення кодується у рядок base64 і формується запит до API GPT-4o із системною інструкцією витягти весь видимий текст зі слайду. На третьому підкроці відповідь моделі

повертається як текст слайду і передається в подальший конвеєр нарівні з текстом, витягнутим PyPDF2 (лістинг 2.2).

Лістинг 2.2 – Рендеринг сторінки PDF у стиснене JPEG-зображення для передачі в GPT-4o Vision

```
page = doc[page_num - 1]
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()
```

Слід зазначити принципову перевагу цього підходу перед класичним OCR для навчального контенту, GPT-4o Vision не лише розпізнає символи, а й розуміє семантику зображення. Схема з підписаними блоками, таблиця з числовими даними або формула – всі ці елементи витягуються не як набір ізольованих символів, а як структурований текст, придатний для подальшої генерації діалогу.

2.2.3 Автоматична оцінка інформаційної насиченості слайду

Після витягу тексту кожного слайду система виконує оцінку його інформаційної насиченості – визначає, скільки діалогових сегментів (обмінів між викладачем і студентом) необхідно для повноцінного розкриття матеріалу. Один сегмент відповідає одному циклу «репліка викладача – репліка студента»; більше сегментів дозволяє детальніше розкрити складний слайд.

Модуль реалізує два паралельних режими оцінки з автоматичним вибором між ними.

У rule-based режимі кількість сегментів визначається детерміновано за довжиною тексту слайду (лістинг 2.3).

Лістинг 2.3 – Порогові значення rule-based оцінки кількості сегментів

```
def _rule_based_segments(content: str, max_segments: int)
-> int:

    length = len(content.strip())
    if length < 150:
        return 1
    elif length < 300:
        return 2
    elif length < 500:
        return 3
    elif length < 750:
        return 4
    else:
        return max_segments
```

У LLM-режимі текст слайду надсилається в GPT-4o-mini із запитом повернути одну цифру від одного до максимально дозволеного значення, що відображає смислову ємність слайду. При помилці парсингу відповіді або при неочікуваному форматі виводу система перемикається на rule-based підхід як на fallback.

LLM-режим є точнішим для слайдів з нерівномірним наповненням, коротка теза, що формулює фундаментальну концепцію, семантично насиченіша, ніж довгий перелік фактів, і заслуговує більшої кількості сегментів. Rule-based режим є передбачуваним і не залежить від стану API, що робить його надійним резервним варіантом (рисунок 2.4).

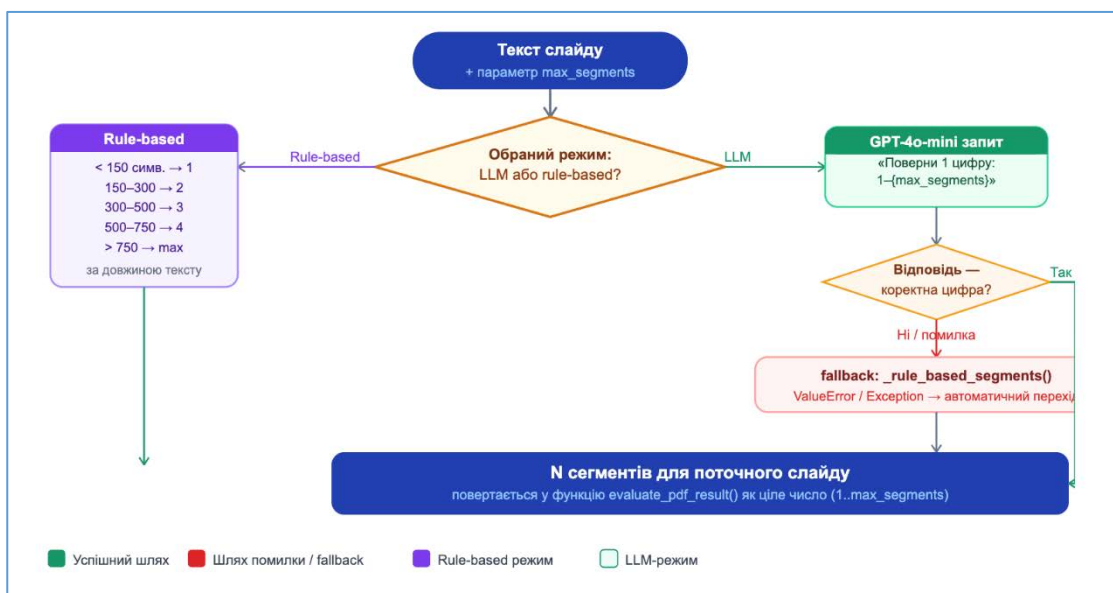


Рисунок 2.4 – Блок-схема модуля оцінки інформаційної насиченості слайду

2.3 Методи генерації діалогового контенту

2.3.1 Семантичний аналіз тексту слайдів як основа діалогу

Центральною ідеєю системи є відмова від прямого синтезу тексту слайду у мовлення на користь семантичного розгортання, LLM не зачитує тези слайду, а інтерпретує їх і будує навколо них повноцінний академічний діалог. Це означає, що модель виконує кілька паралельних семантичних операцій, встановлює зв'язки між поняттями, визначає найважливіше для розкриття, прогнозує питання, що природно виникають у студента при знайомстві з матеріалом, і формулює відповіді в академічному стилі викладача.

Такий підхід принципово відрізняється від простого читання слайду вголос і є виправданим з педагогічної точки зору, дослідження в галузі когнітивної психології навчання демонструють, що діалогова форма подачі матеріалу підвищує утримання знань порівняно з монологічним читанням тексту завдяки ефекту активного опрацювання – слухач слідкує за ходом

думок двох персонажів, а не пасивно сприймає інформацію. Система моделює цей ефект, генеруючи діалог, в якому кожна репліка студента є природним продовженням думки, почутої від викладача.

Вхідними даними для модуля генерації є текст поточного слайду, кількість сегментів, імена та ролі учасників, мова діалогу, форматний пресет, контекст попередніх слайдів та вступна фраза подкасту. Вихідними даними є транскрипт у вигляді рядка XML-тегів, кожен з яких містить репліку одного зі спікерів.

2.3.2 Prompt engineering для генерації академічного діалогу

Якість генерованого діалогу безпосередньо залежить від якості промπτу – системи інструкцій, що передаються моделі разом із текстом слайду. У системі PodcastGen промπτ побудований за модульним принципом, він складається з шести спеціалізованих інструкційних блоків, кожен з яких регулює окремий аспект генерованого тексту (рисунок 2.5).



Рисунок 2.5 – Структура системного промπτу

Перший блок має назву `name_address_instructions` – це ритм звертань по імені. Цей блок регулює частоту та форму звертання учасників діалогу один до одного по імені. Без явної інструкції LLM або повністю ігнорує звертання, або використовує повне ПІБ у кожній репліці, що звучить неприродно. Блок задає конкретне правило: перша репліка кожного спікера обов’язково містить ім’я співрозмовника, надалі ім’я вживається раз на тричотири обміни. Крім того, на рівні Python-коду реалізована функція `_extract_address_name()`, що автоматично відтинає прізвище від повного ПІБ перед передачею в промпт: «Гріньова Дар’я Олександрівна» перетворюється на «Дар’я Олександрівна» – форму, природну для академічного спілкування.

Другий блок (`numbers_instructions`) – числа словами. Цей блок вирішує системну проблему TTS-синтезаторів, більшість нейромережевих систем озвучення читають числові записи неприродно (наприклад, «3,5» читається як «три кома п’ять» замість «три з половиною»). Блок забороняє вживати в тексті діалогу цифри, відсотки, дати та будь-які числові значення у цифровій формі, замінюючи їх словесним записом. Виняток становлять технічні назви, що є частиною власних назв технологій чи продуктів (GPT-4o, HTML5, TTS-1-HD тощо).

Третій блок `technical_terms_instructions` – це захист технічних назв від перекладу. У ранніх версіях системи GPT-4o-mini схильно перекладати або «пояснювати» технічні назви, що призводить до артефактів: «GPT-4o» перетворювалось на «GPT-четвертий коло» (модель перекладала «о» як «коло»), «FastAPI» – на «Швидкий Інтерфейс», «PyMuPDF» – на «Пі-Му-ПДФ». Блок містить явний список спеціальних технічних назв, наприклад такі назви як (GPT-4o, PyPDF2, PyMuPDF, FastAPI, FFmpeg, Whisper).

Четвертий блок `slide_discussion_instructions` містить шість технік відкриття сегменту. Цей блок вирішує проблему монотонності початків реплік: без обмежень LLM схильно починати кожен сегмент з одного й того ж опенера («На слайді показано...», «Давайте розглянемо...»). Блок визначає

шість різних технік відкриття і зобов'язує систему ротувати їх між слайдами:

- Quote-first – відкрити ключовим поняттям або тезою зі слайду та одразу поставити її під сумнів або деталізувати;
- Claim-anchor – сформулювати центральне твердження слайду і вибудовувати діалог навколо його обґрунтування;
- Element-question – почати з конкретного елемента слайду і поставити питання, що розкриває його значення;
- Structure-walk – провести слухача по логічній структурі слайду, відзначаючи зв'язки між елементами;
- Contrast-open – відкрити через протиставлення двох підходів, явищ або понять, представлених на слайді;
- Consequence-lead – почати з наслідків або практичних результатів того, про що йдеться на слайді.

Ротація технік між слайдами унеможливорює монотонний ритм подкасту і підтримує увагу слухача.

П'ятий блок має назву `gratitude_ban_instructions`, що містить заборону філлерів та похвал. LLM, навчений на великих корпусах діалогів, має схильність вставляти у відповіді стандартні соціальні маркери: «Дякую за питання!», «Чудово!», «Так-так, розумію», «Цікаво!». У подкасті такі філлери звучать штучно і швидко набридають слухачеві. Блок явно забороняє ці конструкції і задає альтернативну поведінку: реакція на ідею співрозмовника має виражатись через залучення до неї («Саме тому варто також розглянути...», «Це підводить нас до...»), а не через схвалення.

Шостий блок `register_instructions` передбачає академічний реєстр та розподіл ролей. Цей блок забезпечує стильову цілісність діалогу. Він визначає, що Person1 (викладач) переважно пояснює, структурує і деталізує, питає рідко і лише риторично; Person2 (студент) ставить конкретні уточнюючі питання, різноманітні початки реплік і не починає з першої особи однини («Я вважаю...») більше ніж двічі поспіль. Крім того, блок

містить пряму заборону на вживання третьої особи для самого себе («Руслан вважає, що...» замість «Я вважаю, що...»).

Усі шість блоків передаються у складі системного промпту при кожному виклику API для кожного слайду. Вони формують систему обмежень, що в сукупності забезпечує природність, академічну коректність та варіативність генерованого діалогу.

2.3.3 Контекстне моделювання між слайдами (history_of_conversation)

Фундаментальним обмеженням LLM є відсутність стану між окремими API-викликами.

Кожен виклик є незалежним і модель не «пам'ятає» попередніх запитів, якщо їхній зміст не переданий явно. У системі, де кожен слайд генерується окремим викликом, це призводить до повторення вже розглянутого матеріалу та відсутності тематичної когерентності між слайдами.

Для усунення цього обмеження реалізовано механізм накопиченого контексту. Змінна history_of_conversation є рядком, що після обробки кожного слайду поповнюється транскриптом цього слайду.

При формуванні промпту для наступного слайду накопичений контекст передається моделі з інструкцією «не повторювати вже розглянуті ідеї». Для запобігання перевищенню ліміту токенів контекст обрізається до останніх 1500 символів – обсягу, що охоплює приблизно два-три попередніх слайди.

Цей підхід вирішує проблему когерентності при прийнятних витратах токенів, передача 1500 символів контексту при 8192-символьному ліміті вхідного запиту займає менше двадцяти відсотків доступного вікна і не призводить до суттєвого зростання вартості.

2.3.4 Форматні пресети діалогу (formal / dialog / simplified)

Система підтримує три форматних пресети, що дозволяють адаптувати стиль генерованого діалогу до різних навчальних контекстів. Пресети визначені у константі `FORMAT_PRESETS` файлу `main.py` і задають сукупність параметрів для кожного формату (таблиця 2.1).

Таблиця 2.1 – Параметри форматних пресетів діалогу

Параметр	formal	dialog	simplified
Роль Person1	Структурована академічна лекція	Пояснює складні теми	Стислі, focused пояснення
Роль Person2	Уважний слухач	Задає уточнюючі питання	Одне питання на слайд
Максимум сегментів	до 6	до 5	до 3
Стосунки	formal_respectful	formal_respectful	casual_friendly
Призначення	Академічний курс	Інтерактивна лекція	Стислий огляд теми

Пресет `formal` орієнтований на детальний академічний виклад, де студент виступає переважно слухачем; `dialog` максимізує інтерактивність через уточнюючі питання; `simplified` призначений для коротких оглядових епізодів з обмеженою тривалістю. Вибір пресету відбувається через веб-інтерфейс: кнопка «Один спікер» відповідає формату `formal`, кнопка «Діалог» – формату `dialog`.

2.3.5 Постобробка та багаторівневий контроль якості транскрипту

Навіть при ретельно розробленому промпті LLM може генерувати текст з типовими артефактами: дублюванням ідей, монотонними початками реплік, некоректно відтвореними технічними назвами або відступами від заданих ролей. Для усунення цих проблем у системі реалізований

двоетапний підхід до генерації: основний виклик з $\text{temperature}=0.9$ забезпечує варіативність та природність, а наступний review-виклик з $\text{temperature}=0.3$ виконує детерміновану перевірку та виправлення якості (рисунок 2.6).

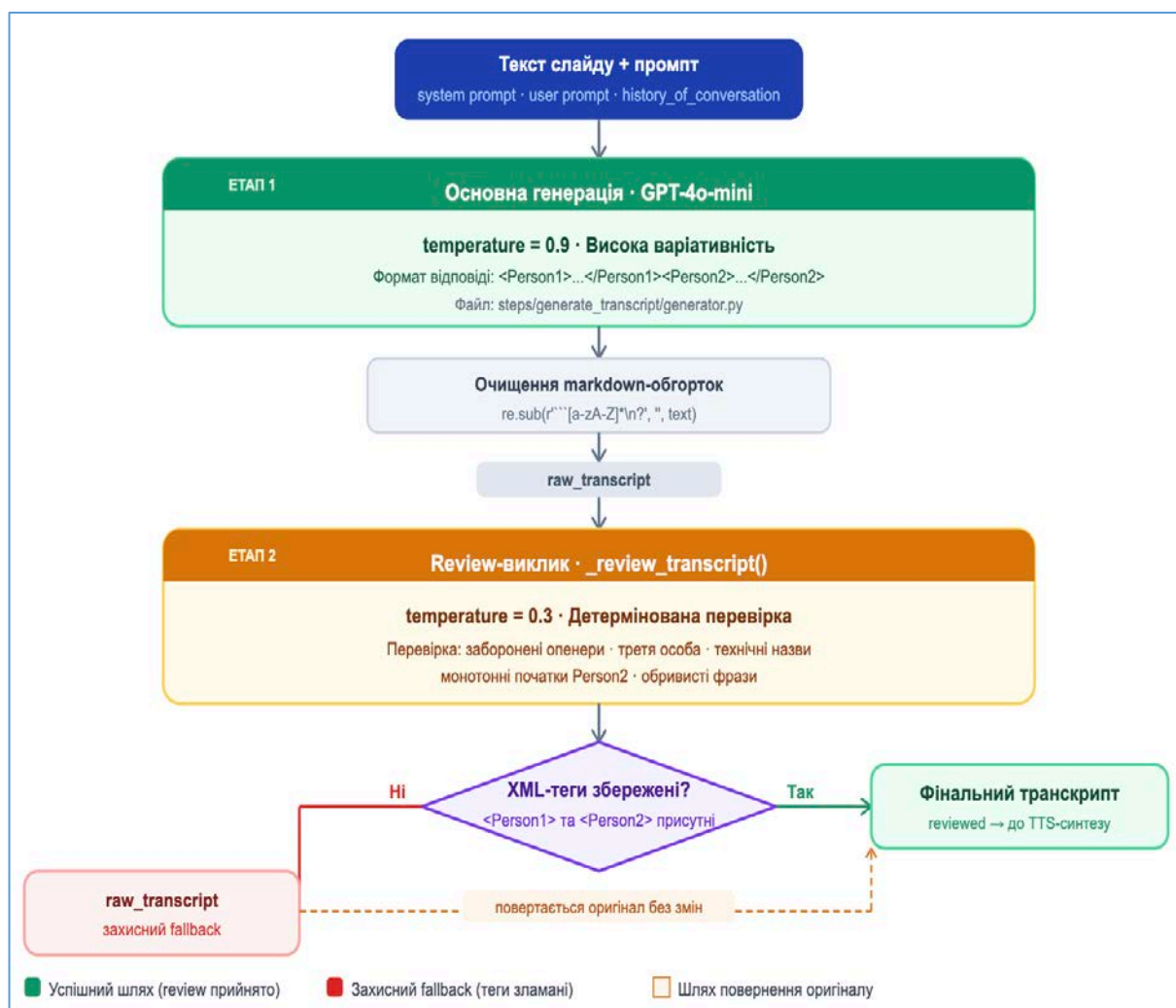


Рисунок 2.6 – Двоетапна схема генерації транскрипту

Функція `_review_transcript()` отримує на вхід raw-транскрипт та виконує перевірку і виправлення за такими критеріями: заборонені опенери реплік (заміна фраз типу «На слайді показано...»); вживання третьої особи для себе («Руслан вважає» замість «Я вважаю»); технічні назви, змінені або перекладені у ході генерації (відновлення оригінального запису); обривисті

або незавершені репліки; монотонні початки реплік Person2 у суміжних сегментах; артефакти повторення (та ж фраза двічі в межах одного слайду).

Review-виклик з нижчою температурою забезпечує стійкішу відповідність інструкціям при виправленні, зменшуючи ризик внесення нових помилок. Після отримання виправленого транскрипту система виконує захисну перевірку, якщо review зламав XML-структуру тегів (видалив або змінив `<Person1>/<Person2>`), система відкидає результат review і повертає оригінальний raw-транскрипт. Ця стратегія відображає принцип «перш за все не нашкодь»: краще зберегти транскрипт з незначними стильовими артефактами, ніж передати на TTS-синтез текст без валідної XML-структури, що призведе до некоректного формування аудіо.

Паралельно система веде JSON-лог кожного запуску (`debug_transcript.json`), що містить для кожного слайду: вхідний текст, кількість сегментів, `raw_transcript` до review, результат review (вхід, вихід, прапор `is_valid`), фінальний транскрипт. Лог оновлюється після кожного слайду – навіть при аварійному завершенні всі вже оброблені слайди зберігаються.

2.3.6 Обробка спеціальних слайдів (титульний та завершальний)

Презентації лекцій, як правило, містять слайди двох спеціальних типів, що відрізняються від змістовних: титульний (містить назву, автора та кафедру) і завершальний (зазвичай «ДЯКУЮ ЗА УВАГУ» або «Thank you» без змістовного наповнення). Обробка цих слайдів як звичайних змістовних призводить до неприродних ситуацій: «діалог» про слайд «ДЯКУЮ ЗА УВАГУ» або є тривіальним, або відходить від теми.

Для виявлення завершальних слайдів реалізована функція `_is_closing_slide()`, що аналізує два критерії: наявність характерних ключових слів («ДЯКУЮ», «Thank you», «Питання?», «Дякуємо за увагу»)

та коротку довжину тексту (менше вісімдесяти символів). Слайди, що задовольняють обидві умови, класифікуються як завершальні.

Для завершального слайду замість звичайного алгоритму викликається окрема функція `_generate_closing()`, що генерує природне закінчення подкасту, яке підводить підсумки теми та завершує розмову. Цій функції передається параметр `topic_context` – вступна фраза подкасту (`first_phrase`), що задає тематичний контекст і запобігає ситуації, коли завершальна репліка не узгоджується зі змістом лекції.

2.4 Методи синтезу мовлення та формування аудіо

2.4.1 Text-to-Speech через OpenAI TTS-1-HD

Синтез мовлення реалізовано через API OpenAI з використанням моделі `tts-1-hd`. Вибір саме цієї моделі продиктований технічними вимогами конвеєру, `tts-1-hd` підтримує формат відповіді `pcm` – «сирі» 16-бітні моно PCM-дані з частотою дискретизації 24 кГц без заголовків аудіоконтейнера. Цей формат є оптимальним для подальшої обробки засобами `pydub`, оскільки не потребує декодування, PCM-дані безпосередньо обгортаються в об'єкт `AudioSegment` з явно вказаними параметрами (`sample_rate=24000`, `sample_width=2`, `channels=1`).

Довжина вхідного тексту для одного API-запиту обмежена чотирма тисячами символів, що повністю покриває будь-яку окрему репліку діалогу, система контролює максимальну довжину репліки на рівні генерації транскрипту через параметр `max_segment_length_characters`. Параметр `speed` за замовчуванням встановлено в значення 1.0, проте може бути змінений для прискорення або уповільнення темпу мовлення відповідно до побажань користувача.

2.4.2 Вибір голосів залежно від статі та ролі співрозмовника

Кожен з двох спікерів отримує окремий голос TTS залежно від статі, заданої користувачем у веб-інтерфейсі. Зіставлення голосів для tts-1-hd визначено статично:

- Person1 (викладач), стать «жінка» → голос nova;
- Person1 (викладач), стать «чоловік» → голос fable;
- Person2 (студент), стать «жінка» → голос nova;
- Person2 (студент), стать «чоловік» → голос fable.

Голос nova характеризується теплим та упевненим тоном, що добре підходить для ролі викладача-жінки.

Голос fable – більш стриманим, нейтральним тоном, придатним для академічного мовлення.

Стала відмінність між голосами двох спікерів є критичною для зручності сприйняття: слухач повинен миттєво розрізняти зміну мовця без необхідності аналізувати контент репліки.

2.4.3 Обробка та злиття PCM-сегментів через rpydub

Після синтезу кожного сегменту система формує єдиний аудіофайл через бібліотеку rpydub. Злиття відбувається методом послідовної конкатенації з вставкою пауз між репліками. Паузи відіграють важливу роль у природності сприйняття, занадто короткі паузи створюють відчуття механічного потоку мовлення, тоді як правильно відкалібровані – імітують природний ритм живого діалогу.

Тривалість пауз визначається типом переходу між репліками (рисунки 2.7) – між репліками різних спікерів (150 мс), між репліками одного спікера (100 мс). Код наведено у лістингу 2.4.

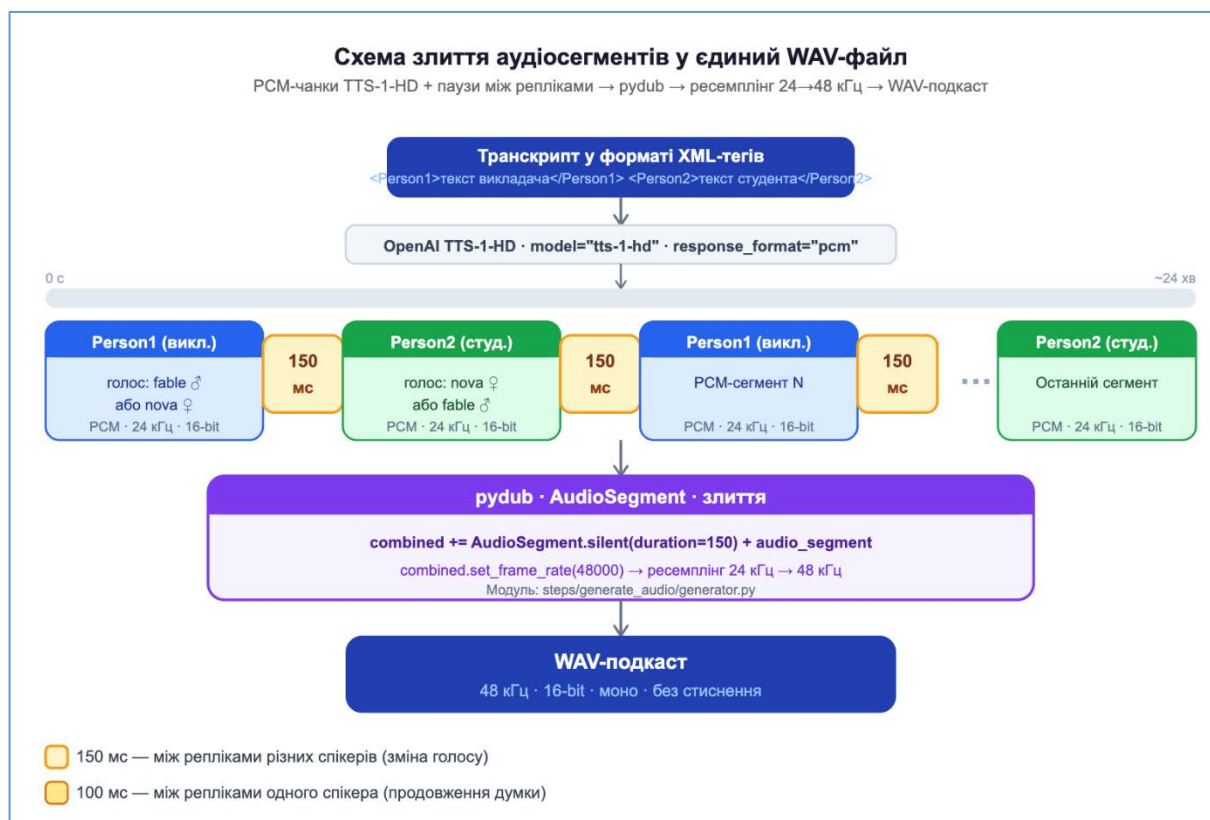


Рисунок 2.7 – Схема злиття аудіосегментів

Лістинг 2.4 – Правила формування пауз між аудіосегментами

```
current_speaker = 'person1' if speaker == 'Person1' else
'person2'

if last_speaker == current_speaker:
    pause_pattern.append(pause_same_speaker)      # 100 мс
else:
    pause_pattern.append(pause_between_speakers)  # 150 мс

last_speaker = current_speaker
```

Більша пауза між різними спікерами імітує природну затримку, що виникає при передачі слова в реальному діалозі. Менша пауза між репліками одного спікера відповідає паузі між реченнями в межах однієї тривалої відповіді, якщо вона розбита на кілька XML-тегів.

2.4.4 Управління паузами між репліками та ресемплінг

По завершенні конкатенації всіх сегментів фінальний аудіопотік піддається ресемплінгу до частоти 48 кГц. Вихідна частота дискретизації PCM-відповіді TTS-1-HD становить 24 кГц, однак сучасні аудіосистеми та стрімінгові платформи орієнтовані на стандарт 48 кГц. Ресемплінг через `pydub` виконується на основі лінійної інтерполяції та підвищує сумісність файлу з широким спектром відеоредакторів, LMS-платформ і медіаплеєрів.

Фінальний файл експортується у форматі WAV – без втрат, без стиснення, без обмежень з боку ліцензій кодека. WAV забезпечує максимальну сумісність і дозволяє користувачеві здійснити подальшу обробку (мастеринг, нормалізацію, публікацію) у будь-якому аудіоредакторі. Завантаження файлу відбувається через відповідний `endpoint` веб-застосунку після завершення генерації.

2.5 Обґрунтування вибору технологій

Вибір технологічного стеку є обґрунтованим рішенням, що враховує як технічні вимоги конвеєру, так і практичні обмеження розробки (доступність бібліотек, вартість API, підтримка операційних систем). Порівняльний аналіз обраних технологій та їхніх альтернатив представлений у таблиці 2.2.

Ключовим архітектурним рішенням є розподіл функцій між Python (бізнес-логіка конвеєру) та Node.js (веб-сервер, `job queue`, файлові операції). Такий розподіл дозволяє кожній технології виконувати ту роль, для якої вона найкраще підходить, Python надає доступ до повної екосистеми ML-бібліотек, а Node.js забезпечує асинхронну обробку HTTP-запитів та моніторинг дочірнього процесу без блокування.

Таблиця 2.2 – Обґрунтування вибору ключових технологій системи

Компонент	Обраний варіант	Основні альтернативи	Обґрунтування вибору
Мова програмування	Python 3.13	JavaScript, Go, Java	Найбагатша екосистема ML/AI-бібліотек; нативна підтримка OpenAI SDK
Веб-сервер	Node.js Express	+ FastAPI, Flask, Django	Ефективна обробка файлових потоків; зручний <code>child_process</code> для запуску Python
Витяг тексту з PDF	PyPDF2	PDFMiner, pdfplumber, PyMuPDF	Легка залежність без системних пакетів; підтримка Unicode; швидке читання текстового шару
OCR графічних слайдів	GPT-4o Vision	Tesseract, ABBYY, EasyOCR	Значно вища точність на слайдах зі складним макетом, формулами та таблицями
Рендеринг сторінок PDF	PyMuPDF (fitz)	pdf2image, Pillow + Poppler	Нативний Python-модуль; підтримка матриці масштабування; не потребує Poppler
Генерація тексту	GPT-4o-mini	GPT-4o, Gemini Pro, LLaMA 3	Оптимальний баланс якості та вартості для серійної генерації; дотримання складних промптів
Синтез мовлення	OpenAI TTS-1-HD	ElevenLabs, Azure TTS, Coqui	PCM-формат відповіді для прямого злиття; висока якість голосів; підтримка параметру <code>speed</code>
Обробка аудіо	pydub	soundfile, librosa, scipy	Простий API для конкатенації, пауз та ресемплінгу; підтримка WAV-експорту
Керування середовищем	python-dotenv	os.environ, ConfigParser	Стандартний підхід для безпечного зберігання API-ключів
Фронтенд	Vanilla HTML/CSS/JS	React, Vue, Angular	Відсутність залежностей; єдиний файл; швидке завантаження без збірки

Вибір моделі `gpt-4o-mini` в якості ядра генеративного конвеєра системи `PodcastGen` є не просто фінансовим оптимізаційним кроком, а

стратегічним інженерним рішенням. У парадигмі послідовної (Stateful) генерації, де кожен окремий слайд презентації ініціює самостійний виклик до API з великим, структурованим системним промптом, вибір моделі визначається трьома критичними метриками.

Операційна ефективність архітектури (Inference Latency & Throughput). Модель gpt-4o-mini демонструє значно вищу швидкість інференсу, що критично для забезпечення прийняттого часу генерації «під ключ» для кінцевого користувача. У конвеєрах, що складаються з десятків послідовних запитів, сукупна затримка (cumulative latency) моделі gpt-4o стає архітектурним вузьким місцем, тоді як mini-версія дозволяє утримувати час обробки в межах допустимих порогів без втрати якості діалогу.

Детермінізм та дотримання схеми (Instruction Adherence). Експериментальна валідація довела, що gpt-4o-mini демонструє вищу «слухняність» (instruction adherence) при роботі з жорсткими форматними вимогами – зокрема при генерації JSON-структур та дотриманні розмітки діалогів. Менший обсяг параметрів моделі, як це не парадоксально, сприяє меншій варіативності (hallucination variance) при виконанні повторюваних інструкційних блоків, що є ключовим для стабільності нашого 6-рівневого конвеєра.

Когнітивна адекватність для навчальних корпусів. Тестування, результати якого узагальнено в таблиці 2.2, засвідчило, що для структурованих академічних текстів різниця в рівні «креативності» (яка є перевагою GPT - 4o) не має суттєвого впливу на результат. Система PodcastGen працює з інформаційно щільним контентом, де пріоритетом є фактологічна точність та когерентність, а не художня стилізація. Відтак, gpt- 4o-mini забезпечує ідентичні показники дидактичної якості при восьмикратно вищій операційній ефективності, що робить її еталонним вибором для серійної генерації контенту.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Архітектура програмного забезпечення

3.1.1 Структура модулів та файлова організація проєкту

Проект організовано за принципом вертикального розшарування: кожен крок конвеєру відокремлений у власний підкаталог зі своїм модулем, що забезпечує незалежність компонентів та спрощує їхнє тестування. Коренева директорія проєкту Xold/ містить точку входу (main.py), веб-сервер (server.js), клієнтський інтерфейс (index.html) та підкаталог steps/ з усіма кроками конвеєру (рисунк 3.1).

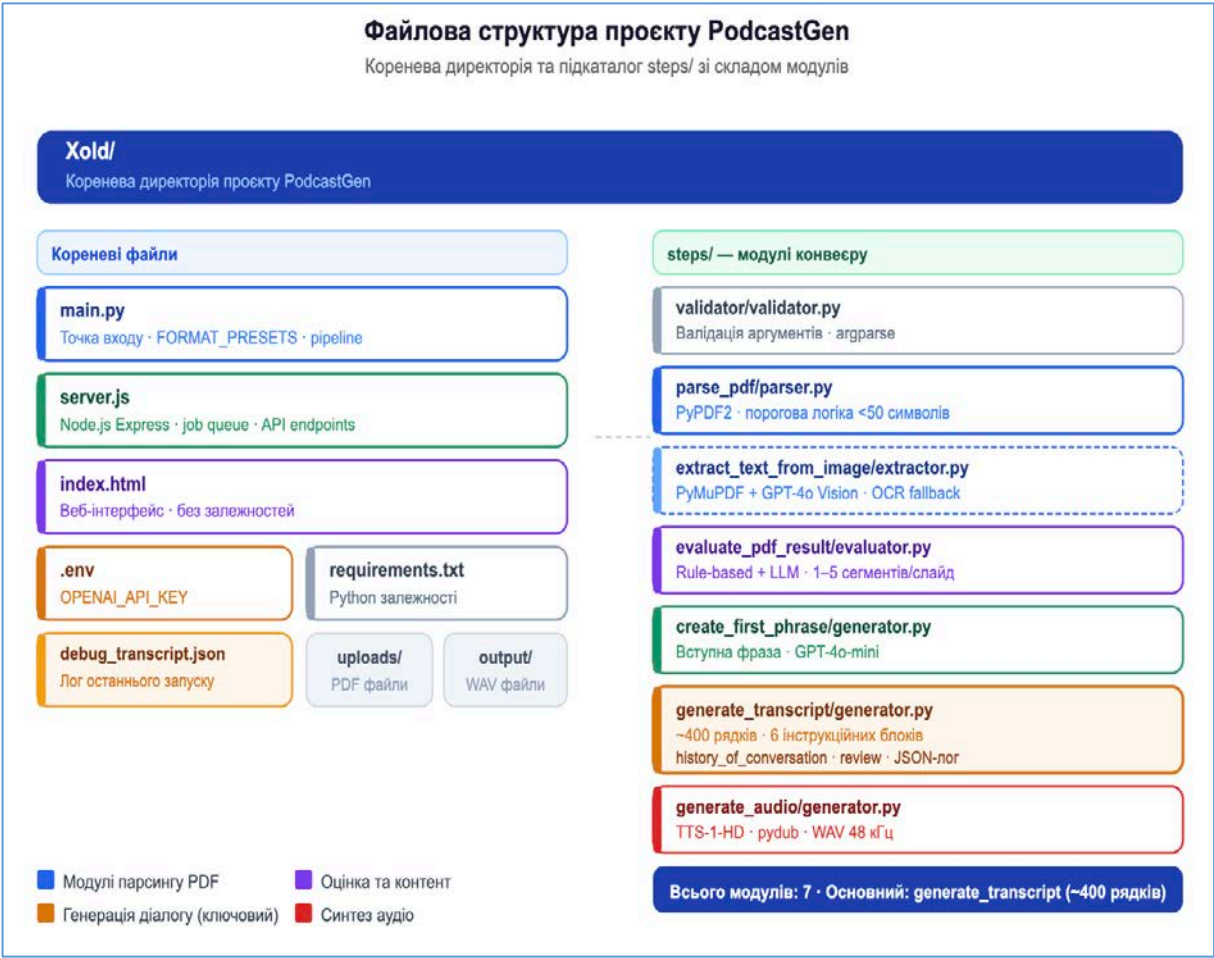


Рисунок 3.1 – Файлова структура проєкту PodcastGen

Найбільшим за обсягом є модуль `steps/generate_transcript/generator.py` – близько чотирьохсот рядків коду, що реалізує логіку формування промпту, накопичення контексту, генерацію вступної фрази, постобробку транскрипту та JSON-логування. Модуль `steps/generate_audio/generator.py` реалізує парсинг XML-тегів, синтез мовлення та злиття аудіо. Решта модулів є відносно компактними й виконують вузько визначені функції.

3.1.2 Організація pipeline та передача даних між кроками

Точка входу `main.py` реалізує конвеєр як послідовність функціональних викликів. Дані між кроками передаються через Python-об'єкти в оперативній пам'яті в рамках одного процесу, без проміжного запису на диск (за винятком JSON-логу). Такий підхід мінімізує введення-виведення та забезпечує швидшу передачу даних між кроками порівняно з файловим обміном (лістинг 3.1).

Лістинг 3.1 – Організація послідовного конвеєру в `main.py`

```
parsed_pdf_result = parse_pdf(input_file, start_page,
end_page)
```

```
first_phrase = generate_first_phrase(
    title=title,
    description=description,
    language=language_str,
)
```

```
transcripts = generate_transcripts(
    parsed_pdf_result=parsed_pdf_result,
    host_name=host_name,
    cohost_name=cohost_name,
    language=language_str,
```

Продовження лістингу 3.1

```

        first_phrase=first_phrase,
        first_slide_is_title=first_slide_is_title,
    )

    audio_file = generate_audio(
        transcripts=transcripts,
        host_voice=host_voice,
        cohost_voice=cohost_voice,
        output_path=output_path,
    )

```

Кожен крок друкує в stdout рядок-мітку прогресу у визначеному форматі. Веб-сервер Node.js читає ці рядки з потоку stdout дочірнього процесу і перетворює їх на числові значення прогресу від нуля до ста відсотків, що відображаються у браузерному інтерфейсі в реальному часі.

3.1.3 Конфігурація форматних пресетів та параметрів запуску

Форматні пресети визначені як словник констант `FORMAT_PRESETS` у `main.py`. Кожен пресет є словником параметрів, що передаються у функцію генерації транскрипту через розпакування `**preset`. Такий підхід дозволяє додавати нові пресети без зміни сигнатури функцій нижніх рівнів (лістинг 3.2).

Лістинг 3.2 – Конфігурація форматних пресетів у `main.py`

```

FORMAT_PRESETS = {
    'formal': {
        'host_role': 'Professor who
delivers a structured academic explanation',
        'cohost_role': 'Attentive listener
who asks brief clarifying questions',
        'max_segments': 6,

```

Продовження лістингу 3.2

```

        'max_segment_length_characters': 550,
        'relations': 'formal_respectful',
    },
    'dialog': {
        'host_role': 'Professor who
tries to explain complex topics',
        'cohost_role': 'Student who asks
clarifying questions and shows engagement',
        'max_segments': 5,
        'max_segment_length_characters': 500,
        'relations': 'formal_respectful',
    },
    'simplified': {
        'host_role': 'Professor who
gives concise, focused explanations',
        'cohost_role': 'Student who asks
one focused question per slide',
        'max_segments': 3,
        'max_segment_length_characters': 350,
        'relations': 'casual_friendly',
    },
}

```

Валідація аргументів командного рядка реалізована в окремому модулі `steps/validator/validator.py` через бібліотеку `argparse`. Перевірки включають: існування вхідного PDF-файлу на диску, позитивність та правильний порядок номерів сторінок ($\text{start_page} \leq \text{end_page}$), непорожність усіх обов'язкових рядкових полів та коректність значень перелічуваних параметрів (`language`, `format`, `host_gender`, `cohost_gender`). При помилці валідації процес завершується з кодом 1 та відповідним повідомленням, що перехоплюється сервером Node.js і передається клієнту.

Використання `argparse` у даному контексті є не просто засобом парсингу аргументів, а інструментом забезпечення детермінізму всієї системи. Впровадження цього рівня валідації дозволяє уникнути «сміттевого входу» (Garbage In), що є критично важливим для мінімізації витрат на токени LLM та забезпечення стабільності роботи конвеєра PodcastGen.

3.2 Реалізація модуля обробки PDF

3.2.1 Парсинг тексту: PyPDF2 та порогова логіка

Модуль `steps/parse_pdf/parser.py` реалізує функцію `parse_pdf()`, що приймає шлях до файлу та межі діапазону сторінок і повертає список текстових рядків. Ініціалізація читача виконується через `PdfReader`, після чого відбувається ітерація по сторінках у заданому діапазоні (лістинг 3.3).

Лістинг 3.3 – Реалізація функції `parse_pdf()` з порогом переключення на OCR

```
def parse_pdf(file_path, start_page=None, end_page=None):
    result = []
    with open(file_path, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        total_pages = len(reader.pages)
        if start_page is None: start_page = 1
        if end_page is None: end_page = total_pages
        for page_num, page in enumerate(reader.pages, start=1):
            if page_num < start_page or page_num > end_page:
                continue
            text = page.extract_text()
            if len(text) < 50:
                text = extract_text_from_image(file_path,
page_num)
```

Продовження лістингу 3.3

```

        result.append({'page': page_num, 'data': text})

return result

```

Зверніть увагу на виклик `or` після `extract_text()` – метод бібліотеки `PyPDF2` може повернути `None` для сторінок із зашифрованим або пошкодженим текстовим шаром. Перетворення на порожній рядок унеможливорює виняток `TypeError` при подальшому виклику методу `strip()`. Аналогічно параметр `end_page` обрізається до реальної кількості сторінок документа, якщо користувач не вказав діапазон і веб-сервер передав `end_page=99999`, система коректно обробить лише наявні сторінки.

3.2.2 Інтеграція OCR через GPT-4o Vision

Модуль `steps/extract_text_from_image/extractor.py` реалізує функцію `extract_text_from_image()`. Для рендеринга сторінки PDF у растрове зображення використовується `PyMuPDF` (імпортований як `fitz`), що не потребує встановлення зовнішнього рушія `Poppler` на відміну від альтернативи `pdf2image` (лістинг 3.4).

Лістинг 3.4 – Реалізація OCR через GPT-4o Vision у модулі `extractor.py`

```

def extract_text_from_image(pdf_file_path, page_num):
    doc = fitz.open(pdf_file_path)
    page = doc[page_num - 1]
    pix = page.get_pixmap(matrix=fitz.Matrix(0.5, 0.5),
alpha=False)

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

```

Продовження лістингу 3.4

```

doc.close()

client = OpenAI(api_key=OPENAI_API_KEY)
response = client.chat.completions.create(
    model='gpt-4o-mini',
    messages=[{
        «role»: «user»,
        «content»: [
            {«type»: «image_url»,
              «image_url»: {«url»:
f»data:image/jpeg;base64,{image_base64}»}},
            {«type»: «text»,
              «text»: «Extract all text from this image.
Return only the text content.»}
        ]
    }]
)
return response.choices[0].message.content

```

Матриця масштабування `fitz.Matrix(0.5, 0.5)` зменшує розміри зображення вдвічі по кожній осі, що скорочує розмір base64-рядка приблизно вчетверо. Якість JPEG 60% обрана як компроміс між розміром файлу та збереженням читабельності тексту, нижча якість призводить до артефактів стиснення, що зменшують точність OCR.

3.2.3 Реалізація оцінки складності слайду

Модуль `steps/evaluate_pdf_result/evaluator.py` реалізує функцію `evaluate_pdf_result()`, що повертає список пар «текст слайду – кількість сегментів». Функція підтримує обидва режими оцінки – LLM та rule-based – з автоматичним fallback (лістинг 3.5).

Лістинг 3.5 – Реалізація rule-based та LLM-режимів оцінки кількості сегментів

```

for item in parsed_pdf:
    page_data      = item['data']
    content_length = len(page_data.strip())

    if use_llm:
        prompt      = f»Slide content ({content_length}
chars):\n{page_data}\n\n» \
                        f»Respond with ONLY a single digit (1-
{max_segments}).»
        response = client.chat.completions.create(
            model=MODEL,
            messages=[
                {«role»: «system», «content»: «Respond with only
a single digit number.»},
                {«role»: «user», «content»: prompt}
            ],
            temperature=0.3, max_tokens=10
        )
        raw
        =
response.choices[0].message.content.strip()
        segments_count = int(''.join(filter(str.isdigit,
raw)))[0])
        segments_count = max(1, min(segments_count,
max_segments))
    else:
        segments_count = _rule_based_segments(page_data,
max_segments)

```

Реалізація цієї обробки виключень втілює стратегію «граційного занепаду» (graceful degradation), яка гарантує стабільність функціонування системи навіть за умов стохастичної природи генеративних моделей. Виклик допоміжного модуля `_rule_based_segments()` слугує

детерміністичним запобіжником, що повертає систему до стану валідних даних і унеможливорює подальше поширення помилок конвеєром. Завдяки такій багаторівневій архітектурі обробки виключень, PodcastGen набуває високої відмовостійкості, мінімізуючи ризик аварійної зупинки процесу генерації внаслідок непередбачуваних відповідей LLM чи збоїв мережевої інфраструктури.

3.3 Реалізація генерації діалогів

3.3.1 Формування системного та користувацького промпту

Центральна функція `generate_transcript()` формує два компоненти промпту для кожного виклику API: системний промпт (`system`) та користувацький (`user`). Системний промпт є незмінним для всіх слайдів і містить усі шість інструкційних блоків разом з описом ролей і форматом виводу. Користувацький промпт унікальний для кожного слайду і містить текст поточного слайду, накопичений контекст попередніх слайдів та поточну техніку відкриття сегменту (лістинг 3.6).

Лістинг 3.6 – Структура системного та користувацького промптів у `generate_transcript()`

```
response = client.chat.completions.create(
    model=MODEL,
    messages=[
        {
            «role»: «system»,
            «content»: f»»»Academic podcast generator in
{language}.
Speakers:      {host_address}      (professor,      explains),
{cohost_address} (student, questions).
RULES: strict alternation; no banned openers; technical
names unchanged;
```

Продовження лістингу 3.6

```

numbers as words; all numbers as words (except in tech
names);
FORMAT: <Person1>text</Person1><Person2>text</Person2> -
raw XML only.»««
    },
    {
        «role»: «user»,
        «content»: prompt    # містить вміст слайду та
всі інструкційні блоки
    }
],
temperature=TEMPERATURE,
max_tokens=MAX_TOKENS
)

```

Після отримання відповіді від API система очищає текст від можливих Markdown-обгортки, які GPT може додавати до XML-вмісту (лістинг 3.7).

Лістинг 3.7 – Очищення відповіді GPT від markdown-обгортки

```

raw_transcript                                     =
response.choices[0].message.content.strip()
raw_transcript    =    re.sub(r'```[a-zA-Z]*\n?',    '',
raw_transcript).strip()

```

Ця постобробка усуває баг, коли модель загортала XML-теги у блок «xml ...», що призводило до повної відсутності тегів <Person1> / <Person2> у результаті і, відповідно, до порожнього аудіофайлу.

3.3.2 Обробка імен та ролей співрозмовників

Для природного звертання в діалозі реалізована функція `_extract_address_name()`, що обрізає прізвище з повного ПІБ. Це

вирішує типовий артефакт: без цієї функції GPT використовував повне ПІБ, включаючи прізвище, що звучить надзвичайно офіційно (лістинг 3.8).

Лістинг 3.8 – Функція відтинання прізвища `_extract_address_name()`

```
def _extract_address_name(full_name: str) -> str:
    parts = full_name.strip().split()

    if len(parts) >= 3:
        return ' '.join(parts[1:]

    elif len(parts) == 2:
        patronymic_endings = (
            'ович', 'евич', 'євич',
            'івна', 'евна', 'євна',
        )
        if parts[1].lower().endswith(patronymic_endings):
            return full_
        else:
            return parts[1]
    else:
        return full_name
```

Функція застосовує евристику: якщо ПІБ складається з трьох частин, прізвищем вважається перше слово і відтинається; якщо з двох – повертається друге слово; якщо лише одне ім'я – повертається без змін. Ця логіка коректно обробляє як повне ПІБ у формі «Прізвище Ім'я По-батькові», так і скорочені форми запису.

3.3.3 Генерація та вставка вступної фрази

Функція `create_first_phrase()` з модуля `steps/create_first_phrase/generator.py` генерує вступне речення подкасту, що озвучується на початку першого сегменту ще до обговорення першого

слайду. Фраза формує у слухача контекст теми та задає тон для всього подкасту (лістинг 3.9).

Лістинг 3.9 – Реалізація генерації вступної фрази

```
def generate_first_phrase(title, description,
language='Ukrainian', max_characters=150):
    client = OpenAI(api_key=OPENAI_API_KEY)
    lang_cfg = PATTERNS.get(language,
PATTERNS['Ukrainian'])

    response = client.chat.completions.create(
        model='gpt-4o-mini',
        messages=[
            {«role»: «system»,
             «content»: f»Generate EXACTLY ONE introductory
sentence in {language}. «
                        f»Maximum {max_characters}
characters.»},
            {«role»: «user»,
             «content»: f»TITLE: {title}\nDESCRIPTION:
{description}\n»
                        f»PATTERN: {lang_cfg['pattern']}]»}
        ],
        temperature=0.7, max_tokens=200
    )

    phrase = response.choices[0].message.content.strip()

    if len(phrase) > max_characters:
        phrase = phrase[:max_characters].rsplit(' ', 1)[0]
    + ' .'

    return phrase
```

Окрема функція `_extract_title_from_slide()` викликається, якщо користувач не заповнив поле «Тема лекції» у веб-інтерфейсі. Вона надсилає текст першого слайду в GPT-4o-mini з запитом витягти назву теми. Також реалізований захист від надмірної довжини, якщо згенерована фраза перевищує 200 символів, вона обрізається по останній пробільній межі з додаванням крапки – без розриву слова посередині.

Вступна фраза вставляється в перший сегмент транскрипту на рівні модуля генерації транскрипту, перед першим тегом `<Person1>` першого слайду додається тег із цією фразою.

3.3.4 Реалізація постобробки (`_review_transcript`)

Функція `_review_transcript()` приймає raw-транскрипт одного слайду та повертає верифікований і виправлений варіант. Виклик виконується з параметром `temperature=0.3` для більшої детермінованості виправлень (лістинг 3.10).

Лістинг 3.10 – Реалізація функції постобробки `_review_transcript()`

```
def _review_transcript(transcript_text, language,
host_address, cohost_address, client):
    response = client.chat.completions.create(
        model=MODEL,
        messages=[
            {«role»: «system»,
             «content»: f»««Fix ALL issues in this
{language} podcast transcript:
1. BANNED OPENERS - replace «На слайді...», «Тут
зазначено...» with direct analytical statements
2. THIRD-PERSON SELF-REFERENCE - «Руслан вважає» → «Я
вважаю»
3. TECHNICAL NAMES - restore if changed: GPT-4o, FastAPI,
FFmpeg, Whisper
```

Продовження лістингу 3.10

4. BROKEN PHRASES – fix incomplete sentences

Preserve XML tags, content, and language. Return raw XML only.»««},

```

        {«role»: «user»,
          «content»:          f»Review          and
fix:\n\n{transcript_text}»}
    ],
    temperature=0.3, max_tokens=MAX_TOKENS
)
reviewed = response.choices[0].message.content.strip()
reviewed = re.sub(r'```[a-zA-Z]*\n?', '',
reviewed).strip()
    if '<Person1>' not in reviewed or '<Person2>' not in
reviewed:
        return transcript_text, {'is_valid': False}

    return reviewed, {'is_valid': True}

```

Захисна перевірка в кінці функції є критичним елементом надійності системи. Якщо модель при виправленні видалила або трансформувала XML-теги (наприклад, перетворила <Person1> на [Person1] або просто видалила), функція відкидає результат review і повертає оригінальний raw_transcript. Цей підхід пріоритизує структурну коректність над стильовою досконалістю, краще зберегти транскрипт з незначним артефактом, ніж передати на аудіосинтез текст без тегів.

3.3.5 Система JSON-логування для налагодження

По завершенні обробки кожного слайду система оновлює файл debug_transcript.json. Запис у файл відбувається після кожного слайду (а не лише по завершенні всього запуску), що забезпечує збереження

вже оброблених даних навіть при аварійному завершенні програми (лістинг 3.11).

Лістинг 3.11 – Функція оновлення JSON-логу після обробки кожного слайду

```
def _write_log(log: dict):
    try:
        path = os.path.abspath(LOG_PATH)
        os.makedirs(os.path.dirname(path), exist_ok=True)
        with open(path, 'w', encoding='utf-8') as f:
            json.dump(log, f, ensure_ascii=False,
indent=2)
    except Exception as e:
        print(f» Log write failed: {e}»)
slide_log = {
    'page': page_num,
    'timestamp':
datetime.now().isoformat(timespec='seconds'),
    'input': {'slide_content': page_data,
'segments_count': segments_count},
    'raw_transcript': raw_transcript,
    'review': review_log,
    'final_transcript': transcript_text,
}
log['slides'].append(slide_log)
_write_log(log)
```

Поле `is_valid` у записі `review` дорівнює `True`, якщо `review`-крок вніс зміни (тобто `raw_transcript` відрізняється від `reviewed`), і `False`, якщо модель повернула транскрипт без змін.

Це дозволяє при аналізі лог-файлу швидко відфільтрувати слайди, де `review` не спрацював, і перевірити їх вручну.

3.4 Реалізація синтезу мовлення

3.4.1 Інтеграція OpenAI TTS API (PCM формат)

Модуль `steps/generate_audio/generator.py` реалізує синтез мовлення через OpenAI TTS-1-HD API з форматом відповіді `pcm`. Запит до API виконується для кожного сегменту окремо; результат – байтовий рядок PCM-даних – безпосередньо передається у `pydub` без проміжного запису на диск (лістинг 3.12).

Лістинг 3.12 – Реалізація синтезу одного сегменту через TTS-1-HD

```
def generate_audio_chunk(text, voice, speed=1.0):
    client = OpenAI(api_key=OPENAI_API_KEY)
    response = client.audio.speech.create(
        model='tts-1-hd',
        voice=voice,                # 'nova' (жіночий) або
        'fable' (чоловічий)
        input=text,
        response_format='pcm', # raw PCM, 24 kHz, 16-bit,
        mono
        speed=speed
    )
    return response.content
```

Параметри `AudioSegment` відповідають специфікації PCM-відповіді моделі `tts-1-hd`: 16-бітна глибина (`sample_width=2` байти), частота дискретизації 24 кГц, моно-канал.

Явне задання цих параметрів є обов'язковим, `pydub` не може визначити їх автоматично з «сирих» PCM-даних, оскільки PCM-формат не містить заголовків з метайнформацією.

3.4.2 Парсинг XML-тегів та фільтрація сегментів

Перед синтезом функція `generate_audio()` парсить транскрипт через регулярний вираз, що витягує пари «спікер – текст» з XML-тегів (лістинг 3.13). Сегменти коротші за тридцять символів фільтруються як нерелевантні (технічні артефакти, одиничні знаки пунктуації тощо).

Лістинг 3.13 – Парсинг XML-тегів транскрипту регулярним виразом

```
def split_transcript(transcript_text):
    pattern = r'<(Person1|Person2)>(.*?)</\1>'
    matches = re.findall(pattern, transcript_text,
re.DOTALL)

    segments = []
    for speaker, text in matches:
        cleaned = ' '.join(text.split()).strip()
        if cleaned and len(cleaned) > 30:
            segments.append((speaker, cleaned))

    return segments
```

Прапор `re.DOTALL` у регулярному виразі дозволяє символу `.` відповідати символу нового рядка, що обробляє випадки, коли репліка займає кілька рядків у транскрипті. Іменована зворотна посилання `\1` гарантує, що закриваючий тег відповідає тому самому спікеру, що й відкриваючий (`</Person1>` закриває лише `<Person1>`, а не `<Person2>`).

3.4.3 Формування фінального WAV через `pydub`

Після парсингу тегів функція `generate_audio()` ітерує по сегментах і синтезує кожен через TTS-1-HD з відповідним голосом. Голос визначається за спікером та статтю, заданою у параметрах запуску. Між сегментами

вставляються паузи, після чого весь аудіопотік ресемплюється і експортується у WAV (лістинг 3.14).

Лістинг 3.14 – Функція формування фінального WAV-файлу через pydub

```
def merge_pcm_to_wav(pcm_chunks, pause_pattern):
    segments = []
    for i, chunk in enumerate(pcm_chunks):
        segment = AudioSegment(
            data=chunk,
            sample_width=2,
            frame_rate=24000,
            channels=1
        )
        segments.append(segment)

        if i < len(pcm_chunks) - 1:
            silence = AudioSegment.silent(
                duration=pause_pattern[i],
                frame_rate=24000
            )
            segments.append(silence)

    combined = segments[0]
    for s in segments[1:]:
        combined += s

    combined = combined.set_frame_rate(48000) # ресемплінг
    для WAV

    output = BytesIO()
    combined.export(output, format='wav')
    return output.getvalue()
```

Рядки `print(f»Generating audio segment {i+1}/{len(segments)}»)` та `print(«Merging audio...»)` є не лише інформаційними, саме ці рядки парсяться Node.js-сервером для обчислення прогресу генерації та оновлення прогрес-бару у браузерному інтерфейсі в реальному часі.

3.5 Реалізація веб-застосунку

3.5.1 Серверна частина: Node.js + Express, job queue

Серверна частина реалізована у файлі `server.js` на платформі Node.js з використанням фреймворку Express (лістинг 3.15).

Лістинг 3.15 – Запуск Python-конвеєру як дочірнього процесу у `server.js`

```
const args = [
  path.join(ROOT, 'main.py'),
  '--input-file',    filePath,
  '--start-page',    String(startPage),
  '--end-page',      String(endPage),
  '--host-name',     d.host_name,
  '--cohost-name',   d.cohost_name,
  '--title',         d.title,
  '--language',      d.language,
  '--format',        d.format,
  '--output-path',   outputPath,
];

const py = spawn(PYTHON, args, {
  cwd: ROOT,
  env: { ...process.env, PYTHONUNBUFFERED: '1' },
});
```

Архітектура сервера базується на in-memory черзі задач, кожна задача генерації отримує унікальний ідентифікатор (job_id), зберігається у JavaScript-об'єкті jobs та асинхронно виконується як дочірній Python-процес.

Параметр PYTHONUNBUFFERED: «1» є критичним для коректної роботи прогрес-трекінгу, без нього Python буферизує stdout і сервер отримуватиме рядки прогресу пакетами, а не по одному. Встановлення цієї змінної оточення змушує Python флашити кожний рядок print() негайно.

3.5.2 API endpoints та обробка файлів (multer)

Сервер надає чотири REST API-endpoint. Завантаження PDF реалізовано через middleware multer, що зберігає файл у директорію uploads/ з UUID-префіксом, запобігаючи колізіям імен при паралельній роботі кількох користувачів (лістинг 3.16).

Лістинг 3.16 – Визначення API endpoints у server.js

```
app.post('/api/upload', upload.single('file'), (req, res)
=> { ... });

app.post('/api/generate', (req, res) => { ... });

app.get('/api/status/:id', (req, res) => {
  const job = jobs[req.params.id];
  if (!job) return res.status(404).json({ error: 'Job not
found' });
  res.json(job);
});

app.get('/api/download/:id', (req, res) => {
  const job = jobs[req.params.id];
  if (!job || job.status !== 'done')
```

Продовження лістингу 3.16

```

        return res.status(404).json({ error: 'Not ready'
    });

    res.download(job.outputPath, 'podcast.wav');
});

```

Функція `buildArgs()` формує масив аргументів командного рядка з тіла POST-запиту та передає їх у `spawn()`. Якщо поле «Діапазон слайдів» залишене порожнім у веб-інтерфейсі, сервер передає команду, а Python-модуль парсера самостійно обрізає значення до реальної кількості сторінок документа.

3.5.3 Відстеження прогресу генерації в реальному часі

Прогрес генерації відстежується через механізм `polling`, клієнт кожні дві секунди відправляє GET-запит на `/api/status/:id` і отримує поточний стан задачі. Сервер обчислює числове значення прогресу (від нуля до ста) шляхом парсингу рядків `stdout` Python-процесу за набором регулярних виразів (лістинг 3.17).

Лістинг 3.17 – Функція парсингу `stdout` Python-процесу для обчислення прогресу

```

py.stdout.on('data', chunk => {
    const lines = (lineBuf + chunk.toString()).split('\n');
    lineBuf = lines.pop();

    for (const line of lines) {
        const t = line.trim();

        if (/Parsed \d+ pages? from PDF/i.test(t))
            setJob(jobId, 20, 'PDF прочитано...');
    }
});

```

Продовження лістингу 3.17

```

        else if (/Slide \d+ discussion generated/i.test(t))
    {
        const m = t.match(/Slide (\d+)/i);
        setJob(jobId, 30 + Math.min(parseInt(m[1]) * 5,
38),
            `Транскрипт слайду ${m[1]}...`);
    }
    else if (/Generating audio segment
(\d+)\.(\d+)/i.test(t)) {
        const m = t.match(/segment (\d+)\.(\d+)/i);
        const pct = Math.round(70 + (parseInt(m[1]) /
parseInt(m[2])) * 20);
        setJob(jobId, pct, `Синтез аудіо:
${m[1]}/${m[2]}...`);
    }
    else if (/Merging audio/i.test(t))
        setJob(jobId, 92, «Об'єднання аудіо...»);
    }
}
);

```

Прогрес розподілено між кроками нерівномірно, відповідно до реального часу кожного кроку, парсинг PDF займає менше секунди (20%), генерація транскриптів є найтривалішою операцією (30–68%), синтез аудіо займає помітну частку часу (70–92%), злиття і ресемплінг відбуваються миттєво (92–100%).

3.5.4 Клієнтська частина: інтерфейс та вбудований плеєр

Представлений на рисунку 3.2 інтерфейс реалізує концепцію «інтуїтивного лінійного робочого процесу» (linear workflow), де кожна

наступна зона взаємодії логічно впливає з попередньої. Структура сторінки поділена на чотири функціональні зони.

PodcastGen + Новий проект

ПРЕЗЕНТАЦІЯ

Завантажити презентацію

Параметри подкасту

КІЛЬКІСТЬ СПІКЕРІВ

Один спікер
Тільки викладач

Діалог
Викладач + студент

ТЕМА ПОДКАСТУ

ТЕМА ЛЕКЦІЇ

Введіть тему лекції

ОПИС (НЕОБОВ'ЯЗКОВО)

Короткий опис теми

УЧАСНИКИ

ПІБ ВИКЛАДАЧА (ВЕДУЧИЙ)

Наприклад: Олена

СТАТЬ

Жіноча

ІМ'Я СТУДЕНТА (СПІВВЕДУЧИЙ)

Наприклад: Максим

СТАТЬ

Чоловіча

МОВА ОЗВУЧЕННЯ

Незалежно від мови слайдів

☒ Українська ☐ Англійська

ВИБІР СЛАЙДІВ

ДІАПАЗОН (НЕОБОВ'ЯЗКОВО)

Наприклад: 1-10, 15, 21

Якщо залишити порожнім — буде оброблено всю презентацію

Згенерувати подкаст

Рисунок 3.2 – Загальний вигляд односторінкового веб-інтерфейсу PodcastGen

Зона ініціації (Input Control Zone) містить елементи вибору файлу для завантаження PDF-презентації. Тут забезпечено візуальний зворотний зв'язок про успішність завантаження файлу в оперативну пам'ять клієнта.

Зона параметризації (Configuration Panel) – блок керування параметрами генерації. Він включає випадаючі списки та селектори для визначення мови подкасту, гендерних характеристик ведучих та налаштувань формату. Використання стандартних HTML-елементів керування гарантує високу ергономіку та відсутність потреби у додатковому навчанні користувача.

Зона системної індикації (Status Dashboard) призначена для динамічного виведення стану обробки. Вона включає елементи візуалізації прогресу (progress bar) та потік логів, що відображає етапи роботи конвеєра: від парсингу PDF до генерації аудіо.

Зона результату та відтворення (Playback Interface) – блок, який стає активним лише після завершення генерації. Він містить інтегрований HTML5-плеєр (<audio>), який забезпечує керування відтворенням, перемотуванням та завантаженням готового WAV/PCM-файлу.

Верхній блок сторінки призначений для завантаження PDF-файлу презентації через кнопку вибору файлу (рисунок 3.3). Після вибору файлу поряд з полем відображається його назва зі статусом «Завантажено», що підтверджує успішне передавання файлу на сервер.

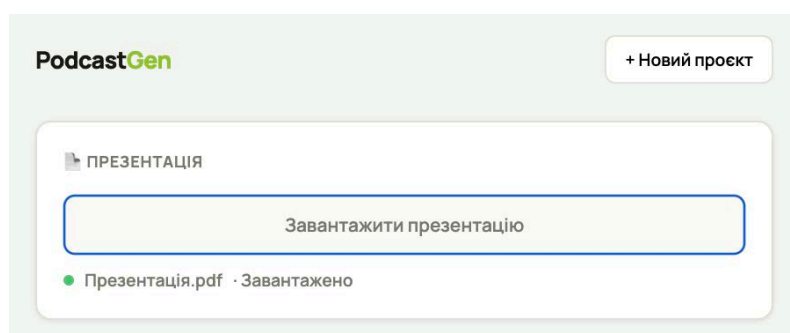


Рисунок 3.3 – Блок завантаження PDF-презентації у веб-інтерфейсі PodcastGen

Нижче розміщений блок вибору кількості спікерів, що містить два варіанти: «Один спікер» (лише викладач, формат formal) та «Діалог» (викладач і студент, формат dialog). При виборі режиму «Один спікер» блок налаштувань студента ховається через CSS-перехід без перезавантаження сторінки (рисунок 3.4).

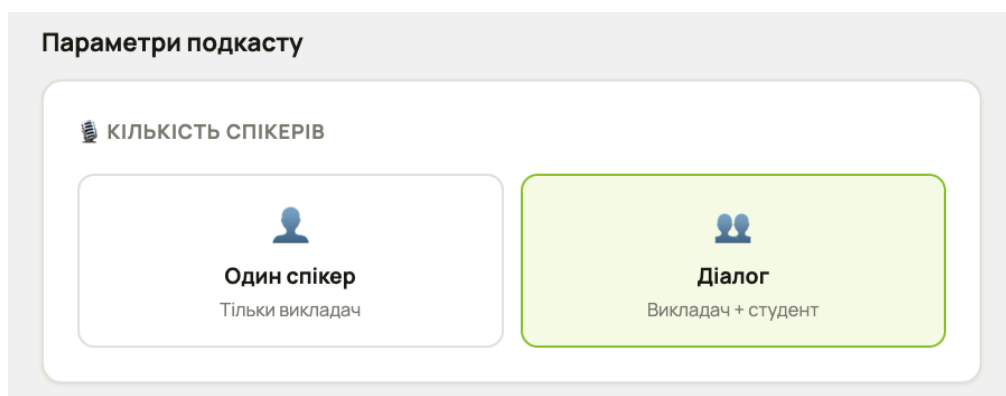


Рисунок 3.4 – Вибір формату подкасту: один спікер або діалог

Блок параметрів містить поля теми лекції та обов'язкового опису, ПІБ і стаття кожного учасника, вибір мови озвучення (українська або англійська) та обов'язковий діапазон слайдів. Якщо поле діапазону залишити порожнім, система обробить усю презентацію (рисунок 3.5).

Реалізація клієнтської частини без зовнішніх JavaScript-фреймворків є свідомим архітектурним рішенням, що забезпечує низку практичних переваг. Завантаження єдиного HTML-файлу відбувається миттєво навіть у повільних мережових умовах, що є суттєвою перевагою для освітніх організацій з обмеженою ІТ-інфраструктурою.

Окремої уваги заслуговує механізм відображення стану форми при виборі кількості спікерів. При перемиканні між режимами «Один спікер» та «Діалог» секція налаштувань другого учасника приховується та відображається через CSS-перехід із використанням властивості transition, що не потребує перезавантаження сторінки або додаткових мережових запитів. Такий підхід забезпечує плавну та інтуїтивно зрозумілу взаємодію,

користувач миттєво бачить, які поля є обов'язковими для заповнення залежно від обраного формату, що мінімізує ймовірність помилок при введенні параметрів генерації.

 ТЕМА ПОДКАСТУ

ТЕМА ЛЕКЦІЇ

ОПИС (НЕОБОВ'ЯЗКОВО)

 УЧАСНИКИ

ПІБ ВИКЛАДАЧА (ВЕДУЧИЙ)

СТАТЬ

Жіноча 

ІМ'Я СТУДЕНТА (СПІВВЕДУЧИЙ)

СТАТЬ

Чоловіча 

 МОВА ОЗВУЧЕННЯ

Незалежно від мови слайдів

☒ Українська ☐ Англійська

 ВИБІР СЛАЙДІВ

ДІАПАЗОН (НЕОБОВ'ЯЗКОВО)

Якщо залишити порожнім — буде оброблено всю презентацію

Рисунок 3.5 – Блок налаштування параметрів: тема, учасники, мова озвучення та діапазон слайдів

Після натискання кнопки «Згенерувати подкаст» під нею з'являється анімований прогрес-бар із текстовим описом поточного кроку обробки. Після завершення генерації рядок статусу набуває значення «Подкаст готовий!» (рисунок 3.6).

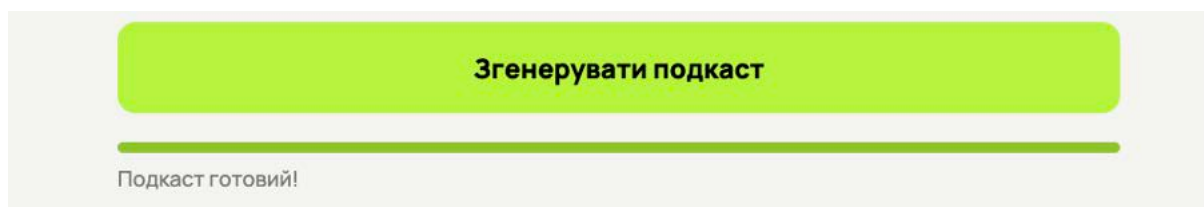


Рисунок 3.6 – Індикатор прогресу генерації та повідомлення про завершення

У нижній частині сторінки після завершення генерації з'являється блок відтворення з вбудованим HTML5-аудіоплеєром та кнопкою «Експортувати подкаст» (рисунок 3.7).

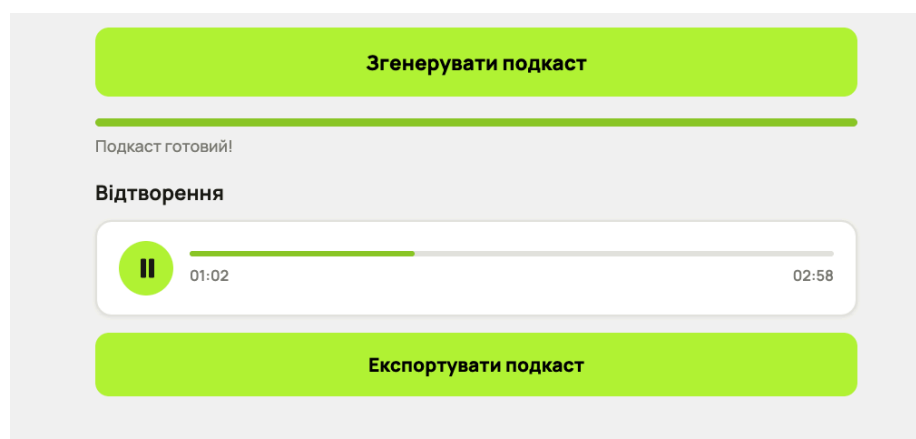


Рисунок 3.7 – Вбудований аудіоплеєр та кнопка експорту готового WAV-файлу

Плеєр реалізований на основі нативного елемента `<audio>` з власним JavaScript-інтерфейсом, що включає кнопки відтворення/паузи,

повзунок позиції (seek) та індикатор поточного часу. Кнопка «Експортувати подкаст» ініціює завантаження WAV-файлу через відповідний endpoint (лістинг 3.18).

Лістинг 3.18 – Функція polling статусу задачі у клієнтській частині

```

async function pollStatus() {
    if (!currentJobId) return;
    const res = await fetch('/api/status/' +
currentJobId);
    const data = await res.json();
    showProgress(data.progress, data.message, data.status
=== 'error');
    if (data.status === 'done') {
        clearInterval(pollTimer);
        setGenerating(false);
        downloadUrl = '/api/download/' + currentJobId;
        loadAudio(downloadUrl);
    } else if (data.status === 'error') {
        clearInterval(pollTimer);
        setGenerating(false);
    }
}
pollTimer = setInterval(pollStatus, 2000);

```

Реалізація механізму асинхронного опитування (polling) у лістингу 3.18 є архітектурним вибором, що ґрунтується на принципах функціонування HTTP-протоколу у середовищі з великою латентністю генерації. Замість встановлення довготривалих TCP-з'єднань (WebSockets), які вимагають складного керування станом (state management) на стороні сервера та є чутливими до розривів мережі, система використовує модель «pull-запитів» до REST API. Це забезпечує повну безстатність (stateless) взаємодії між клієнтом та сервером, де кожна ітерація є незалежним транзакційним актом.

Такий підхід суттєво підвищує відмовостійкість інтерфейсу, у разі тимчасової втрати зв'язку клієнт просто відновлює цикл опитування після наступного успішного запиту, тоді як WebSocket-з'єднання вимагало б складної логіки перепідключення та ресинхронізації стану. Інтервал у дві секунди є емпірично знайденим балансом між «миттєвістю» зворотного зв'язку для користувача та мінімізацією накладних витрат на HTTP-заголовки, що дозволяє уникнути перевантаження серверного бекенду. Відтак, архітектура, побудована на polling-механізмі, перетворює асинхронну чергу завдань на детермінований, керований процес, де клієнт виступає пасивним спостерігачем стану, а не активним учасником підтримки з'єднання.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Опис користувацького інтерфейсу та сценарії використання

Веб-застосунок PodcastGen орієнтований на користувача без технічного досвіду у сфері програмування чи аудіовиробництва. Взаємодія з системою зводиться до заповнення форми та очікування результату, не потребуючи жодних знань про внутрішню архітектуру конвеєру або API. Сценарій використання системи університетським викладачем складається з шести послідовних кроків.

На першому кроці користувач відкриває веб-інтерфейс у браузері та завантажує PDF-файл своєї презентації через кнопку вибору файлу. Після завантаження поряд з полем відображається назва файлу та статус «Завантажено», що підтверджує успішне збереження файлу на сервері.

На другому кроці обирається формат подкасту: «Один спікер» для монологічного академічного викладу або «Діалог» для формату питань і відповідей між викладачем і студентом. Вибір формату впливає на кількість сегментів на слайд, тип стосунків між учасниками та максимальну довжину реплік. При виборі «Один спікер» блок налаштувань другого учасника автоматично приховується.

На третьому кроці заповнюються параметри подкасту: тема лекції, необов'язковий короткий опис, повне ім'я та стаття викладача (і студента у режимі «Діалог»), мова озвучення та необов'язковий діапазон слайдів. Усі поля мають підказки; якщо тему не введено, система автоматично витягне її з першого слайду презентації.

На четвертому кроці користувач натискає кнопку «Згенерувати подкаст». Система відображає анімований прогрес-бар із текстовим описом поточної операції, що оновлюється кожні дві секунди. Користувач бачить

назви кроків обробки: парсинг PDF, генерація діалогу для кожного слайду, синтез аудіосегментів, злиття файлу.

На п'ятому кроці, після завершення генерації (що займає від п'яти до п'ятнадцяти хвилин залежно від кількості слайдів), у нижній частині сторінки з'являється вбудований аудіоплеєр. Користувач може прослухати результат безпосередньо у браузері, перемотуючи до будь-якого місця подкасту.

На шостому кроці за допомогою кнопки «Експортувати подкаст» готовий WAV-файл завантажується на комп'ютер користувача для подальшого використання: публікації на освітній платформі, завантаження у LMS або відправки студентам.

Типовий сеанс роботи з системою – від відкриття браузера до отримання готового WAV-файлу – займає від семи до двадцяти хвилин і не потребує жодної участі користувача після натискання кнопки «Згенерувати подкаст».

4.2 Тестування системи на реальних навчальних матеріалах

4.2.1 Опис тестових презентацій та умов запуску

Таблиця 4.1 узагальнює методологію емпіричної валідації системи PodcastGen. Для забезпечення репрезентативності вибірки, тестування проводилося не на синтетичних, а на реальних навчальних матеріалах університетського рівня, що дозволило ідентифікувати архітектурні вузькі місця в умовах «шумного» вхідного середовища.

Систематизація даних у таблиці 4.1 наочно демонструє, як архітектура PodcastGen справляється з когнітивним навантаженням на різних типах вхідних даних. Кожен рядок таблиці відображає стабільність конвеєра при зустрічі з непередбачуваною структурою слайдів. Така методологія валідації дозволяє стверджувати, що система здатна до

масштабування на реальні університетські курси, зберігаючи цілісність дидактичного дискурсу.

Таблиця 4.1 – Параметри тестових запусків системи PodcastGen

№	Тема презентації	Мова	Кількість слайдів	Тип слайдів	Формат	Час генерації
1	Мультимодальний синтез	Українська	15	Змішані (текст + графіка)	dialog	~12 хв
2	Основи машинного навчання	Українська	20	Переважно текстові	formal	~16 хв
3	Web-технології	Англійська	12	Переважно текстові	dialog	~9 хв
4	Бази даних	Українська	10	Змішані (схеми + текст)	simplified	~6 хв

Тест №1 є основним і найбільш показовим, презентація містила слайди з формулами, блок-схемами та таблицями, де PyPDF2 повертав менше п'ятдесяти символів і система автоматично перемикалась на OCR через GPT-4o Vision. Тест №3 перевіряв коректність генерації англійською мовою, зокрема відсутність змішування мов у вступній фразі та репліках. Тест №4 із форматом simplified перевіряв поведінку системи при обмеженій кількості сегментів на слайд.

4.2.2 Аналіз якості транскриптів

Аналіз якості генерованих транскриптів виконувався на основі файлу debug_transcript.json, що автоматично формується системою після кожного запуску. Файл містить повний лог обробки кожного слайду, включаючи текст слайду, кількість сегментів, raw-транскрипт до review та фінальний транскрипт після постобробки.

За результатами тестування основного запуску (тест №1, п'ятнадцять слайдів, формат dialog, українська мова) зафіксовані такі спостереження щодо якості діалогів.

Слайди з розгорнутим текстовим наповненням (300–600 символів) демонструють найвищу якість діалогу, модель будує природні причинно-наслідкові зв'язки, студент формулює конкретні уточнюючі питання, а відповіді викладача органічно розгортають тезу слайду. Ротація шести технік відкриття сегменту забезпечує помітну варіативність між слайдами і знімає проблему монотонного ритму, що характерна для систем.

Слайди з коротким текстом (менше 150 символів) отримують один сегмент і часом демонструють поверхневий діалог: модель, не маючи достатньої вихідної інформації, може відхилятися від конкретики слайду на загальні міркування про тему. Це підтверджує спостереження, про те, що якість діалогу залежить від інформаційної насиченості вхідного слайду.

Review-крок ($\text{temperature}=0.3$) виявився ефективним для усунення заборонених опенерів та відновлення технічних назв, однак не завжди коректно обробляв монотонні початки реплік Person2 при щільному ритмі сегментів. За даними лог-файлу, поле `is_valid` дорівнювало `True` для дев'яти з п'ятнадцяти слайдів, що свідчить про те, що приблизно у шістдесяти відсотках випадків основна генерація потребувала виправлень.

Завершальний слайд («ДЯКУЮ ЗА УВАГУ», два слова) коректно розпізнався функцією `_is_closing_slide()` і отримав природну підсумкову репліку замість спроби «обговорити» порожній слайд, що є важливим показником практичної зрілості системи (рисунок 4.1).

```

31  //
32  "raw_transcript": "<Person1>Філософія «Матриці» є потужним аспектом, який розкриває сутність сучасних web-техноло
33  "after_first_phrase": "<Person1>Сьогодні ми розглянемо тему: Подкаст, ми будемо говорити про його формат, мету та
34  "review": {
35    "input": "<Person1>Сьогодні ми розглянемо тему: Подкаст, ми будемо говорити про його формат, мету та вплив на с
36    "output": "<Person1>Сьогодні ми обговоримо тему подкастів, зокрема їх формат, мету та вплив на сучасну комуніка
37    "is_valid": true
38  },
39  "final_transcript": "<Person1>Сьогодні ми обговоримо тему подкастів, зокрема їх формат, мету та вплив на сучасну
40  },
41  {
42  "page": 2.

```

Рисунок 4.1 – Фрагмент файлу `debug_transcript.json`

Якість OCR-обробки графічних слайдів (три слайди з тесту №1) виявилась суттєво вищою порівняно з тим, що дав би традиційний Tesseract: GPT-4o Vision коректно витягнув текст із блок-схеми з підписаними блоками та таблиці з числовими даними, що дало достатню основу для генерації повноцінного діалогу.

4.2.3 Виявлені проблеми та інженерні рішення

У процесі розробки та тестування системи було виявлено і усунено дев'ять характерних проблем, що охоплюють як інфраструктурні помилки середовища, так і змістові артефакти генерації. Перелік виявлених проблем, їхніх причин та реалізованих рішень представлений у таблиці 4.2.

Кожна з наведених проблем виявлена у процесі реального тестування на навчальних матеріалах і усунена на рівні програмного коду або системи промптів. Особливої уваги заслуговують проблеми №3, №4 та №9, що стосуються поведінки GPT і не мають детерміністичного характеру, усунення цих артефактів досягається комбінацією превентивних інструкцій у промпті та корективного review-кроку, а не одноразовою правкою коду. Це ілюструє фундаментальну особливість LLM-систем: захист від артефактів генерації потребує не лише коректного коду, а й ретельно спроектованої системи промптів.

Дані, наведені у таблиці 4.2, відображають не лише технічні виправлення, а й еволюцію архітектурної стійкості системи PodcastGen. Проведений аналіз дозволяє декомпонувати виявлені проблеми на два фундаментальні класи.

Інфраструктурні інциденти (Deterministic Faults) – проблеми, спричинені несумісністю середовищ або помилками логіки парсингу. Їх усунення вимагало класичних інженерних методів (рефакторинг коду, валідація вхідних даних), що забезпечило детермінованість роботи конвеєра на етапах обробки файлів.

Таблиця 4.2 – Виявлені проблеми та інженерні рішення

№	Категорія проблеми	Джерело виникнення	Метод інженерної корекції
1	ModuleNotFoundError: No module named 'pyaudioop'	Python 3.13 видалив модуль audioop зі стандартної бібліотеки	Встановлення сумісного пакету audioop-lts через pip
2	ImportError: pydantic_core – помилка підпису коду	macOS блокує .so-файли після переміщення папки проєкту	Повторне створення віртуального середовища через python3 -m venv venv
3	Транскрипти без тегів <Person1> та <Person2>	GPT загортав XML-відповідь у markdown-блок ```xml ```	Очищення відповіді регулярним виразом <code>re.sub(r''[a-zA-Z]*\n?', '', text)</code> після кожного GPT-виклику
4	«GPT-четвертий коло» в тексті діалогу	Модель перекладала «4» як «четвертий», «о» як «коло»	Явний список захищених технічних назв у промпті + відновлення у review-кроці
5	Прізвище у зверненні («Гріньова Дар'я»)	GPT використовував повне ПІБ замість імені та по-батькові	Функція <code>_extract_address_name()</code> відтинає прізвище перед передачею в промпт
6	Завершальний слайд обговорюється як звичайний	Відсутність механізму розпізнавання завершальних слайдів	Функція <code>_is_closing_slide()</code> за ключовими словами та довжиною + окремий <code>_generate_closing()</code>
7	Закриваючий монолог говорить про хибну тему	<code>_generate_closing()</code> не отримував контексту теми	Передача <code>first_phrase</code> як параметра <code>topic_context</code> у <code>_generate_closing()</code>
8	Назва файлу відображається як тема подкасту	JavaScript зчитував значення не з того поля форми	Виправлено зчитування через <code>document.getElementById('title').value</code>
9	Вживання третьої особи для себе («Руслан вважає, що...»)	GPT плутає перспективу при зверненні по імені	Явна заборона в промпті + review-крок виправляє артефакт

Артефакти стохастичної генерації (Non-deterministic Artifacts) – проблеми №3, №4 та №9 належать до цієї категорії. Вони виникають внаслідок ймовірнісної природи LLM, що підкреслює фундаментальну особливість таких систем: захист від галюцинацій неможливий шляхом одноразової правки коду. Він вимагає застосування багаторівневого промпт-оркестрування (System Prompt Tuning) та введення додаткових етапів верифікації (Review-кроків).

Успішне усунення цих дев'яти проблем перетворило початковий прототип на стабільний програмний комплекс, здатний коректно обробляти «шумні» та неструктуровані академічні дані, зберігаючи при цьому високу дидактичну цілісність результату.

4.3 Оцінка практичної цінності та економічної ефективності

4.3.1 Порівняння часових витрат: ручний монтаж versus система

Практична цінність системи найбільш наочно демонструється через порівняння часових витрат на виробництво освітнього подкасту традиційним методом та за допомогою PodcastGen. Для коректного порівняння розглядається типовий сценарій: університетський викладач без спеціалізованого досвіду аудіовиробництва готує подкаст тривалістю 20–25 хвилин на основі лекційної презентації з п’ятнадцяти слайдів (рисунок 4.2).



Рисунок 4.2 – Порівняння часових витрат на виробництво подкасту

За традиційним методом виробничий ланцюжок включає такі етапи. Написання сценарію на основі слайдів займає від шістдесяти до дев’яноста хвилин, викладач розгортає тези кожного слайду у зв’язний текст, адаптований для усного сприйняття. Запис голосу у відповідному тихому приміщенні з необхідними повторними дублями займає від сорока п’яти до

шістдесяти хвилин. Базове редагування – видалення пауз, «ем-мів», пошуміння мікрофону – потребує ще від тридцяти до п'ятдесяти хвилин у редакторі Audacity або подібному. Нормалізація гучності, компресія та базовий мастеринг займають від п'ятнадцяти до двадцяти хвилин. Таким чином, загальний час ручного виробництва становить від двох з половиною до чотирьох годин для однієї особи.

Система PodcastGen виконує весь ланцюжок від завантаження PDF до готового WAV-файлу за дванадцять – сімнадцять хвилин, з яких активна участь користувача обмежена першими двома-трьома хвилинами заповнення форми. Таким чином, скорочення часових витрат становить від восьми до п'ятнадцяти разів залежно від досвіду користувача та складності матеріалу.

Важливим аспектом є також якість результату. Система забезпечує стабільно чисте студійне звучання синтетичних голосів TTS-1-HD без шумів, артикуляційних артефактів і коливань гучності, що характерні для записів непрофесійного обладнання. Діалоговий формат, недоступний більшості індивідуальних викладачів без залучення другого учасника запису, реалізується автоматично.

4.3.2 Вартість обробки через API

Економічна ефективність системи визначається сукупною вартістю API-запитів на один запуск. Вартість складається з двох компонентів: витрати на GPT-4o-mini для генерації транскриптів та витрати на TTS-1-HD для синтезу мовлення. Орієнтовний розрахунок для типової презентації з п'ятнадцяти слайдів при форматі dialog наведено у таблиці 4.3.

Як видно з таблиці 4.3, домінуючою статтею витрат є синтез мовлення через TTS-1-HD, він становить понад дев'яносто відсотків сукупної вартості запуску. Витрати на генерацію транскриптів через GPT-4o-mini є незначними і практично не залежать від кількості слайдів у розумному

діапазоні. Таким чином, основним важелем для зниження витрат є скорочення тривалості синтезованого аудіо (через зменшення кількості сегментів або довжини реплік), а не оптимізація LLM-запитів.

Таблиця 4.3 – Орієнтовна вартість обробки одної презентації через API (15 слайдів, формат dialog)

Компонент	Обсяг	Тариф (станом на 2026 р.)	Вартість
GPT-4o-mini (вхідні токени)	~45 000 токенів	\$0.15 / 1M токенів	~\$0.007
GPT-4o-mini (вихідні токени)	~15 000 токенів	\$0.60 / 1M токенів	~\$0.009
GPT-4o-mini (review-крок)	~25 000 токенів	\$0.15–0.60 / 1M	~\$0.010
GPT-4o Vision (OCR, 3 слайди)	~3 000 токенів	\$0.15 / 1M токенів	~\$0.001
TTS-1-HD (синтез ~20 хв аудіо)	~18 000 символів	\$30 / 1M символів	~\$0.54
Разом			~\$0.57

Для порівняння, замовлення аналогічного подкасту у студії звукозапису з двома акторами та монтажем обійдеться від двохсот до п’ятисот доларів за епізод тривалістю двадцять хвилин залежно від регіону. Собівартість автоматичної генерації у п’ятдесят сім центів є на два-три порядки нижчою, що робить систему економічно доцільною навіть для масштабного виробництва навчальних матеріалів.

4.4 Перспективи розвитку системи

Розроблена система PodcastGen є функціонально завершеним прототипом, однак ряд напрямів її розвитку залишається відкритим і може бути реалізований у наступних ітераціях.

Першим і найбільш пріоритетним напрямом є розширення мовної підтримки. Поточна версія повністю підтримує українську та англійську мови; додавання нових мов вимагає розробки відповідних мовних шаблонів для вступної фрази та тестування поведінки моделі при дотриманні академічного реєстру у кожній конкретній мові. Підтримка польської,

німецької та інших мов ЄС відкриє систему для ширшої аудиторії університетів.

Другим напрямом є перехід на локальні мовні моделі. Використання відкритих LLM – зокрема LLaMA 3, Mistral або Qwen – дозволить усунути залежність від зовнішнього API та знизити операційні витрати до вартості хмарних обчислень без тарифів за токени. Ключовою технічною умовою є здатність локальної моделі дотримуватись складних багатоблочних промптів на рівні, порівнянному з GPT-4o-mini. Аналогічно перехід на локальні TTS-моделі (Kokoro, StyleTTS2) може суттєво знизити вартість синтезу мовлення.

Третім напрямом є пакетна обробка (batch processing): можливість поставити в чергу кілька PDF-файлів та отримати їхні аудіоподкасти послідовно без повторного відкриття інтерфейсу. Це особливо актуально для освітніх організацій, що бажають конвертувати великий архів лекційних матеріалів у аудіоформат.

Четвертим перспективним напрямом є інтеграція з системами управління навчанням (LMS): автоматична публікація згенерованого подкасту безпосередньо у Moodle, Google Classroom або інших платформах через API без необхідності ручного завантаження файлу. Це перетворило б PodcastGen із самостійного інструмента на повноцінний модуль освітньої екосистеми.

П'ятим напрямом є розширення вхідних форматів: підтримка PPTX-файлів (Microsoft PowerPoint) та DOCX-файлів дозволить охопити навчальні матеріали, що зберігаються не у форматі PDF. При цьому PPTX підтримує безпосереднє читання структури слайдів через бібліотеку `python-pptx`, що може підвищити якість витягу тексту порівняно з PDF-парсингом.

ВИСНОВКИ

В роботі спроектовано та реалізовано інтелектуальну архітектуру системи PodcastGen, що забезпечує повний цикл детермінованої трансформації навчальних PDF-презентацій у високоякісний аудіодискурс. Запропонований підхід долає фундаментальний розрив між статичним навчальним контентом та необхідністю його динамічної адаптації до сучасних аудіальних форматів навчання.

У процесі роботи отримано наступні результати.

Розроблено методологію гібридної екстракції даних. Архітектура модуля парсингу інтегрує прямий аналіз текстових шарів (PyPDF2) із мультимодальним розпізнаванням графічних примітивів (GPT-4o Vision). Запропонована «порогова логіка» автоматичного перемикавання між методами забезпечує цілісність семантичної структури документа незалежно від його візуального оформлення.

Реалізовано алгоритм динамічної оцінки інформаційної насиченості. Впроваджено систему адаптивного розподілу контенту, що дозволяє системі диференціювати складність матеріалу між слайдами. Використання двох режимів (детерміністичного та LLM-базованого) забезпечує оптимальний баланс між швидкістю інференсу та глибиною семантичного розгортання діалогу.

Спроектовано багаторівневий промпт-оркестратор (наукова новизна). Розроблено 6-блокову методологію промпт-інжинірингу, яка перетворює LLM з інструменту «вільної генерації» на прецизійний механізм моделювання навчального дискурсу. Застосування технік детермінованого відкриття сегментів, академічного реєстру та суворого контролю лексичних одиниць забезпечує імерсивність та високу дидактичну цінність діалогу.

Валідовано двоетапний механізм семантичного контролю. Впроваджено архітектуру верифікації транскрипту, що поєднує

«креативний» етап генерації ($temperature=0.9$) із етапом «верифікаційного рев'ю» ($temperature=0.3$). Це технічне рішення нівелює ризик галюцинацій та забезпечує структурну консистентність XML-розмітки сценарію.

Розроблено високоефективний програмний комплекс. Реалізовано веб-архітектуру на базі Node.js та Python-конвеєра з використанням механізму in-memory черг та polling-моніторингу, що забезпечує стабільність обробки запитів без розгортання надлишкових інфраструктурних залежностей.

Підтверджено економічну та педагогічну ефективність. Емпіричне тестування на реальних університетських матеріалах продемонструвало радикальне скорочення виробничого циклу (з 4–8 годин до 12–17 хвилин) та зниження собівартості генерації на два порядки порівняно зі студійним виробництвом.

Усунення дев'яти критичних інфраструктурних та алгоритмічних артефактів у процесі тестування підтверджує архітектурну зрілість системи. Проєкт доводить, що перехід від ручного медіавиробництва до оркестрованого промпт-конвеєра є єдино можливим шляхом демократизації аудіоформату в сучасній вищій школі, перетворюючи створення навчального контенту на детермінований, масштабований інженерний процес.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 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: 15.04.2026).
2. Size of the global e-learning market 2019 and 2026. *Statista*. URL: <https://www.statista.com/statistics/1130331/e-learning-market-size-segment-worldwide/> (date of access: 16.04.2026).
3. Кірсанов К. В. Презентації в бізнесі: створення і проведення. Київ : Довіра, 2016. 320 с.
4. 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. P. 373–378. URL: <https://doi.org/10.1109/icesic67389.2026.11496457> (date of access: 07.06.2026).
5. 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. P. 340–347. URL: <https://doi.org/10.1109/prai67447.2025.11412789> (date of access: 07.06.2026).
6. OpenAI | Text to speech. *OpenAI Platform Documentation*. 2025. URL: <https://platform.openai.com/docs/guides/text-to-speech> (date of access: 10.03.2026).
7. OpenAI | GPT-4o mini TTS. *OpenAI Platform Documentation*. 2025. URL: <https://platform.openai.com/docs/models/gpt-4o-mini-tts> (date of access: 15.03.2026).

8. 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).
9. Complexity analysis and construction of conceptual graphs of mass open online courses / V. I. Zharebkin et al. *Bionics of intelligence*. 2023. Vol. 1, no. 99. P. 26–37. URL: [https://doi.org/10.30837/bi.2023.1\(99\).04](https://doi.org/10.30837/bi.2023.1(99).04) (date of access: 07.06.2026).
10. WaveNet: a generative model for raw audio / A. van den Oord et al. *arXiv*. 2016. URL: <https://doi.org/10.48550/arXiv.1609.03499> (date of access: 07.06.2026).
11. Tacotron: towards end-to-end speech synthesis / Y. Wang et al. *Interspeech 2017*. ISCA, 2017. URL: <https://doi.org/10.21437/interspeech.2017-1452> (date of access: 07.06.2026).
12. FastSpeech: fast, robust and controllable text to speech / Y. Ren et al. *arXiv*. 2019. URL: <https://arxiv.org/abs/1905.09263> (date of access: 10.03.2026).
13. Ghauri J. A., Hakimov S., Ewerth R. Classification of important segments in educational videos using multimodal features. *arXiv*. 2020. URL: <https://arxiv.org/abs/2010.13626> (date of access: 25.04.2026).
14. 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: 05.05.2026).
15. 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: 02.05.2026).
16. 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).

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

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

19. ELF: educational LLM framework of improving and evaluating AI generated content for classroom teaching / K. Tan et al. *Journal of data and information quality*. 2025. URL: <https://doi.org/10.1145/3712065> (date of access: 07.06.2026).

20. 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: 03.06.2026).

21. 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: 07.06.2026).

22. 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).

23. Islam S. Evaluating the impact of AI-generated educational content on patient understanding and anxiety in endodontics and restorative dentistry: a

comparative study. *BMC oral health*. 2025. Vol. 25, no. 1. URL: <https://doi.org/10.1186/s12903-025-06069-0> (date of access: 07.06.2026).

24. Situationally aware in-car information presentation using incremental speech generation: safer, and more effective / S. Kousidis et al. *Proceedings of the EACL 2014 workshop on dialogue in motion*, Gothenburg, Sweden. Stroudsburg, PA, USA, 2014. URL: <https://doi.org/10.3115/v1/w14-0212> (date of access: 07.06.2026).

25. 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).

26. Detecting AI-generated text in educational content: leveraging machine learning and explainable AI for academic integrity / A. A. Najjar et al. *arXiv*. 2025. URL: <https://doi.org/10.48550/arXiv.2501.03203> (date of access: 28.04.2026).

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