Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Комп'ютерного моделювання та інтелектуальних технологій
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження підходів до інтелектуальної обробки та автоматизованої генерації навчального матеріалу в системі керування навчанням із використанням методів машинного навчання та великих мовних моделей
(тема)

Виконав:

студент 2 курсу, групи СПРм-24-1

Цибань Д. С.
(прізвище, ініціали)

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

Тип програми Освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне проєктування
(повна назва освітньої програми)

Керівник доц. Чорна О. С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ _____
(підпис) (прізвище, ініціали)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Комп'ютерного моделювання та інтелектуальних технологій

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки

Тип програми Освітньо-наукова

Освітня програма Системне проєктування

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«___» _____ 2026 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Цибаню Дмитру Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема Дослідження підходів до інтелектуальної обробки та автоматизованої генерації навчального матеріалу в системі керування навчанням із використанням методів машинного навчання та великих мовних моделей

2. Термін подання студентом роботи до екзаменаційної комісії 15.05.2026

3. Вихідні дані до роботи Розробити API-модуль для інтеграції LLM у LMS з функціями обробки навчальних матеріалів, генерації конспектів і тестів, відповідей на запитання та аналізу студентських робіт.

4. Перелік питань, що потрібно опрацювати в роботі: 4.1 Вступ 4.2 Аналіз стану проблеми та теоретичних підходів до інтелектуальної генерації навчального контенту 4.2.1 Огляд літератури та сучасних досліджень 4.2.2 Характеристика предметної області 4.2.3 Визначення сфери застосування результатів 4.2.4 Постановка задачі дослідження 4.2.5 Методологія проведення дослідження 4.3 Проєктування архітектури API-модуля для інтеграції LLM у системи керування навчанням 4.3.1 Аналіз існуючих систем управління навчанням 4.3.1.1 Google Classroom 4.3.1.2 Moodle 4.3.1.3 Coursera 4.3.1.4 Udemy 4.3.2 Огляд технологічного стеку 4.3.2.1 Мови програмування, їхні бібліотеки та фреймворки 4.3.2.2 Системи

зберігання та безпеки даних 4.3.2.3 Інфраструктура та розгортання проєкту 4.3.2.4 Сторонні сервіси та API 4.3.2.5 Вибір оптимальних інструментів розробки 4.3.3 Методи машинного навчання та їх реалізація, метрики оцінки 4.3.4 Проєктування архітектури API-модуля 4.3.5 Вимоги до функціоналу ML-моделей і LLM у проєкті 4.4 Практична реалізація та дослідження ефективності розробленого API-модуля 4.4.1 Реалізація архітектури API та можливості масштабування 4.4.2 Порівняльний експеримент ML.NET проти Python 4.4.3 Тестування та оптимізація роботи модуля 4.4.4 Економічне обґрунтування впровадження інтелектуального модуля 4.5 Висновки 4.6 Перелік джерел посилання.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій: API Gateway автентифікується від імені клієнта, Візуальний огляд потоку MLOps та циклу зворотного зв'язку моделі, Архітектура та потік виконання NimbusML, Діаграма структури коду програми та ітераційного процесу розробки моделі, Загальна схема навченої моделі, що використовується у виробництві, Методи наукових та експериментальних досліджень, Інтерфейс сторінки зібраної аналітики успішності учасників курсу в Google Classroom, Схематична архітектура проєкту

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Аналіз стану проблеми та теоретичних підходів до інтелектуальної генерації навчального контенту	Доц. каф. КМІТ Чорна О. С.		12.05.2026
Проєктування архітектури API-модуля для інтеграції LLM у системи керування навчанням	Доц. каф. КМІТ Чорна О. С.		12.05.2026
Практична реалізація та дослідження ефективності розробленого API-модуля	Доц. каф. КМІТ Чорна О. С.		12.05.2026

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів	Примітка
1	Отримання завдання на кваліфікаційну роботу	22.01.26 – 24.01.26	Вик
2	Аналіз теми, літератури та сучасних досліджень	25.01.26 – 05.02.26	Вик
3	Аналіз LMS та сфери застосування результатів	06.02.26 – 15.02.26	Вик
4	Постановка задачі та визначення методології дослідження	16.02.26 – 22.02.26	Вик
5	Огляд технологічного стеку та вибір інструментів розробки	23.02.26 – 05.03.26	Вик
6	Проєктування архітектури API-модуля	06.03.26 – 15.03.26	Вик
7	Реалізація основних функцій API, тестування, оптимізація	11.04.26 – 20.04.26	Вик
8	Економічне обґрунтування та оформлення пояснювальної записки	26.04.26 – 30.04.26	Вик

Дата видачі завдання 22 січня 2026 р.

Студент Цибань Дмитро Сергійович

(підпис)

Керівник роботи доц. каф. КМІТ Чорна О. С.

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Звіт кваліфікаційної роботи містить: 100 сторінок, 15 рисунків, 8 таблиць, 32 джерела, 2 додатки.

СИСТЕМА КЕРУВАННЯ НАВЧАННЯМ (LMS), ВЕЛИКІ МОВНІ МОДЕЛІ (LLM), МАШИННЕ НАВЧАННЯ, АВТОМАТИЗОВАНА ГЕНЕРАЦІЯ КОНТЕНТУ, ІНТЕЛЕКТУАЛЬНА ОБРОБКА ДАНИХ, НАТУРАЛЬНО-МОВНА ОБРОБКА (NLP), АДАПТИВНІ ОСВІТНІ ТЕХНОЛОГІЇ.

Об’єктом досліджень є процес інтелектуальної обробки та автоматизованої генерації навчального контенту в системах керування навчанням (LMS).

Предметом досліджень є методи машинного навчання та архітектури LLM для структурування, адаптації та генерації матеріалів в освітніх платформах.

Мета досліджень: підвищення ефективності освітнього процесу в LMS через автоматизацію створення персоналізованого контенту на основі LLM.

До методів дослідження у цій роботі відносяться: аналіз архітектур Transformers, методи NLP (семантичний аналіз), оперативне проектування (Prompt Engineering), RAG-технології та статистична оцінка якості тексту.

Дослідження розвиває методи інтеграції ШІ та адаптивного навчання в LMS.

Наукова новизна полягає в удосконаленні семантичного структурування через RAG та моделюванні генерації завдань різної складності. Практичним результатом є прототип модуля для автоматизації планів, конспектів і тестів. Впровадження у ЗВО та корпоративне навчання знизить навантаження на фахівців, а подальший розвиток дозволить створити мультимодальних асистентів із мінімальним рівнем «галюцинацій».

Галузь застосування – LMS-платформи, інструменти автоматизації педагогічного дизайну, системи дистанційного та корпоративного навчання.

ABSTRACT

The qualification work report contains: 83 pages, 15 figures, 8 tables, 32 sources, 1 appendix.

LEARNING MANAGEMENT SYSTEM (LMS), LARGE LANGUAGE MODELS (LLM), MACHINE LEARNING, AUTOMATED CONTENT GENERATION, INTELLIGENT DATA PROCESSING, NATURAL LANGUAGE PROCESSING (NLP), ADAPTIVE EDUCATIONAL TECHNOLOGIES.

The object of research is the process of intelligent processing and automated generation of educational content in learning management systems (LMS).

The subject of research is machine learning methods and LLM architectures for structuring, adapting, and generating materials in educational platforms.

The goal of research is to increase the efficiency of the educational process in LMS through the automation of personalized content creation based on LLMs.

The research methods in this work include: analysis of Transformer architectures, NLP methods (semantic analysis), prompt engineering, RAG technologies, and statistical evaluation of text quality.

The research develops methods for integrating AI and Adaptive Learning into LMS.

The scientific novelty lies in the improvement of semantic structuring via RAG and the modeling of task generation considering varying levels of complexity. The practical result is a prototype module for automating plans, summaries, and tests. Implementation in higher education institutions and corporate training will reduce the workload on specialists, while further development will allow for the creation of multimodal assistants with a minimal level of "hallucinations."

The field of application includes LMS platforms, instructional design automation tools, distance learning systems, and corporate training systems.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ЗВО – Заклад вищої освіти

ВНЗ – Вищий навчальний заклад

ПЗ – Програмне забезпечення

LMS – Learning management system

LLM – Large Language Model

API – Програмний інтерфейс додатків

NLP (Natural Language Processing) – Обробка природної мови

RAG – Retrieval-Augmented Generation

MOOC – Масові відкриті онлайн-курси (Coursera, Udemy, EdX тощо)

.NET – кросплатформена екосистема для розробки ПЗ від Microsoft

ML (Machine Learning) – Машинне навчання

ML.NET – відкритий фреймворк від Microsoft, призначений для інтеграції сценаріїв машинного навчання

AI – Artificial Intelligence

ШІ – Штучний інтелект

Промпт – Сформульований живою мовою запит до штучного інтелекту

БД – База даних

XSS (Cross–Site Scripting) – Міжсайтовий скриптинг; вразливість, що дозволяє зловмисникам впроваджувати шкідливий код у веб–сторінки

CSRF (Cross-Site Request Forgery) – Міжсайтова підробка запиту; вид кібератаки, що змушує браузер користувача виконувати несанкціоновані дії на веб–ресурсі, де він пройшов автентифікацію.

CI/CD – Автоматизація інтеграції та розгортання коду

AWS (Amazon Web Services) / Azure – хмарні інфраструктури для обчислень

GPU – графічний процесор для прискорення обчислень ШІ

HTTP / gRPC – протоколи передачі даних

ВСТУП

Сучасний етап цифровізації освіти зумовив стрімке зростання обсягів електронних ресурсів та інтенсифікацію дистанційного навчання. Системи керування навчанням (LMS) є базовим інструментом сучасної освіти, проте вони переважно базуються на статичному відображенні контенту. Це створює значне когнітивне навантаження на викладачів, оскільки структурування лекцій, розробка тестів та адаптація матеріалів під потреби студентів досі виконуються вручну.

Актуальність дослідження зумовлена технологічним розривом між можливостями штучного інтелекту та їх реальним впровадженням в освітню практику. Поява великих мовних моделей (LLM) дозволяє автоматизувати рутинні операції педагогічної роботи. Методи машинного навчання дають змогу здійснювати семантичний аналіз бази знань курсу, виокремлювати ключові концепції та формувати адаптивні освітні траєкторії.

Особливої ваги набуває перехід до модульних рішень та API. Такий підхід дозволяє інтегрувати аналітичні можливості сучасних моделей штучного інтелекту в існуючу інфраструктуру ЗВО без повної перебудови програмного забезпечення. Розробка методів автоматизованої генерації матеріалів є необхідною для підвищення якості дистанційної освіти, персоналізації навчання та оптимізації роботи викладача.

Метою роботи є підвищення ефективності освітнього процесу в LMS шляхом розробки методів інтелектуальної обробки даних та генерації персоналізованого контенту на основі LLM. Для цього вирішено низку завдань: аналіз архітектур LLM та методів RAG, дослідження семантичного структурування даних, проектування архітектури програмного модуля у вигляді API та розробка алгоритмів генерації конспектів і тестів з оцінкою їхньої релевантності.

Об’єктом дослідження виступає процес інтелектуальної обробки текстової інформації та формування навчальних матеріалів у LMS. Предметом – методи машинного навчання, алгоритми обробки природної мови та архітектурні рішення

на базі LLM, що використовуються для структурування та адаптації освітнього контенту.

Для вирішення завдань застосовано системний аналіз, методи обробки природної мови (NLP), машинного навчання для класифікації знань, а також методи оперативного проектування та статистичного аналізу. Це забезпечило обґрунтованість рішень та їх відповідність вимогам сучасної педагогічної практики й системного проектування.

Наукова новизна полягає в удосконаленні підходів до динамічного структурування даних шляхом інтеграції технологій RAG та LLM, що забезпечило високу контекстуальну точність контенту. Дістали розвитку методи формування перевірочних завдань через семантичні графи знань, що дозволяє контролювати когнітивну складність матеріалу. Запропоновано модульну архітектуру обробки даних для адаптивної генерації ресурсів у режимі реального часу.

Практичне значення визначається розробкою універсального API-модуля для автоматизації створення конспектів та тестових баз. Рішення інтегрується в будь-які діючі LMS, що забезпечує масштабованість технології та зниження трудомісткості роботи викладачів. Результати можуть бути використані ЗВО та корпораціями для створення персоналізованих освітніх середовищ.

1 АНАЛІЗ СТАНУ ПРОБЛЕМИ ТА ТЕОРЕТИЧНИХ ПІДХОДІВ ДО ІНТЕЛЕКТУАЛЬНОЇ ГЕНЕРАЦІЇ НАВЧАЛЬНОГО КОНТЕНТУ

1.1 Огляд літератури та сучасних досліджень

«C# Yellow Book» Роба Майлза [1] є визнаним посібником із розробки на базі стеку .NET, що поєднує фундаментальний технічний аналіз із доступним викладом складних концепцій. Автор детально розкриває принципи об’єктно-орієнтованого програмування, архітектуру типів даних та механізми управління пам’яттю, фокусуючись на створенні відмовостійкого програмного забезпечення. Книга виходить за межі простого вивчення синтаксису, приділяючи значну увагу професійним стандартам написання коду та методам ефективної обробки виняткових ситуацій.

У контексті розробки інтелектуального API–модуля ця праця слугує надійним методологічним підґрунтям для проєктування серверної логіки, написаною мовою C#, яка є основою платформи .NET. Використання викладених у книзі підходів дозволяє реалізувати стабільну та масштабовану архітектуру сервісу, що є критично важливим для обробки великих текстових масивів.

«The API Gateway Handbook» Томаса Байєра та Тобіаса Поллі [2] – це глибоке технічне дослідження архітектурних патернів, що забезпечують ефективне управління сучасними прикладними інтерфейсами. Автори детально аналізують роль шлюзів як центральної ланки в розподілених системах, приділяючи особливу увагу питанням безпеки, маршрутизації трафіку та оптимізації продуктивності. Книга пропонує практичні рішення для реалізації автентифікації, обмеження частоти запитів і кешування, що дозволяє створювати надійну інфраструктуру, здатну витримувати значні навантаження в мікросервісних середовищах.

На діаграмі, що на рисунку 1.1, показано, як шлюз розташований між клієнтом та серверною частиною, прозоро виконуючи автентифікацію та захищаючи клієнта від процедур входу, специфічних для серверної частини.

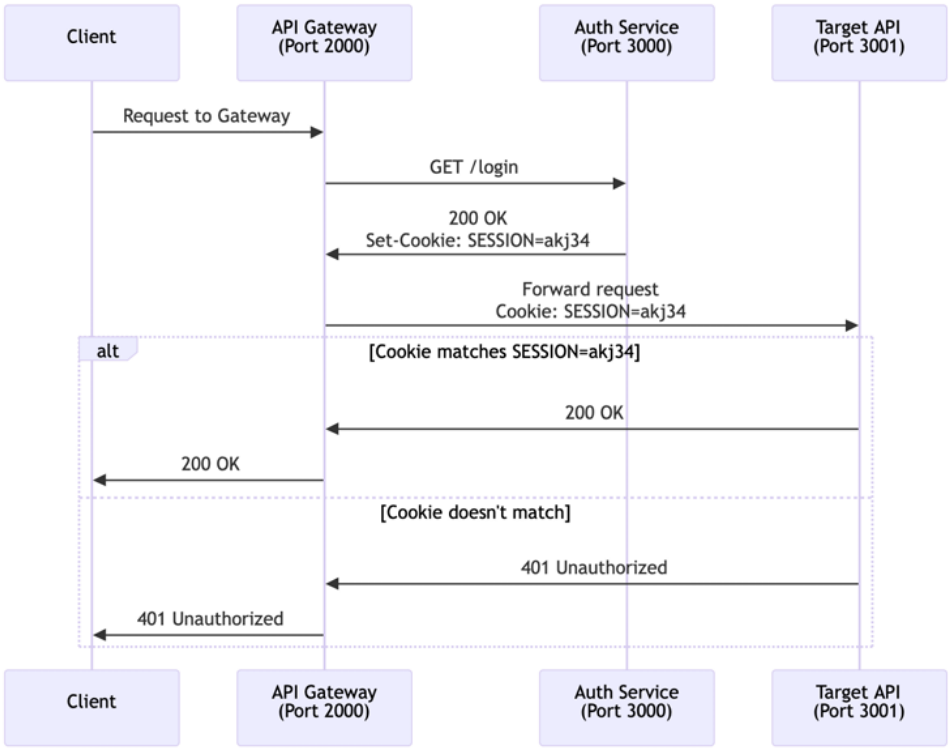


Рисунок 1.1 – API Gateway автентифікується від імені клієнта

Для нашого проєкту цей посібник є незамінним інструментом при проєктуванні зовнішнього контуру інтелектуального API-модуля. Описані в ній механізми контролю трафіку (rate limiting) дозволяють нам запобігти перевантаженню системи та контролювати витрати при зверненні до платних сервісів великих мовних моделей. Патерни шлюзу, викладених у книзі, допоможуть реалізувати безпечну та прозору інтеграцію з різними LMS, забезпечуючи при цьому високу швидкість обробки даних і стабільність з'єднання при передачі великих обсягів навчального контенту.

«Azure Data and AI Architect Handbook» [3] – це стратегічний маніфест для тих, хто будує складні інтелектуальні системи у хмарі Microsoft. Автори пропонують не просто технічний опис сервісів, а цілісну методологію проєктування сучасних екосистем даних, де штучний інтелект стає центральним ядром, а не додатковою функцією. Видання детально розбирає найважливіші архітектурні виклики: від масштабування векторних баз даних до впровадження принципів

відповідального ШІ, надаючи чіткі інструкції, як перетворити розрізнені сервіси Azure на єдиний, високонадійний механізм обробки інформації.

У розрізі нашого проєкту цей посібник стає головною дорожньою картою для розгортання інтелектуального API в екосистемі Azure. Він допомагає нам професійно обґрунтувати вибір конкретних інструментів для реалізації RAG–архітектури, зокрема інтеграцію Azure OpenAI та керування векторними пошуковими запитами. Завдяки цій книзі ми отримуємо перевірені шаблони для побудови масштабованого конвеєра даних, що дозволяє модулю стабільно опрацьовувати навчальний контент, гарантуючи при цьому захищеність корпоративного рівня та високу швидкість відгуку всієї системи. Приклад ілюстрації з цього джерела наведено на рисунку 1.2.

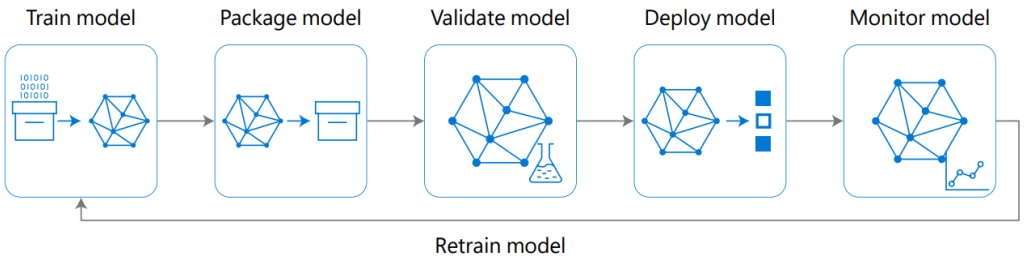


Рисунок 1.2 – Візуальний огляд потоку MLOps та циклу зворотного зв'язку моделі

Стаття «Machine Learning at Microsoft with ML.NET» [4] є програмним документом, який описує еволюцію та внутрішню кухню одного з найпотужніших інструментів для розробників в екосистемі .NET. Автори розкривають амбітну мету Microsoft – стерти межу між академічним машинним навчанням та промисловою розробкою ПЗ, пропонуючи фреймворк, що дозволяє інтегрувати складні моделі безпосередньо в основний код програми. У документі детально розібрано архітектуру ML.NET, систему конвеєрів даних та результати бенчмарків, які доводять, що цей інструмент не просто конкурує з популярними Python–бібліотеками, а в багатьох сценаріях перевершує їх за швидкістю та ефективністю використання пам’яті.

Для нашого дослідження ця стаття стає фундаментальним науковим доказом на користь обраного технологічного стеку. Вона надає чітке теоретичне та практичне обґрунтування того, чому використання ML.NET у парі з C# є оптимальним рішенням для побудови високонавантажених освітніх сервісів. Описані в тексті підходи до порівняння продуктивності з Python–бібліотеками стають основою для нашого експериментального розділу, дозволяючи нам фахово підкріпити вибір платформи .NET як надійного середовища для запуску інтелектуальних модулів без втрати швидкості обробки запитів.

На рисунку 1.3 зображено архітектуру та потік виконання NimbusML. Під час імпорту NimbusML ініціалізується екземпляр CoreCLR з ML.NET. Коли конвеєр надсилається на виконання, граф точок входу операторів NimbusML витягується та надсилається на виконання разом з вказівниками на набори даних, якщо вони знаходяться в пам'яті. Виконавець графів виконує граф та повертає результат до програми Python як об'єкт фрейму даних.

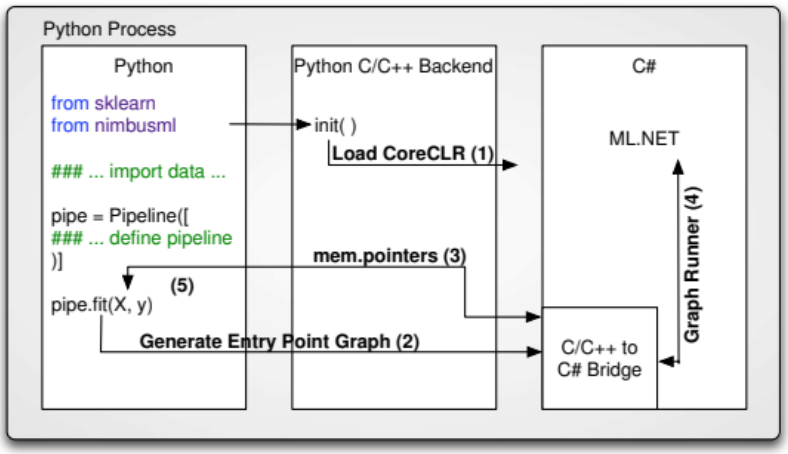


Рисунок 1.3 – Архітектура та потік виконання NimbusML

Стаття «Microsoft Agent Framework: Building Blocks for AI – Part 3» [5] фокусується на еволюції інтелектуальних систем у середовищі .NET – від простих викликів мовних моделей до складних агентних архітектур. У ній детально розібрано механізми побудови автономних агентів, які здатні самостійно розбивати комплексні завдання на підзадачі, оперувати контекстною пам'яттю та залучати

зовнішні інструменти через виклики API. Основна увага приділяється створенню багатокрокових робочих процесів, де ШІ не просто генерує текст, а виступає активним диригентом, що координує виконання логічних операцій для досягнення цілісного результату.

У контексті даної роботи цей матеріал є ключовим для архітектурного обґрунтування переходу від лінійної генерації контенту до інтелектуального агентного підходу. Описані в статті блоки дозволяють спроєктувати ваш API-модуль як мережу спеціалізованих агентів, де кожен відповідає за окремий аспект педагогічної роботи. Це підсилило би технічну складність проєкту, демонструючи використання найбільш актуальних фреймворків Microsoft для вирішення конкретних освітніх завдань у стеку .NET.

Agent Framework надає AIContextProvider, механізм для введення контекстної інформації в робочий процес агента [5]. На лістингу 1.1 – спрощена версія зразка пам'яті (04_memory), який витягує та запам'ятовує інформацію про користувача.

Лістинг 1.1 – Програмний код механізма для введення контекстної інформації в робочий процес агента

```
internal sealed class UserInfoMemory : AIContextProvider
{
    private readonly ProviderSessionState<UserInfo> _sessionState;
    private readonly IChatClient _chatClient;

    public UserInfoMemory(IChatClient chatClient)
    {
        _sessionState = new ProviderSessionState<UserInfo>(
            _ => new UserInfo(),
            GetType().Name);
        _chatClient = chatClient;
    }

    protected override async ValueTask StoreAIContextAsync(
        InvokedContext context,
        CancellationToken cancellationToken = default)
    {
        var userInfo =
_sessionState.GetOrInitializeState(context.Session);

        if (userInfo.UserName is null
```

```
            && context.RequestMessages.Any(x => x.Role ==
ChatRole.User))
        {
            var result = await _chatClient.GetResponseAsync<UserInfo>(
                context.RequestMessages,
                new ChatOptions()
                {
                    Instructions =
                      "Extract the user's name from the message if present."
                },
                cancellationToken: cancellationToken);

            userInfo.UserName ??= result.Result.UserName;
        }

        _sessionState.SaveState(context.Session, userInfo);
    }

    protected override ValueTask<AIContext> ProvideAIContextAsync(
        InvokingContext context,
        CancellationToken cancellationToken = default)
    {
        var userInfo =
_sessionState.GetOrInitializeState(context.Session);

        var instructions = userInfo.UserName is null
            ? "Ask the user for their name."
            : $"The user's name is {userInfo.UserName}.";

        return new ValueTask<AIContext>(
            new AIContext { Instructions = instructions });
    }
}
```

Офіційна документація Microsoft Learn [6] є першоджерелом технічної істини для всієї екосистеми .NET. Вона пропонує вичерпні стандарти проєктування та експлуатації програмного забезпечення. Тож, це не просто збірка інструкцій, а динамічна база знань, що визначає кращі галузеві практики розробки, безпеки та масштабування сучасних корпоративних систем. У рамках дослідницької роботи, цей ресурс стане фундаментальним еталоном, який дозволяє валідувати кожен архітектурний крок згідно з актуальними вимогами індустрії.

Розділи, присвячені мові C#, забезпечують глибоке розуміння її внутрішньої архітектури. Для нашого проєкту це джерело є критично важливим, оскільки воно

регламентує використання високорівневих конструкцій, необхідних для швидкої обробки текстових масивів та надійної роботи серверної частини API. Офіційні керівництва допомагають нам уникнути типових помилок при роботі з багатопотоковістю, що безпосередньо впливає на швидкість генерації навчального контенту.

Документація ML.NET [7] розкриває методи безшовної інтеграції машинного навчання в .NET-додатках, пропонуючи готові сценарії для класифікації текстів, аналізу тональності та семантичного пошуку. Вона надає чіткі інструкції щодо побудови конвеєрів обробки даних та використання інструментів AutoML, що дозволяє нам фахово обґрунтувати вибір цього фреймворку як потужної альтернативи Python.

На рисунку 1.4 зображено діаграму структури коду програми та ітераційного процесу розробки моделі. Детальніший опис процесу, зображеного на діаграмі [7]:

- Збирання та завантаження навчальних даних в об'єкт IDataView.
- Вказування конвеєра операцій для вилучення ознак та застосовування алгоритму машинного навчання.
- Навчання моделі, викликаючи Fit(IDataView) у конвеєрі.
- Оцінка моделі та виконання ітерації для покращення.
- Зберігання моделі у двійковому форматі для використання в застосунку.
- Завантаження моделі назад в об'єкт ITransformer.
- Створення прогнозів, викликаючи PredictionEngineBase<TSrc,TDst>.Predict.

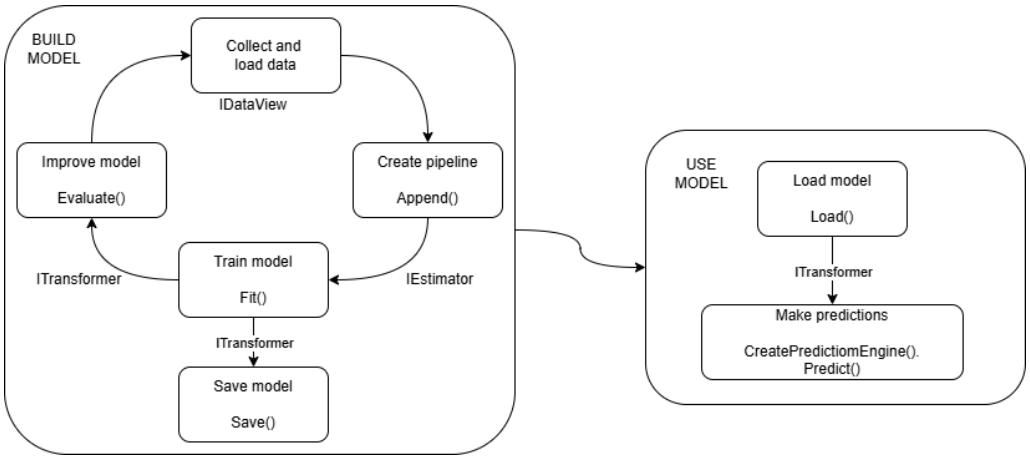


Рисунок 1.4 – Діаграма структури коду програми та ітераційного процесу розробки моделі

Додаток ML.NET починається з об'єкта MLContext. Цей об'єкт-синглтон містить каталоги. Каталог – це фабрика для завантаження та збереження даних, перетворень, трейнерів та компонентів операцій моделі. Кожен об'єкт каталогу має методи для створення різних типів компонентів.

Таблиця 1.1 – Каталоги об'єкта MLContext

Задача	Каталог
Завантаження і збереження даних	DataOperationsCatalog
Підготовка даних	TransformsCatalog
Двійкова класифікація	BinaryClassificationCatalog
Багатокласова класифікація	MulticlassClassificationCatalog
Виявлення аномалій	AnomalyDetectionCatalog
Кластеризація	ClusteringCatalog
Прогнозування	ForecastingCatalog
Ранжування	RankingCatalog
Регресія	RegressionCatalog
Рекомендація	RecommendationCatalog
Часові ряди	TimeSeriesCatalog
Операція з моделлю	ModelOperationsCatalog

Стаття «Integrating Machine Learning with Fullstack Development Using ML.NET» [8] присвячена трансформації сучасних веб–додатків шляхом впровадження інтелектуальних компонентів безпосередньо у внутрішню структуру системи. Автори досліджують потенціал фреймворку ML.NET як інструменту, що дозволяє розробникам будувати та тренувати моделі машинного навчання без глибокої спеціалізації у сфері даних. Основна увага приділяється вирішенню проблем статичності контенту: стаття демонструє, як інтеграція ШІ допомагає автоматизувати прийняття рішень, персоналізувати досвід користувача та прогнозувати його дії в режимі реального часу. Пропозиції, які викладені в статті авторами [8]:

– заохочуйте розробників повного циклу (full-stack) зосереджуватися на опануванні основних технологій, водночас приділяючи час постійному навчанню. Організації можуть підтримати це, надаючи навчальні програми та ресурси для підвищення кваліфікації;

– впроваджуйте стиль керівництва, який наголошує на наставництві, чіткій комунікації та розширенні можливостей команди. Керівники повинні зосередитися на створенні сприятливого середовища, де члени команди можуть процвітати та ефективно робити свій внесок;

– використовуйте сучасні методи проектування, такі як мікросервіси та контейнеризація, щоб забезпечити готовність систем до зростання. Регулярно переглядайте та оптимізуйте продуктивність системи, щоб проактивно вирішувати проблеми масштабованості;

– встановіть чіткі канали комунікації між членами команди, зацікавленими сторонами та клієнтами. Такі інструменти, як програмне забезпечення для управління проєктами, можуть допомогти підтримувати узгодженість та прозорість на всіх етапах розробки;

– заохочуйте використання найкращих практик, таких як модульне кодування, дотримання принципів SOLID та регулярні перевірки коду. Автоматизовані системи

тестування можуть допомогти підтримувати високоякісну кодову базу та зменшити технічний борг;

– випереджайте галузеві тенденції, досліджуючи нові інструменти та методології, такі як безсерверні обчислення та практики DevOps. Ці інновації можуть оптимізувати робочі процеси та підвищити ефективність системи;

Для нашого проєкту цей матеріал слугує вагомим архітектурним підкріпленням, доводячи доцільність вибору стеку .NET для створення освітніх сервісів. Описані в роботі підходи до поєднання клієнтської та серверної частин із ML-моделями дозволяють нам професійно спроєктувати API-модуль як єдину, високопродуктивну екосистему.

«Programming ML.NET» [9] від двох Еспозіто виступає свого роду технологічним мостом, що перетворює машинне навчання з елітарної дисципліни для обраних на доступний інструмент для широкого кола інженерів. Автори пропонують подивитися на інтелектуальні алгоритми не як на ізольовану «чорну скриньку», а як на органічне розширення можливостей платформи .NET, де звичний синтаксис C# стає ключем до складних предиктивних моделей. Книга майстерно розвінчує міф про недосяжність ML для прикладних програмістів, фокусуючись на практичній інтеграції обчислювальних конвеєрів безпосередньо в екосистему Visual Studio.

Технічна глибина видання вражає своєю послідовністю: від фундаментальних основ математичної статистики до розгортання моделей у промислових масштабах. Еспозіто детально розбирають повний життєвий цикл ML-рішення – завантаження даних, їх трансформацію, тренування та фінальну валідацію – надаючи розробнику готові рецепти для вирішення задач класифікації чи семантичного аналізу. Це не просто опис функцій бібліотеки, а глибоке занурення в архітектурну філософію, де машинне навчання стає прогнозованим та керованим інженерним процесом, позбавленим зайвої академічної перевантаженості.

На рисунку 1.5 представлено абстрактне та конкретне уявлення про те, як клієнтська програма зрештою використовує навчену модель. Вхідні дані надходять, а шестерні графа обробляють числа та генерують відповідь для програми.

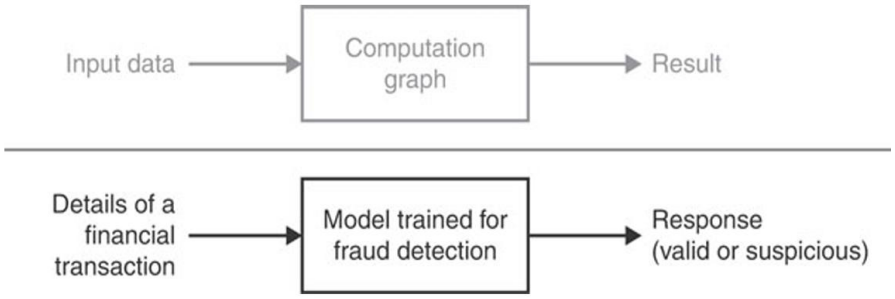


Рисунок 1.5 – Загальна схема навченої моделі, що використовується у виробництві

Для розробки надійного освітнього сервісу ця праця чудово підходить як орієнтир архітектури. «Programming ML.NET» надає чітке підтвердження того, що обрана платформа здатна ефективно обробляти складні сценарії генерації контенту, зберігаючи при цьому стабільність та високу швидкість відгуку, властиву корпоративним системам.

1.2 Характеристика предметної області

Характеристика предметної області є фундаментальним етапом дослідження, оскільки вона визначає межі теоретичного аналізу та формує базис для подальшого проєктування системи. У системному проєктуванні під предметною областю розуміють сукупність об'єктів [10], процесів та їхніх взаємозв'язків, що підлягають автоматизації або дослідженню. Для даної кваліфікаційної роботи предметна область охоплює методи машинного навчання, технології обробки природної мови та архітектурні рішення [11] на основі великих мовних моделей, що інтегруються в освітні екосистеми для інтелектуального аналізу та генерації навчального контенту.

Основними категоріями, що формують структуру цієї області, є методи машинного навчання (ML) та обробки природної мови (NLP), які забезпечують

здатність системи інтерпретувати текстові дані. Великі мовні моделі (LLM) виступають центральним технологічним компонентом, що дозволяє виконувати складні задачі генерації та реферування. Для забезпечення достовірності знань використовується технологія пошуку, розширеного генерацією (RAG), яка дозволяє моделі спиратися на верифіковану базу знань курсу. Взаємодія між цими компонентами та системами керування навчанням здійснюється через прикладні інтерфейси програмування (API), що дозволяє реалізувати модульний підхід до побудови інтелектуальних сервісів. Семантичне структурування при цьому виконує роль механізму впорядкування неструктурованих даних у логічно послідовні навчальні елементи.

Історичний розвиток предметної області демонструє швидку еволюцію від класичних статистичних методів та рекурентних нейронних мереж до сучасних архітектур [13-16] на основі механізмів уваги (Transformers). Сучасні тенденції в освіті характеризуються відходом від статичного збереження інформації до створення адаптивних середовищ, де ШІ стає активним учасником педагогічного дизайну. Попри значні успіхи, галузь стикається з серйозними викликами, серед яких головними є схильність моделей до «галюцинацій», складність збереження контекстуальної точності у вузькоспеціалізованих дисциплінах та гостра потреба в інструментах персоналізації навчального процесу без втрати якості методичного забезпечення.

Наукові підходи у даній області базуються на комбінації методів семантичного аналізу та класифікації знань [17]. У роботі особлива увага приділяється методам оперативного проєктування (Prompt Engineering) для точного налаштування результатів генерації та використанню RAG-архітектур для мінімізації помилок. Важливим аспектом дослідження є порівняння технологічних стеків, зокрема аналіз продуктивності між екосистемами .NET та Python, що дозволяє визначити найбільш ефективне середовище для розгортання інтелектуальних API-модулів у корпоративних та університетських мережах. Статистична оцінка якості згенерованого тексту при цьому виступає об’єктивним критерієм підтвердження дієвості обраних алгоритмів.

Зв’язок предметної області з темою дослідження є безпосереднім, оскільки саме методи інтелектуальної обробки даних визначають можливість створення функціонального модуля для автоматизації роботи викладача. Глибоке дослідження процесів структурування даних дозволяє трансформувати звичайні лекційні матеріали у динамічні навчальні об’єкти, такі як плани занять, конспекти та бази тестових питань. Актуальність обраної області для сучасних освітніх технологій підтверджується необхідністю створення масштабованих інструментів, що здатні підвищити ефективність навчання в умовах глобальної цифровізації та зростаючого навантаження на освітню інфраструктуру.

Висновки до даного підрозділу дозволяють констатувати, що обрана предметна область визначає сталий теоретичний фундамент для розробки інтелектуального API-модуля. Узагальнення характеристик галузі підкреслює актуальність впровадження методів машинного навчання в LMS як необхідного кроку для модернізації систем дистанційної освіти. Чітке визначення меж предметної області та аналіз наявних методів дозволяють перейти до наступного етапу дослідження – деталізації галузі застосування результатів роботи та аналізу потреб потенційних користувачів.

1.3 Визначення сфери застосування результатів

Визначення сфери застосування результатів дослідження окреслює практичний контекст використання розроблених методів та оцінити їхній потенційний вплив на сучасну освітню інфраструктуру. У межах моєї кваліфікаційної роботи сфера застосування охоплює широкий спектр галузей, пов'язаних із розвитком освітніх технологій (EdTech), функціонуванням систем керування навчанням (LMS) та організацією корпоративного сектору підготовки кадрів. Оскільки розроблюваний інтелектуальний API-модуль базується на універсальних принципах обробки природної мови та архітектурі великих мовних моделей, результати дослідження можуть бути успішно адаптовані для різних типів

навчальних середовищ, де виникає потреба у швидкій та якісній обробці значних масивів неструктурованої інформації.

Основними галузями застосування результатів роботи виступають насамперед заклади вищої освіти (ЗВО), які сьогодні стикаються з викликом масової цифровізації традиційних курсів. Впровадження інтелектуальної генерації дозволяє автоматизувати створення лекційних конспектів, структурування планів занять та формування бази тестових питань, що суттєво пришвидшує процес наповнення цифрових платформ. Водночас у системах дистанційного навчання результати дослідження знаходять своє відображення в інструментах персоналізації контенту, де система здатна адаптувати складність та обсяг матеріалу під індивідуальні потреби студента. Окремим важливим сегментом є корпоративне навчання, де динаміка оновлення знань є надзвичайно високою - тут використання API-модуля дозволяє оптимізувати підготовку нових кадрів та знизити навантаження на внутрішніх тренерів за рахунок швидкої трансформації технічної документації у навчальні модулі. Крім того, розроблені підходи можуть слугувати допоміжним інструментом для спеціалістів із педагогічного дизайну, забезпечуючи їх автоматизованими чернетками методичних матеріалів для подальшого експертного опрацювання.

Аналіз потреб сфери застосування свідчить про наявність низки проблем, які моє дослідження покликане вирішити. Одним із головних викликів є надмірне когнітивне навантаження на викладачів, які змушені витрачати значну частину часу на технічне оформлення курсів та розробку контрольних заходів замість безпосередньої взаємодії зі студентами. Існуюча недостатня персоналізація навчального процесу часто призводить до зниження мотивації слухачів через невідповідність матеріалу їхньому рівню підготовки. Висока трудомісткість створення якісного контенту та потреба у масштабованих рішеннях, здатних працювати з різними дисциплінами, створюють гострий запит на впровадження інтелектуальних систем, які могли б забезпечити гнучкість освітнього процесу без втрати його академічної глибини.

Очікувані результати впровадження розробленого модуля полягають у суттєвому зниженні трудових витрат на підготовку навчально-методичного забезпечення. Автоматизація рутинних операцій, таких як виокремлення ключових концепцій із тексту та генерація на їхній основі перевірочних завдань, дозволяє підвищити якість та адаптивність навчання. Завдяки використанню методів машинного навчання, згенеровані матеріали володіють високим рівнем семантичної зв'язності, що позитивно впливає на сприйняття інформації користувачами. Універсальність запропонованого API-підходу гарантує масштабованість рішення, дозволяючи використовувати єдине інтелектуальне ядро для обслуговування різних типів освітніх платформ, незалежно від їхньої внутрішньої архітектури чи мови розробки.

Практичне впровадження результатів роботи передбачає інтеграцію API-модуля у найбільш розповсюджені системи, такі як Moodle, Canvas або Google Classroom. Такий підхід дозволяє викладачу завантажити вихідний файл у звичному інтерфейсі та отримати автоматично сформовану структуру курсу або набір тестів, що базуються на завантаженому матеріалі. У корпоративному секторі результати дослідження можуть бути інтегровані у платформи на кшталт SAP SuccessFactors або спеціалізовані сервіси для бізнесу, забезпечуючи швидку адаптацію нових співробітників через автоматизовану перевірку знань. Крім того, існує значний потенціал для створення на базі отриманих результатів мультимодальних освітніх асистентів, здатних не лише генерувати текст, а й рекомендувати супутні ресурси чи візуальні матеріали.

Підсумовуючи, можна стверджувати, що визначена сфера застосування результатів дослідження безпосередньо підтверджує його практичну значущість та актуальність. Розроблені методи інтелектуальної обробки та генерації контенту знаходять своє місце як у класичній академічній освіті, так і в гнучкому корпоративному навчанні, створюючи місток між можливостями сучасного штучного інтелекту та реальними потребами користувачів освітніх систем. Окреслення меж застосування дозволяє мені перейти до подальших розділів

роботи, де будуть детально розглянуті архітектурні рішення та програмна реалізація системи, що відповідає виявленим запитам галузі.

1.4 Постановка задачі дослідження

Постановка задачі є логічним підсумком проведеного аналізу предметної області та вивчення актуальних потреб сфери застосування сучасних освітніх технологій. Вона формує чітку дорожню карту мого дослідження, визначаючи послідовність наукових та практичних кроків, які необхідно здійснити для досягнення поставленої мети. Цей етап дозволяє трансформувати загальну ідею дослідження у конкретний перелік дослідницьких завдань, що забезпечує методологічну цілісність всієї магістерської роботи.

Основна проблема, на розв’язання якої спрямоване дане дослідження, полягає у відсутності доступних та легкоінтегрованих автоматизованих інструментів для інтелектуальної обробки та генерації навчального контенту безпосередньо в існуючих системах керування навчанням. Сучасний стан розвитку EdTech-середовищ демонструє значний розрив між потенційними можливостями великих мовних моделей та їх реальним впровадженням у повсякденну практику закладів освіти. Наслідком цього є критичне перевантаження викладацького складу рутинними завданнями з підготовки методичних матеріалів та недостатній рівень персоналізації навчання, що робить розробку інтелектуальних засобів автоматизації вкрай актуальною науково-технічною задачею.

Загальною метою мого дослідження є розробка та наукове обґрунтування архітектури інтелектуального API-модуля для автоматизованої генерації освітнього матеріалу на основі великих мовних моделей. Такий підхід дозволить підвищити ефективність освітнього процесу через надання універсального інструменту, який здатний трансформувати неструктуровані дані у якісні навчальні ресурси. Мета роботи безпосередньо відповідає на виявлені виклики, пропонуючи модульне рішення, яке може бути інтегроване в будь-яку існуючу LMS без необхідності її повної перебудови.

Для досягнення поставленої мети у роботі необхідно вирішити наступну послідовність завдань. По-перше, слід провести поглиблений аналіз існуючих архітектур великих мовних моделей та методів обробки природної мови для вибору оптимального технологічного рішення. По-друге, необхідно дослідити можливості застосування RAG-технологій як інструменту для семантичного структурування даних та забезпечення контекстуальної точності контенту. По-третє, потрібно спроєктувати архітектуру API-модуля, що базується на принципах слабкої зв'язності та масштабованості. По-четверте, слід реалізувати програмні алгоритми для автоматизованого формування планів занять, конспектів та тестових завдань. По-п’яте, важливо провести експериментальне порівняння реалізацій на базі платформ ML.NET та Python для визначення найбільш ефективного стеку. По-шосте, необхідно виконати комплексну оцінку якості згенерованих матеріалів та підготувати рекомендації щодо впровадження результатів у практику ЗВО та корпоративного навчання.

Очікуваними результатами мого дослідження є діючий прототип інтелектуального модуля у вигляді прикладного інтерфейсу програмування, розроблена система критеріїв для оцінювання якості згенерованого контенту та науково обґрунтовані методичні рекомендації щодо їх експлуатації. Наукова новизна роботи полягатиме в удосконаленні методів динамічного структурування навчальних даних через інтеграцію RAG-архітектур та розробці моделі генерації завдань, що враховує когнітивну складність матеріалу. Практична значущість підтверджуватиметься можливістю швидкої інтеграції модуля в існуючі освітні платформи для зниження трудомісткості педагогічного дизайну.

Таким чином, постановка задачі визначає наскрізну логіку всієї роботи. Чітке визначення завдань дозволяє забезпечити наукову достовірність отриманих результатів та гарантувати їхню відповідність сучасним вимогам до інтелектуальних систем системного проєктування.

1.5 Методологія проведення дослідження

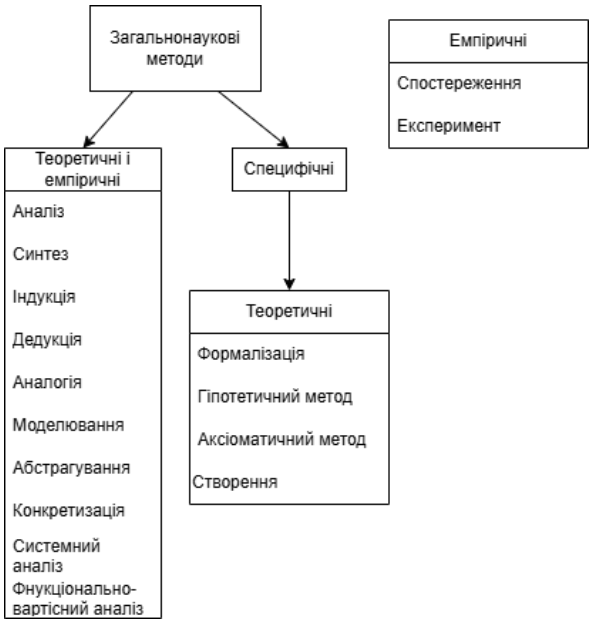


Рисунок 1.6 – Методи наукових та експериментальних досліджень [18]

Методологія проведення дослідження є системною основою магістерської роботи, яка визначає сукупність підходів, методів та інструментальних засобів, необхідних для розв’язання поставлених завдань. Вона базується на принципах системного проєктування, що дозволяє розглядати процес інтелектуальної генерації контенту не як ізольовану функцію, а як складний взаємозв’язок між базою знань, алгоритмами обробки природної мови та зовнішньою інфраструктурою освітніх платформ. Обраний методологічний апарат забезпечує наукову достовірність отриманих результатів та дозволяє поєднати теоретичні розробки з практичною реалізацією у формі програмного API-модуля.

На початковому етапі дослідження застосовується метод системного аналізу предметної області, що дозволяє структурувати функціональні вимоги до сучасних систем керування навчанням та виявити критичні точки в процесах підготовки навчальних матеріалів. Використання методів порівняльного аналізу архітектур великих мовних моделей та існуючих NLP-фреймворків дає змогу обґрунтувати вибір технологічного стеку. Важливою частиною теоретичної методології є вивчення концепцій семантичного структурування даних, що базуються на теорії графів та методах векторного представлення знань, що створює підґрунтя для

розробки алгоритмів, здатних виявляти логічні зв’язки всередині неструктурованих текстів.

Центральне місце в методології займає підхід, що базується на технології пошуку, розширеного генерацією (RAG). Цей метод використовується як основний інструмент забезпечення контекстуальної точності та мінімізації «галюцинацій» мовних моделей через обмеження простору пошуку відповідей конкретними навчальними матеріалами. У межах цього підходу застосовуються методи оперативної розробки (Prompt Engineering), що розглядаються як наукова методика тонкого налаштування поведінки LLM для отримання результатів, які відповідають заданим педагогічним критеріям, таким як рівень складності за таксономією Блума або відповідність заданому формату (конспект, тест, план заняття).

Практична частина дослідження реалізується через методологію прототипування, що передбачає створення модульного прикладного інтерфейсу програмування (API). Проєктування архітектури системи базується на принципах слабкої зв'язності (loose coupling) та кросплатформеності, що дозволяє інтегрувати розроблене рішення у різнорідні програмні середовища. Особлива увага приділяється методам асинхронної обробки даних та керування потоками виконання, що є критично важливим для стабільної роботи інтелектуальних сервісів в умовах високого навантаження.

Експериментальна частина методології передбачає проведення порівняльного дослідження продуктивності двох технологічних платформ – .NET (із використанням фреймворку ML.NET) та Python (із застосуванням стандартних бібліотек обробки природної мови). Для оцінки ефективності розроблених рішень використовується комплексний набір метрик, що включає часові показники (затримка відповіді, час обробки токенів), ресурсні показники (використання оперативної пам'яті та процесорного часу), а також специфічні NLP-метрики для оцінки якості згенерованого тексту, такі як ROUGE та BLEU. Крім статистичних методів, методологія передбачає елементи експертного оцінювання педагогічної релевантності матеріалів, що дозволяє перевірити відповідність результатів генерації реальним освітнім стандартам.

Запропонована методологія проведення дослідження охоплює повний цикл розробки інтелектуального модуля. Поєднання класичних методів системного аналізу з сучасними підходами до роботи з великими мовними моделями дозволяє не лише створити працездатний прототип модуля, а й підтвердити ефективність обраних архітектурних рішень для вдосконалення систем керування навчанням у вищій школі та корпоративному секторі.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ API-МОДУЛЯ ДЛЯ ІНТЕГРАЦІЇ LLM У СИСТЕМИ КЕРУВАННЯ НАВЧАННЯМ

2.1 Аналіз існуючих систем управління навчанням

Аналіз наявного ринку систем керування навчанням та глобальних веб-платформ для розміщення освітніх курсів є необхідним для визначення технологічних стандартів та функціональних прогалин у сучасній цифровій освіті. Дослідження існуючих рішень дає можливість систематизувати підходи до організації дистанційного навчання, оцінити рівень автоматизації рутинних процесів та виявити обмеження, що перешкоджають масштабованості педагогічних процесів. Попри значне різноманіття програмних продуктів, більшість із них зосереджена на адмініструванні та статичному збереженні контенту, залишаючи поза увагою можливості динамічної інтелектуальної обробки даних. Також, аналіз ринку створює обґрунтовану базу для впровадження інноваційних модулів, що використовують методи машинного навчання для трансформації традиційних освітніх середовищ у високотехнологічні адаптивні системи. У таблиці 2.1 зображено стисло переваги та недоліки кожної розглянутої платформи.

Таблиця 2.1 – Переваги та недоліки розглянутих LMS

Платформа	Переваги	Недоліки
Google Classroom	Безшовна інтеграція з Google Workspace; максимальна простота розгортання та використання.	Відсутність інструментів семантичного аналізу; обмежений функціонал для структурування знань.
Moodle	Необмежена кастомізація через плагіни; повний контроль над безпекою та локальним зберіганням даних.	Висока складність адміністрування; низька продуктивність вбудованих ML–модулів (на базі PHP).

Кінець таблиці 2.1

Платформа	Переваги	Недоліки
Coursera	Високотехнологічні AI–асистенти (Coach); суворі академічні стандарти контенту.	Закритість екосистеми для сторонніх розробників; висока вартість та жорстка структура курсів.
Udemy	Висока швидкість створення та публікації курсів; розвинені інструменти маркетингу та аналітики для авторів.	Нестабільна якість навчальних матеріалів; брак інструментів для глибокого педагогічного дизайну.

2.1.1 Google Classroom

Google Classroom сьогодні є одним із найбільш впізнаваних інструментів у сфері цифрової освіти, який фактично переосмислив поняття доступності навчальних технологій. На відміну від класичних, складних LMS, Classroom створювався як полегшена оболонка для екосистеми Google Workspace, що дозволило миттєво масштабуватися під час глобальних викликів дистанційного навчання. Його головна філософія полягає в максимальному спрощенні комунікації між викладачем та студентом, де Google Drive виступає сховищем, а Docs та Forms – основними інструментами створення контенту. Проте саме ця простота і водночас певна обмеженість функціоналу роблять Classroom ідеальним об’єктом для аналізу в межах розробки більш інтелектуальних рішень.

Функціональна архітектура платформи побудована навколо управління потоком завдань, а не навколо створення інтелектуального контенту. Google Classroom майстерно автоматизує рутину: розподіл копій документів, збір робіт у визначені терміни та інтеграцію відеоконференцій через Meet. Проте система залишається «пасивною» стосовно самого навчального матеріалу. Вона не пропонує інструментів для автоматичного структурування лекцій, генерації перевірочних тестів на основі завантажених файлів чи створення стислих

конспектів. Викладач у Classroom все ще змушений вручну опрацьовувати величезні масиви інформації, щоб перетворити їх на дидактичний матеріал, що створює значне когнітивне навантаження. На рисунку 2.1 зображено інтерфейс викладача, де людина-власник курсу може спостерігати успішність його учасників.

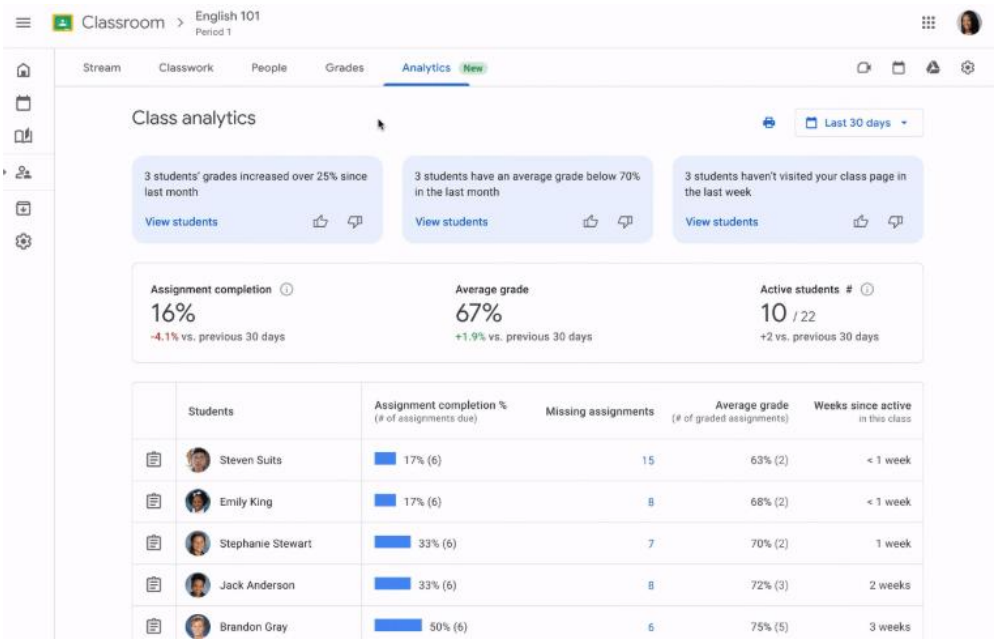


Рисунок 2.1 – Інтерфейс сторінки зібраної аналітики успішності учасників курсу в Google Classroom

З точки зору технічної інтеграції, Google Classroom надає потужний API, який дозволяє стороннім розробникам розширювати можливості платформи. Це відкриває шлях для впровадження зовнішніх модулів, таких як ваш API-проєкт на базі .NET. Classroom чудово справляється з роллю фронтенд-інтерфейсу та системи розподілу прав доступу, але він гостро потребує «інтелектуальних надбудов», здатних виконувати семантичний аналіз завантажених документів. По суті, система є ідеальним транспортним рівнем, який може приймати структуровані дані від сторонніх ML–моделей і доставляти їх кінцевому споживачу в звичному інтерфейсі.

Аналіз обмежень Google Classroom виявляє критичну прогалину між збереженням файлів та їхнім змістовним опрацюванням. Наприклад, система

оцінювання у Classroom залишається досить лінійною і не підтримує складних алгоритмів аналізу якості відповідей, які могли б забезпечити сучасні LLM. Classroom забезпечує масовість та простоту, а наш розроблюваний модуль додає необхідну інтелектуальну глибину, перетворюючи платформу з простого файлообмінника на повноцінний когнітивний асистент. Таким чином, Google Classroom розглядається не просто як конкурент, а як потенційне середовище для розгортання нашого API, де синергія хмарної стабільності Google та потужності вашої архітектури на Azure може дати максимальний результат.

2.1.2 Moodle

Moodle – це визнаний стандарт у середовищі академічної освіти, який домінує завдяки своїй модульній архітектурі та статусу відкритого коду. На відміну від легковагових рішень, ця платформа пропонує глибокий інструментарій для управління курсами, проте її складність часто стає перепоною для швидкого впровадження інновацій. Система вимагає значних ресурсів для підтримки, а її архітектурна консервативність змушує викладачів витрачати чимало часу на ручне налаштування контенту.

Важливим кроком платформи до інтелектуалізації став плагін Tutor AI (рис. 2.2, 2.3), який демонструє реальний запит академічної спільноти на інструменти штучного інтелекту. Він дозволяє автоматизувати генерацію запитань для тестів та структурувати навчальні плани безпосередньо всередині системи. Проте, будучи вбудованим PHP-модулем, він часто стикається з обмеженнями продуктивності при обробці великих обсягів даних або реалізації складних RAG-сценаріїв, які вимагають значних обчислювальних потужностей.

Для нашого проєкту Moodle є ідеальним полігоном для інтеграції зовнішнього API-модуля. Наявність таких інструментів, як Tutor AI, підтверджує готовність ринку до автоматизації, проте архітектура на базі .NET та Azure може запропонувати вищу стабільність і швидкість обробки, недоступну внутрішнім плагінам.

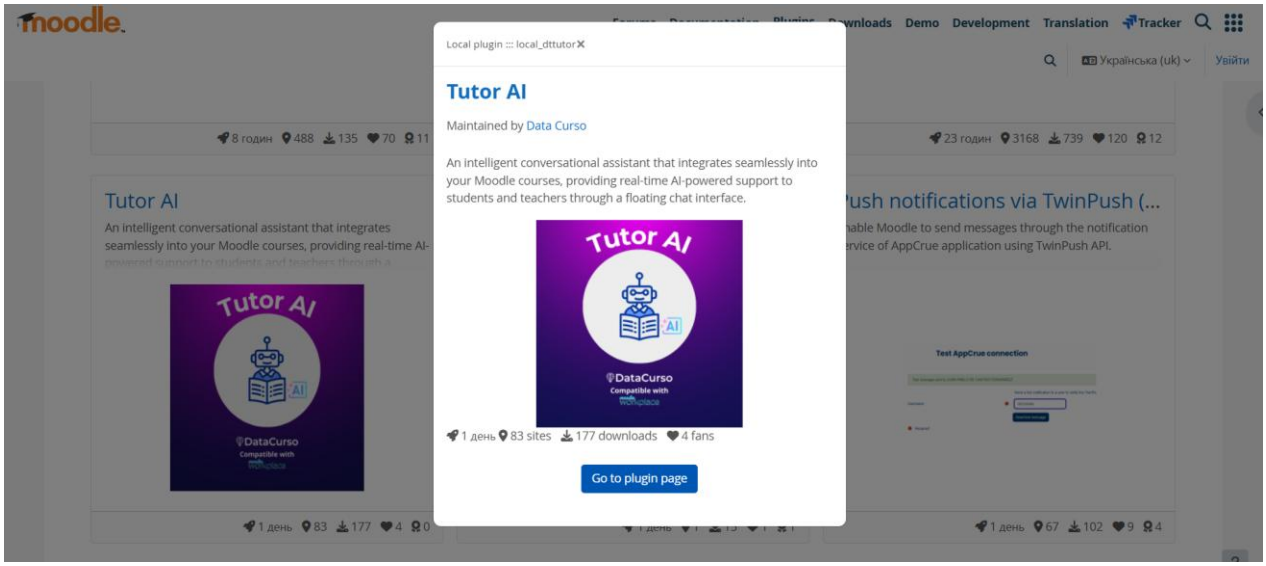


Рисунок 2.2 – Плагін Tutor AI на платформі Moodle

AI Tutor Chat for Moodle

An intelligent conversational assistant that integrates seamlessly into your Moodle courses, providing real-time AI-powered support to students and teachers through a floating chat interface.

What does it do?

AI Tutor Chat adds a floating avatar button to course pages that opens an AI-powered chat drawer. Students and teachers can interact with the AI assistant to:

- **Get instant help** with course content and questions
- **Receive personalized guidance** based on their role (student/teacher)
- **Experience real-time responses** with streaming text display
- **Access contextual assistance** aware of the current course and activity

The chat interface features:

- Customizable avatar with 10 built-in options
- Flexible positioning (bottom-right or bottom-left corner)
- Real-time streaming responses using Server-Sent Events (SSE)
- Automatic role detection for personalized interactions
- Keyboard shortcuts (Enter to send, Escape to close)
- Mobile-responsive design

Рисунок 2.3 – Опис плагіну Tutor AI

2.1.3 Coursera

Coursera виступає глобальним орієнтиром у сфері масових онлайн-курсів, трансформуючи статичний контент провідних університетів у динамічний цифровий продукт. На відміну від традиційних систем, вона функціонує як масштабований агрегатор знань, де ключовий фокус зосереджено на зручності дистрибуції та безперебійному доступі мільйонів користувачів до інтерактивного навчання. Платформа успішно демонструє, як глобальна інфраструктура може

підтримувати високі стандарти академічної якості через автоматизацію рутинних процесів перевірки та адміністрування.

Впровадження інструментів Coursera Coach (рис. 2.4) та AI Course Builder підтверджує перехід індустрії до активного використання генеративного штучного інтелекту для структурування контенту та персоналізації підтримки. Для нашого проєкту це є важливим підтвердженням актуальності RAG-технологій: Coursera вже використовує подібну логіку для створення описів курсів та роз'яснення складних концепцій. Розроблюваний API-модуль фактично заповнює ту саму нішу, але пропонує викладачам більшу автономність у роботі з власними матеріалами, зберігаючи професійний рівень автоматизації, характерний для лідерів ринку.

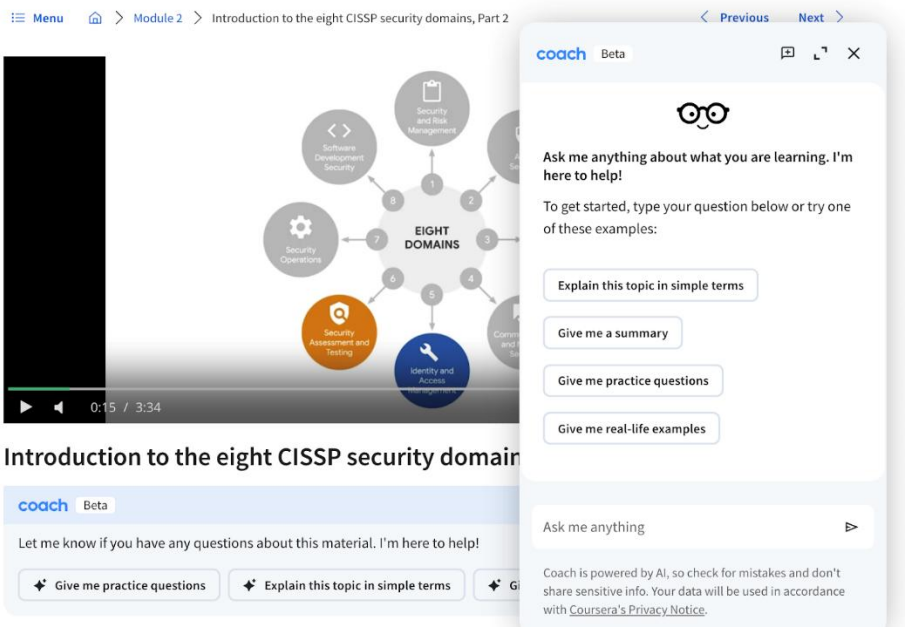


Рисунок 2.4 – ШІ-помічник Coursera Coach

2.1.4 Udemy

Udemy займає унікальну нішу у світі освітніх платформ, функціонуючи не як закрита академічна екосистема, а як відкритий глобальний маркетплейс. На відміну від Coursera, де контент формується університетами, Udemy демократизує навчання, дозволяючи будь-якому експерту створювати та масштабувати власні курси. Ця модель призвела до накопичення колосального масиву неструктурованих

даних, що вимагає надзвичайно гнучкої та потужної інфраструктури для пошуку, рекомендацій та управління якістю контенту. Платформа орієнтована на практичні навички та швидку адаптацію до ринку, що робить її архітектуру еталоном динамічності та масштабованості.

Останнім часом Udemy активно впроваджує інтелектуальні інструменти в межах своєї «Intelligent Skills Platform», використовуючи ШІ для персоналізації навчальних траєкторій та підтримки авторів. Зокрема, платформа надає інструменти для автоматичного створення підсумків лекцій та допомоги у формуванні структури курсу, що значно знижує поріг входу для нових викладачів. Такий підхід демонструє перехід від простого розміщення відео до створення інтелектуального середовища, де алгоритми допомагають структурувати знання та забезпечувати зворотний зв'язок у реальному часі.

Якщо в академічних системах на кшталт Moodle фокус зміщений на контроль, то на маркетплейсах головним викликом є швидкість перетворення сирої інформації на готовий навчальний продукт. Наш API-модуль на базі .NET ідеально вписується в цю логіку, пропонуючи інструментарій для миттєвої генерації конспектів та тестів, що дозволяє авторам контенту значно скоротити час на технічну підготовку матеріалів. Досвід Udemy доводить, що майбутнє освітніх платформ лежить у площині інтелектуальних сервісів, які беруть на себе рутину структурування даних, залишаючи людині роль творчого експерта.

2.2 Огляд технологічного стеку

2.2.1 Мови програмування, їхні бібліотеки та фреймворки

Світ сучасної розробки сьогодні фактично розділений між кількома гігантам, два з яких – C# та Python. Кожен пропонує свій підхід до побудови інтелектуальних систем. Якщо C# традиційно вважається лідером корпоративного сектору завдяки своїй суворій типізації та високій швидкодії, то Python став справжнім стандартом у світі штучного інтелекту через неймовірну гнучкість та величезну кількість

готових математичних бібліотек. Перевага C# полягає у безпеці та продуктивності на високих навантаженнях, тоді як Python виграє за рахунок лаконічності коду та швидкості перетворення ідеї в працюючий прототип.

Платформа .NET пропонує розробнику цілісну екосистему, де ASP.NET [12] виступає в ролі потужного фундаменту для створення швидких та надійних API. Це середовище дозволяє будувати масштабовані веб-сервіси, які легко витримують велику кількість запитів, що критично для освітніх платформ. Для роботи з даними використовується Entity Framework – сучасний інструмент, який перетворює взаємодію з базою даних на роботу зі звичайними об’єктами коду, позбавляючи потреби писати складні SQL-запити вручну.

Унікальність цього стеку підсилюється фреймворком ML.NET (табл. 2.2) [9, 19]. Він дає можливість тренувати та запускати моделі машинного навчання безпосередньо всередині C#-додатку, не змушуючи розробника перемикатися на інші мови. Це значно спрощує архітектуру системи, оскільки і логіка сервісу, і його інтелектуальна складова працюють в одному керованому середовищі, забезпечуючи стабільну продуктивність та легку підтримку коду.

Таблиця 2.2 – Задачі ML.NET

Задача	Опис
AnomalyDetection	Виявлення неочікуваних або незвичайних подій чи поведінку порівняно з отриманим навчанням.
BinaryClassification	Класифікація даних на одну або дві категорії.
Clustering	Розділення даних на кількість можливих корелюючих груп без знання аспектів, що могли б споріднювати дані.
Forecasting	Вирішення проблеми прогнозування, в яких вхідні дані, що передаються моделі, є часовим рядом (послідовністю значень).
MulticlassClassification	Класифікація даних на три і більше категорій.

Кінець таблиці 2.2

Ranking	Вирішення проблеми ранжування, кінцевою метою яких є обчислення релевантності даних. Певною мірою це можна апроксимувати за допомогою багатокласової класифікації.
Regression	Передбачення значення елемента даних. Певною мірою його можна розглядати як надклас задач прогнозування.

Python, у свою чергу, пропонує зовсім іншу філософію, де акцент зміщено на максимально швидку інтеграцію найсучасніших нейромережевих рішень. Для створення серверної частини тут часто обирають FastAPI – лаконічний фреймворк, який за своєю швидкістю наближається до компільованих мов. Його головна перевага полягає в автоматичній генерації документації та вбудованій підтримці асинхронності, що робить його ідеальним вибором для мікросервісів, які взаємодіють із LLM.

Справжня сила Python розкривається у бібліотеках на кшталт LangChain або PyTorch, які фактично є індустріальним стандартом для роботи з LLM та побудови RAG-систем. Ці інструменти дозволяють створювати складні ланцюжки обробки тексту та проводити глибокий семантичний аналіз з мінімальними зусиллями з боку програміста. Хоча інтерпретована природа мови іноді поступається у швидкості сирих обчислень, величезна спільнота та миттєва поява нових бібліотек роблять Python вкрай популярним для впровадження найбільш інноваційних алгоритмів штучного інтелекту.

2.2.2 Системи зберігання та безпеки даних

Будь-який сучасний API тримається на двох стовпах – надійному зберіганні інформації та її захисті від зовнішніх загроз. Обидва наші стеки однаково дружні до популярних БД на кшталт PostgreSQL, SQL Server чи MongoDB, але підходять до роботи з ними з різною філософією. В екосистемі .NET панує Entity Framework Core – потужний інструмент, який уже описано вище. У світі Python схожу роль

виконує SQLAlchemy – гнучкий та маневрений фреймворк, який дає розробнику повну свободу в управлінні даними, хоч і вимагає при цьому більше уваги до деталей.

Безпека в .NET – це не просто додаткова функція, а частина ДНК самої платформи, оскільки ця технологія орієнтована на інтеграцію у великі корпоративні продукти. ASP.NET Core пропонує готові рішення для захисту від найбільш розповсюджених атак, таких як SQL-ін'єкції, крос-сайтний скриптинг (XSS) чи підробка запитів (CSRF), прямо «з коробки». Вбудована система автентифікації та авторизації дозволяє розробнику не винаходити велосипед, а використовувати перевірені часом стандарти для захисту користувацьких даних. Сувора типізація C# тут стає додатковим фільтром: компілятор просто не дозволить коду з очевидними помилками в логіці безпеки потрапити в робочу систему.

Python також має солідний арсенал захисних засобів, але вони часто існують у вигляді сторонніх модулів або специфічних налаштувань фреймворків. Хоча той же FastAPI чудово справляється з перевіркою вхідних даних за допомогою бібліотеки Pydantic, загальний рівень захищеності системи багато в чому залежить від дисципліни розробника. На відміну від .NET, де безпека є цілісним каркасом, у Python-розробці це швидше конструктор, який потрібно вміти правильно зібрати, щоб уникнути вразливостей.

Якщо порівнювати ці два підходи, то головна перевага .NET над Python полягає в корпоративній зрілості та принципі «безпеки за замовчуванням». Microsoft регулярно оновлює платформу, закриваючи нові вразливості на рівні самого фундаменту мови та веб-сервера. Для великих освітніх проєктів, де критично важливо захистити інтелектуальну власність та персональні дані студентів, .NET виглядає надійнішим вибором. Він пропонує передбачуване середовище, де більшість дірок у безпеці закриті ще до того, як програміст напише перший рядок коду, що значно знижує ризики для кінцевого продукту.

2.2.3 Інфраструктура та розгортання проєкту

Наступний етап – винесення написаного коду за межі робочої станції розробника та перетворити на життєздатний сервіс. У сучасному системному проєктуванні стандартом де-факто стала контейнеризація за допомогою Docker (рис. 2.5). Це дозволяє запакувати API-модуль разом з усіма його залежностями, бібліотеками ML та налаштуваннями середовища у легкий образ, який буде працювати однаково стабільно як на локальному сервері, так і у великій хмарі. Для управління такими контейнерами зазвичай використовується Kubernetes (рис. 2.6), який автоматично масштабує систему: якщо кількість запитів на генерацію конспектів різко зростає, він просто запускає додаткові копії сервісу, розподіляючи навантаження без втручання людини.

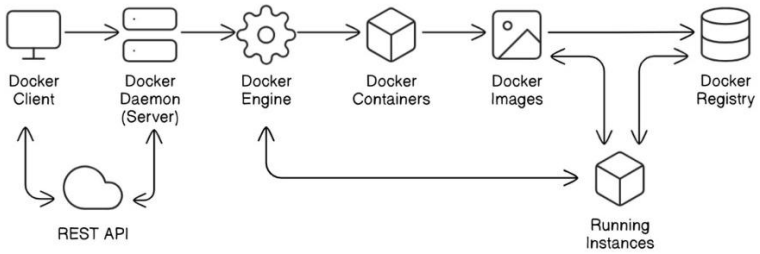


Рисунок 2.5 – Зображення Docker архітектури

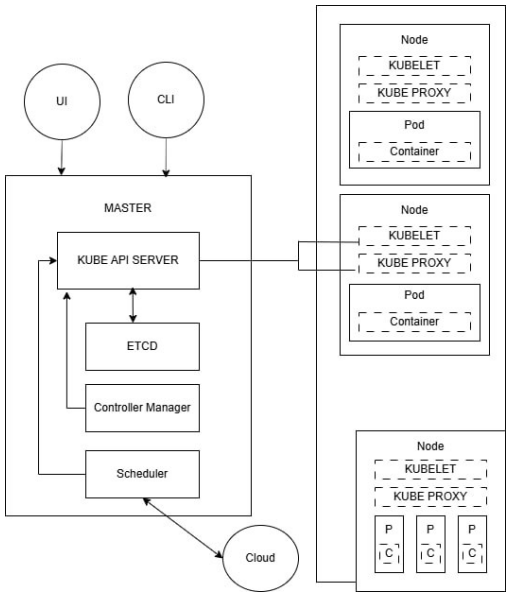


Рисунок 2.6 – Схематичне зображення кластеру Kubernetes

Екосистема .NET найкраще розкривається у хмарі Microsoft Azure, де інтеграція між сервісами доведена до автоматизму. Завдяки високопродуктивному веб-серверу Kestrel, додатки на ASP.NET Core демонструють вражаючу швидкість обробки запитів, а використання Azure DevOps дозволяє налаштувати безперервний конвеєр CI/CD. Це означає, що кожна зміна в коді автоматично проходить через сито тестів і миттєво розгортається в робочому середовищі. Така інфраструктура забезпечує «залізобетонну» стабільність, що особливо важливо для університетських систем, які мають працювати без збоїв під час пікових навантажень у сесійний період.

Python-стек пропонує більш гнучкий та маневрений підхід до інфраструктури, особливо в середовищах AWS або Google Cloud [22]. Тут часто використовується зв’язка з FastAPI та сервером Uvicorn, що дозволяє обробляти тисячі асинхронних з'єднань одночасно. Оскільки проєкти на Python часто зав'язані на важкі нейромережеві моделі, інфраструктура розгортання зазвичай включає спеціалізовані GPU-образи або хмарні сервіси для обробки тензорів. Гнучкість Python дозволяє легко будувати безсерверні (Serverless) архітектури, де окремі функції генерації тестів чи планів занять викликаються лише тоді, коли вони потрібні, що суттєво економить бюджет на утримання серверів.

Особливе місце в розгортанні обох систем посідає API Gateway [2] – інтелектуальний шлюз, який стоїть перед нашими модулями. Він бере на себе роль регулювальника, забезпечуючи безпеку, контроль доступу та кешування відповідей. У поєднанні з сучасними методами моніторингу це дає повну картину стабільності системи в реальному часі. Вибір між стеками тут знову впирається в пріоритети: .NET обирають заради передбачуваності та монолітної надійності корпоративного рівня, тоді як Python стає базою для проєктів, де потрібно миттєво адаптуватися до нових хмарних інструментів та швидко міняти конфігурацію AI-компонентів.

Побудова розумного API вимагає відходу від класичних монолітів на користь гнучких архітектурних патернів, які дозволяють системі дихати та масштабуватися. Основним тут стає патерн мікросервісів, де інтелектуальний модуль виноситься в окремий автономний блок. Це дає змогу не перевантажувати основну навчальну

платформу важкими обчисленнями: поки LMS займається тестами та оцінками, наш сервіс у фоні спокійно перетравлює гігабайти лекцій, перетворюючи їх на конспекти.

Серцем системи стає архітектурний патерн RAG. Це не просто підключення ШІ, а цілий конвеєр, де дані проходять через етап вилучення, векторизації та семантичного пошуку. Замість того, щоб кидати в модель весь підручник одразу, ми використовуємо патерн «пошукового вікна», витягуючи лише найважливіші фрагменти знань. Це дозволяє створювати архітектуру, яка «не галюцинує», а чітко слідує логіці завантаженого викладачем матеріалу.

Оскільки генерація контенту за допомогою нейромереж – процес не миттєвий, критично важливим стає патерн асинхронної взаємодії. Ми не змушуємо користувача дивитися на індикатор завантаження по кілька хвилин; система приймає запит, ставить його в чергу і відпускає клієнта. Коли ШІ закінчує свою магію, сервіс відправляє готовий результат через механізм Webhooks або зберігає його в базі для подальшого отримання. Такий підхід робить роботу з API безшовною та приємною, навіть коли мова йде про обробку дуже складних і довгих курсів.

На рівні внутрішньої логіки самого сервісу найкраще працює багатошарова архітектура (Layered Architecture). Вона дозволяє чітко розділити «транспорт», який приймає HTTP-запити, «бізнес-логіку», що керує правилами створення тестів, та «рівень даних», де зберігаються результати. Така чистота в структурі коду дозволяє легко міняти одну мовну модель на іншу або оновлювати бібліотеки машинного навчання без ризику зламати всю систему.

2.2.4 Сторонні сервіси та API

Робота сучасного інтелектуального модуля була б неможливою без залучення зовнішніх ресурсів, оскільки ми не винаходимо колесо, а використовуємо кращі потужності світових технологічних лідерів. Центром системи стають великі мовні моделі, доступ до яких ми отримуємо через хмарні API. OpenAI з їхнім GPT-4

забезпечує залізну логіку при генерації тестів, Google Gemini підкуповує своєю швидкістю та глибокою інтеграцією з хмарними сховищами, а Claude від Anthropic демонструє неймовірну точність у роботі з великими обсягами лекційного тексту. Така багатополярність дозволяє перемикатися між «мізками» системи залежно від складності завдання та бюджету проєкту.

Організація самої розробки також потребує чіткої інфраструктури, де головним інструментом планування виступає Jira (рис. 2.4). Вона дозволяє розбити складний процес побудови API на зрозумілі етапи, відстежувати баги та керувати релізами. Коли ж справа доходить до кінцевого користувача, система стає частиною його повсякденного цифрового життя через інтеграцію з Google Calendar. Це дозволяє автоматично ставити згенеровані плани занять у розклад, нагадувати про дедлайни або синхронізувати час навчання між викладачем та студентом. Безпечний вхід у систему забезпечується протоколами OAuth, що дозволяє користувачам не створювати нові паролі, а використовувати свої звичні акаунти.

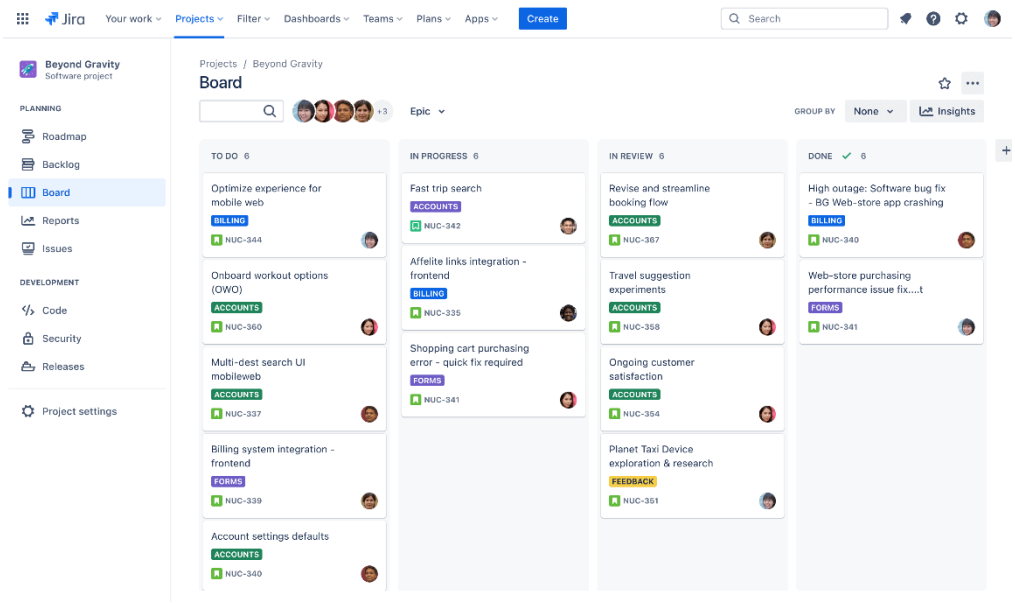


Рисунок 2.4 – Приклад інтерфейсу відслідковування задач у Jira

Бізнесова частина проєкту та його життєздатність тримаються на платіжних шлюзах, таких як Stripe або PayPal. Ці сервіси беруть на себе всю складну логіку міжнародних транзакцій, захисту карткових даних та підписок, якщо мова йде про

платний доступ до корпоративних курсів. Поєднання всіх цих сторонніх API в один вузол створює справжню екосистему, де наш модуль виступає диригентом. Кожен сервіс виконує свою вузьку задачу – від обробки платежу до написання конспекту нейромережею – що в результаті дає цілісний, швидкий та професійний продукт для сучасної освіти.

2.2.5 Вибір оптимальних інструментів розробки

Після детального аналізу всіх варіантів технологічного інструментарія, варто підсумувати та зафіксувати фінальний склад стеку для розроблюваного проєкту. Хоча Python залишається чудовим інструментом для швидких експериментів у лабораторіях, для створення промислового API-модуля, який має стати частиною великих освітніх систем, ми обираємо платформу .NET. Цей вибір зумовлений не просто симпатією до технологій Microsoft, а прагматичним розрахунком: нам потрібна архітектура, яка не «впаде» під час сесії, забезпечить максимальний захист персональних даних та дозволить безшовно поєднати високу швидкість веб-сервісу з інтелектуальною обробкою тексту.

Основою нашої розробки стає мова C# та фреймворк ASP.NET Core, оскільки це – один із найшвидших веб-двигунів у світі, який дозволяє будувати API з мінімальними затримками. У системному проєктуванні важливо, щоб кожна мілісекунда була на рахунку, і .NET тут дає значну перевагу над інтерпретованим Python. Крім того, використання ML.NET дозволяє нам інтегрувати частину логіки машинного навчання безпосередньо в основний код, уникаючи зайвих «місточків» між різними мовами програмування, що робить систему простішою та надійнішою в підтримці.

Для розгортання та життя нашого проєкту ми робимо ставку на хмарну інфраструктуру Microsoft Azure. Це рішення закриває одразу кілька критичних питань: від автоматичного масштабування під час пікових навантажень до гарантованої безпеки корпоративного рівня. Інтеграція Azure DevOps дозволить налаштувати процес так, щоб кожне оновлення коду проходило через автоматичні

тести та миттєво потрапляло в робоче середовище без ризику щось зламати. Використання Docker-контейнерів у цій зв’язці робить наш модуль по-справжньому універсальним – дає можливість розгорнути його в будь-якій хмарі, але саме в Azure він отримає максимальну продуктивність завдяки нативній підтримці сервісів.

Зберігання даних буде делеговано PostgreSQL – надійному та перевіреному часом базі даних, яка ідеально працює в парі з Entity Framework Core. Такий тандем дозволяє нам не просто зберігати конспекти та тести, а й будувати складні семантичні зв’язки між ними, не переймаючись за цілісність інформації. EF Core виступає в ролі інтелектуального прошарку, який бере на себе всю рутину роботи з базою, дозволяючи розробнику зосередитися на головному – алгоритмах генерації контенту.

Що стосується «мізків» нашої системи, то ми обираємо гібридний підхід. Хоча архітектура будується на .NET, для безпосередньої генерації тексту ми звертаємося до кращих зовнішніх API, таких як GPT-4 та Claude через їхні офіційні інтерфейси. Це дає нам можливість використовувати найпотужніші у світі мовні моделі, зберігаючи при цьому повний контроль над бізнес-логікою та контекстом через наш власний RAG-модуль. У результаті ми отримуємо стек, який поєднує в собі корпоративну стабільність Microsoft, гнучкість сучасних хмарних рішень та неймовірну потужність штучного інтелекту. Це не просто набір інструментів, а професійний фундамент для продукту, який готовий до реальної експлуатації в університетах та великих компаніях.

Вибір середовища розробки для проєкту такого рівня – це не просто питання особистого комфорту, а стратегічне рішення, що безпосередньо впливає на швидкість реалізації архітектурних задумів та якість кінцевого коду. Visual Studio залишається безумовним лідером для роботи зі стеком .NET, оскільки надає найбільш глибоку інтеграцію з хмарною інфраструктурою Azure та нативну підтримку ML.NET. Наявність вбудованого інструменту Model Builder дозволяє нам візуалізувати процес навчання моделей та миттєво генерувати необхідний код на C#, що значно прискорює створення інтелектуального ядра нашої системи. Для

дослідження критично важливими є також потужні засоби діагностики та профілювання пам’яті.

VS Code – надзвичайно легковаговий та гнучкий редактор, який ми розглядаємо як допоміжний інструмент. Його головна перевага полягає в універсальності: він ідеально підходить для написання легких скриптів на Python, які ми використовуємо для порівняльного аналізу, або для швидкого редагування JSON-конфігурацій та фронтенд-складових. Однак для повноцінного проєктування складного API–модуля з розгалуженою логікою RAG-конвеєрів VS Code може виявитися недостатньо потужним, оскільки потребує великої кількості сторонніх плагінів, що іноді негативно впливає на стабільність робочого процесу.

JetBrains Rider стає «золотою серединою» та серйозним конкурентом для Visual Studio, особливо завдяки своїй високій швидкості роботи та інтелектуальним інструментам рефакторингу. Його кросплатформенна природа дозволяє забезпечити ідентичний досвід розробки на різних операційних системах, що є важливим при роботі в гетерогенних середовищах. Rider славиться своєю здатністю знаходити потенційні архітектурні вразливості ще на етапі написання коду, що допомагає нам підтримувати високу якість API та уникати технічного боргу. У контексті нашого проєкту Rider виступає як чудовий інструмент для швидкої ітеративної розробки, де продуктивність програміста має першочергове значення.

Для реалізації нашої системи ми обираємо Visual Studio як основну IDE через її безшовну синергію з Microsoft Azure та найбільш зрілу підтримку інструментів машинного навчання.

Swagger UI виступає критично важливим інструментом для візуалізації та інтерактивного тестування програмних інтерфейсів, перетворюючи технічну специфікацію OpenAPI на живу, зрозумілу документацію. У середовищі .NET він інтегрується переважно через бібліотеку Swashbuckle, автоматично генеруючи вебінтерфейс, який детально відображає всі доступні кінцеві точки, вимоги до вхідних даних та формати відповідей сервера. Нам як розробникам це дозволяє не просто описувати методи, а й миттєво перевіряти їхню працездатність безпосередньо з браузера, що значно спрощує налагодження складних ланцюжків

обміну інформацією між основним сервісом та інтелектуальними модулями обробки тексту.

Для нашого проєкту Swagger UI стає ключовою ланкою в забезпеченні прозорості та легкості інтеграції з освітніми платформами. Оскільки ми розробляємо API-модуль, що працює з асинхронними запитами до мовних моделей та RAG-конвеєрів, наявність структурованої «пісочниці» дозволяє розробникам навчальних систем швидко зрозуміти логіку взаємодії із сервісом без вивчення сотень сторінок тексту.

2.3 Методи машинного навчання та їх реалізація, метрики оцінки

Оцінка моделі з використанням багатокласової класифікації[9]. Алгоритм SDCA поєднує в собі кілька найкращих властивостей та можливостей алгоритмів логістичної регресії та SVM і, в більшості випадків, дуже добре підходить для багатокласових задач. Однак, як визначити, чи достатньо багатокласова модель для даної задачі? Давайте розглянемо деякі метрики:

```
// Train the model
ITransformer model = pipeline.Fit(trainingDataSet);

// Grab some metrics
var testMetrics = mlContext
.MulticlassClassification
.Evaluate(model.Transform(testDataSet));
```

Метод Evaluate повертає об'єкт MulticlassClassificationMetrics. У таблиці 2.3 наведено деякі з найчастіше використовуваних метрик, що входять у звіти цього об'єкту. У межах API-модуля для LMS методи машинного навчання використовуються не як окрема самоціль, а як прикладний інструмент для аналізу навчальних матеріалів, класифікації контенту, пошуку релевантних фрагментів, оцінювання студентських робіт і контролю якості згенерованих результатів.

З використанням ML.NET, у нас є доступ до бібліотек Microsoft.ML.Data і Microsoft.ML.

Таблиця 2.3 – Деякі метрики для багатокласової класифікації

Властивість	Опис
ConfusionMatrix	Повертає матрицю плутанини до класифікатора
LogLoss	Вказує середнє значення логарифмічних втрат, розрахованих для кожного класу
LogLossReduction	Вказує відсоток переваги, яку класифікатор надає порівняно з випадковим прогнозом
MacroAccuracy	Вказує середнє значення балу F1, розрахованого для кожного класу.
MicroAccuracy	Вказує F1-оцінку для всіх прогнозів, зроблених моделлю
PerClassLogLoss	Отримує логарифмічне значення втрат класифікатора для кожного класу.

Класифікація використовується для автоматичного визначення типу навчального матеріалу: лекція, практична робота, тест, студентська відповідь або пояснення.

Метрики класифікації:

– MicroAccuracy показує загальну частку правильних прогнозів;

– MacroAccuracy враховує якість класифікації по кожному класу окремо, що корисно при незбалансованих даних;

– LogLoss показує штраф за невпевнені або неправильні прогнози; чим менше значення, тим краща модель.

У межах LMS це дозволяє автоматично маршрутизувати матеріали до відповідних обробників API:

```
public class LearningText
{
    public string Text { get; set; } = "";
    public string Category { get; set; } = "";
}

public class TextPrediction
{
    [ColumnName("PredictedLabel")]
    public string Category { get; set; } = "";
}

var ml = new MLContext();

var data = ml.Data.LoadFromEnumerable(new[]
{
```

```
        new LearningText { Text = "Лекція про RAG та LLM", Category =
"Lecture" },
        new LearningText { Text = "Реалізувати API endpoint", Category =
"Assignment" },
        new LearningText { Text = "Оберіть правильну відповідь", Category
= "Test" }
    });

    var split = ml.Data.TrainTestSplit(data, testFraction: 0.3);

    var pipeline = ml.Transforms.Conversion.MapValueToKey("Label",
"Category")
        .Append(ml.Transforms.Text.FeaturizeText("Features", "Text"))

.Append(ml.MulticlassClassification.Trainers.SdcaMaximumEntropy())

.Append(ml.Transforms.Conversion.MapKeyToValue("PredictedLabel"));

    var model = pipeline.Fit(split.TrainSet);
    var predictions = model.Transform(split.TestSet);

    var metrics = ml.MulticlassClassification.Evaluate(predictions);

    Console.WriteLine($"MicroAccuracy: {metrics.MicroAccuracy:0.###}");
    Console.WriteLine($"MacroAccuracy: {metrics.MacroAccuracy:0.###}");
    Console.WriteLine($"LogLoss: {metrics.LogLoss:0.###}");
```

Регресійна модель може прогнозувати умовний рівень складності лекції або тестового питання за числовими ознаками: довжина тексту, кількість термінів, середня довжина речень, кількість формул тощо.

Метрики:

– MAE показує середню абсолютну помилку прогнозу.

– RMSE сильніше штрафує великі помилки.

– R2 показує, наскільки добре модель пояснює зміну цільового показника; ближче до 1 означає кращу якість.

Прогнозування умовного рівня складності через регресію:

```
public class MaterialFeatures
{
    public float WordCount { get; set; }
    public float TermCount { get; set; }
    public float AvgSentenceLength { get; set; }

    [ColumnName("Label")]
    public float Difficulty { get; set; }
```

```
}

public class DifficultyPrediction
{
    public float Score { get; set; }
}

var data = ml.Data.LoadFromEnumerable(new[]
{
    new MaterialFeatures { WordCount = 800, TermCount = 30,
AvgSentenceLength = 18, Difficulty = 3 },
    new MaterialFeatures { WordCount = 1500, TermCount = 80,
AvgSentenceLength = 25, Difficulty = 5 },
    new MaterialFeatures { WordCount = 400, TermCount = 10,
AvgSentenceLength = 12, Difficulty = 1 }
});

var split = ml.Data.TrainTestSplit(data, testFraction: 0.3);

var pipeline = ml.Transforms.Concatenate(
        "Features",
        nameof(MaterialFeatures.WordCount),
        nameof(MaterialFeatures.TermCount),
        nameof(MaterialFeatures.AvgSentenceLength))
    .Append(ml.Regression.Trainers.Sdca());

var model = pipeline.Fit(split.TrainSet);
var predictions = model.Transform(split.TestSet);

var metrics = ml.Regression.Evaluate(predictions);

Console.WriteLine($"MAE: {metrics.MeanAbsoluteError:0.###}");
Console.WriteLine($"RMSE: {metrics.RootMeanSquaredError:0.###}");
Console.WriteLine($"R2: {metrics.RSquared:0.###}");
```

Кластеризація дозволяє автоматично групувати фрагменти лекцій за темами. Це корисно для побудови структури курсу, формування тематичних блоків і підготовки матеріалів для RAG-конвеєра.

Метрики:

– AverageDistance показує середню відстань об’єктів до центрів кластерів.

– DaviesBouldinIndex оцінює компактність і роздільність кластерів; менше значення зазвичай краще.

Автоматичне групування фрагментів лекцій за темами через кластеризацію:

```
public class TextFragment
{
    public string Text { get; set; } = "";
}

var data = ml.Data.LoadFromEnumerable(new[]
{
    new TextFragment { Text = "LLM генерує відповідь на основі
контексту" },
    new TextFragment { Text = "RAG виконує пошук релевантних
фрагментів" },
    new TextFragment { Text = "Unit-тести перевіряють коректність
коду" }
});

var pipeline = ml.Transforms.Text.FeaturizeText("Features", "Text")
    .Append(ml.Clustering.Trainers.KMeans(
        featureColumnName: "Features",
        numberOfClusters: 2));

var model = pipeline.Fit(data);
var predictions = model.Transform(data);

var metrics = ml.Clustering.Evaluate(predictions);

Console.WriteLine($"AverageDistance:
{metrics.AverageDistance:0.###}");
Console.WriteLine($"DaviesBouldinIndex:
{metrics.DaviesBouldinIndex:0.###}");
```

У RAG-конвеєрі ключовим етапом є пошук релевантних фрагментів лекції. Для цього текст перетворюється у векторне представлення, а близькість між запитом і фрагментом оцінюється через косинусну схожість.

У практичній реалізації ці вектори можуть створюватися embedding-моделлю, після чого зберігатися у векторній базі даних. Чим ближче значення косинусної схожості до 1, тим більш релевантним є фрагмент до запиту користувача:

```
static double CosineSimilarity(float[] a, float[] b)
{
```

```
        double dot = 0, normA = 0, normB = 0;

        for (int i = 0; i < a.Length; i++)
        {
            dot += a[i] * b[i];
            normA += a[i] * a[i];
            normB += b[i] * b[i];
        }

        return dot / (Math.Sqrt(normA) * Math.Sqrt(normB));
    }

    var queryVector = new float[] { 0.12f, 0.45f, 0.31f };
    var fragmentVector = new float[] { 0.10f, 0.40f, 0.35f };

    var similarity = CosineSimilarity(queryVector, fragmentVector);

    Console.WriteLine($"Similarity: {similarity:0.###}");
```

Для оцінки якості пошуку в RAG-конвеєрі можна використовувати Precision@K. Вона показує, яка частка з перших K знайдених фрагментів справді є релевантною.

Якщо Precision@3 = 0.667, це означає, що серед трьох перших знайдених фрагментів два були релевантними. Для LMS це важливо, бо якість відповіді LLM напряму залежить від якості знайденого контексту:

```
    static double PrecisionAtK(bool[] relevance, int k)
    {
        return relevance.Take(k).Count(x => x) / (double)k;
    }

    var topResults = new[] { true, true, false, true, false };

    Console.WriteLine($"Precision@3: {PrecisionAtK(topResults,
3):0.###}");
```

Для оцінювання згенерованого конспекту можна порівнювати його з еталонним конспектом викладача. ROUGE-L базується на найдовшій спільній послідовності слів:

```
    static int LcsLength(string[] a, string[] b)
    {
```

```
        var dp = new int[a.Length + 1, b.Length + 1];

        for (int i = 1; i <= a.Length; i++)
            for (int j = 1; j <= b.Length; j++)
                dp[i, j] = a[i - 1] == b[j - 1]
                    ? dp[i - 1, j - 1] + 1
                    : Math.Max(dp[i - 1, j], dp[i, j - 1]);

        return dp[a.Length, b.Length];
    }

    static double RougeL(string generated, string reference)
    {
        var gen = generated.Split(' ',
StringSplitOptions.RemoveEmptyEntries);
        var refText = reference.Split(' ',
StringSplitOptions.RemoveEmptyEntries);

        var lcs = LcsLength(gen, refText);

        return lcs / (double)refText.Length;
    }

    var score = RougeL(
        "RAG поєднує пошук та генерацію відповіді",
        "RAG використовує пошук і генерацію для відповіді");

    Console.WriteLine($"ROUGE-L: {score:0.###}");
```

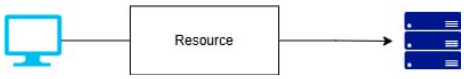
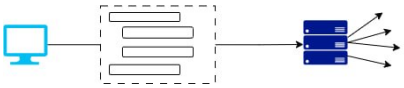
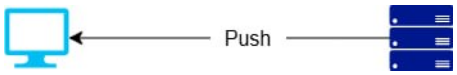
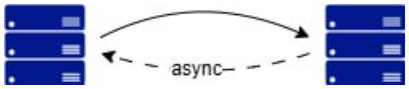
ROUGE-L ближче до 1 означає, що згенерований конспект краще покриває зміст еталонного тексту. Проте для нашого проєкту доцільно поєднувати автоматичні метрики з експертною оцінкою викладача.

2.4 Проєктування архітектури API-модуля

Для розроблюваного API-модуля доцільно використати модульну сервісно-орієнтовану архітектуру. Такий підхід передбачає, що інтелектуальні функції системи не вбудовуються безпосередньо в код системи, а реалізуються як окремий незалежний сервіс, з яким LMS взаємодіє через REST API. Це дозволяє інтегрувати модуль у різні освітні платформи без повної зміни їхньої внутрішньої структури.

У таблиці 2.4 зображені деякі з класичних типів архітектур, які використовуються у розробці повноцінних великих проєктів систем управління навчанням та систем інших типів, а також прикладних інтерфейсів програмування, яким і є наш розроблюваний проєкт.

Таблиця 2.4 – Класичні типи архітектур

Тип архітектури	Зображення	Використання
SOAP		Стандарт на основі протоколу, що використовує XML, з такими функціями, як безпека та транзакції
RESTful		Використовує стандартні методи HTTP, ключовими обмеженнями є відсутність стану, кешування та єдиний інтерфейс.
GraphQL		Дозволяє клієнтам запитувати лише ті дані, які їм потрібні, пропонуючи гнучке та ефективне отримання даних
gRPC		Високопродуктивний фреймворк RPC від Google, що використовує буфери протоколів для ефективної серіалізації
WebSocket		Забезпечує повнодуплексний зв'язок у режимі реального часу через одне з'єднання
Webhook		Асинхронний для подієво-керованих застосунків

Обрана архітектура будується на принципах API-first, що забезпечує доступ до всіх функціональних можливостей модуля через суворо регламентовані запити. В основі лежить багатошарова структура (Layered architecture), де чітко розмежовано рівні контролерів, бізнес-логіки, доступу до даних та безпосередньої інтеграції із зовнішніми мовними моделями. Такий підхід органічно доповнюється сервіс-орієнтованою моделлю, у межах якої кожен функціональний блок функціонує як незалежний сервіс, що значно спрощує подальше масштабування та технічну підтримку системи.

Для забезпечення стабільної роботи при виконанні ресурсомістких завдань, таких як семантичний аналіз PDF-документів чи автоматизоване формування тестів, у системі реалізовано подієво-орієнтовану модель з асинхронною обробкою запитів у фоновому режимі. Ключовим інтелектуальним елементом архітектури є впровадження конвеєрів на основі RAG. Це дозволяє моделі формувати відповіді не лише на базі загальних знань, закладених у промпт, а й з урахуванням специфічного контексту, оперативно видобутого з конкретних фрагментів навчальних матеріалів, що гарантує високу точність та релевантність генерованого контенту.

LMS виконує роль зовнішньої системи, яка надсилає навчальні матеріали, запити студентів або здані роботи. API-модуль приймає ці дані, обробляє їх і повертає результат: конспект, відповідь ШІ-помічника, набір тестових питань або звіт про студентську роботу.

Важливою особливістю є асинхронна обробка. Операції з великими файлами та LLM можуть тривати кілька секунд або хвилин, тому система не повинна блокувати користувача. Замість цього API приймає завдання, створює запис про нього в базі даних, ставить його в чергу та повертає ідентифікатор задачі. Після завершення обробки результат можна отримати через окремий endpoint. Наприклад:

```
POST /api/materials/upload
GET  /api/tasks/{taskId}
GET  /api/materials/{id}/summary
POST /api/assistant/ask
POST /api/tests/generate
POST /api/submissions/analyze
```

2.5 Вимоги до функціоналу ML-моделей і LLM у проєкті

LLM та ML-моделі складають інтелектуальне ядро системи, забезпечуючи повний цикл обробки навчальних матеріалів – від витягу тексту з файлу поширених форматів та його очищення до глибокої семантичної сегментації. Кожен фрагмент

зберігає детальні метадані (тема, сторінка), що є критичним для роботи RAG-конвеєра та забезпечення прозорості джерел при формуванні відповідей.

Функціонал системи охоплює автоматичне створення конспектів, структурування планів занять та генерацію різнопланових тестів із семантично близькими дистракторами. Завдяки RAG-підходу відповіді на запитання студентів базуються виключно на верифікованих даних курсу, що практично виключає «галюцинації» ШІ. Крім того, модулі аналізують студентські роботи, надаючи розгорнутий фідбек щодо повноти текстових відповідей та логіки програмного коду.

Якість генерації валідується метриками ROUGE, BLEU та Precision@K, а технічна стабільність забезпечується асинхронною моделлю обробки фонових завдань через систему ідентифікаторів. Безпека даних гарантується розмежуванням ролей та ізоляцією контексту запитів. У підсумку, синергія ML, LLM та RAG перетворює API-модуль на високоефективне інтелектуальне розширення для сучасних LMS, здатне перебрати на себе основний тягар методичної роботи.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО API-МОДУЛЯ

3.1 Реалізація архітектури API та можливості масштабування

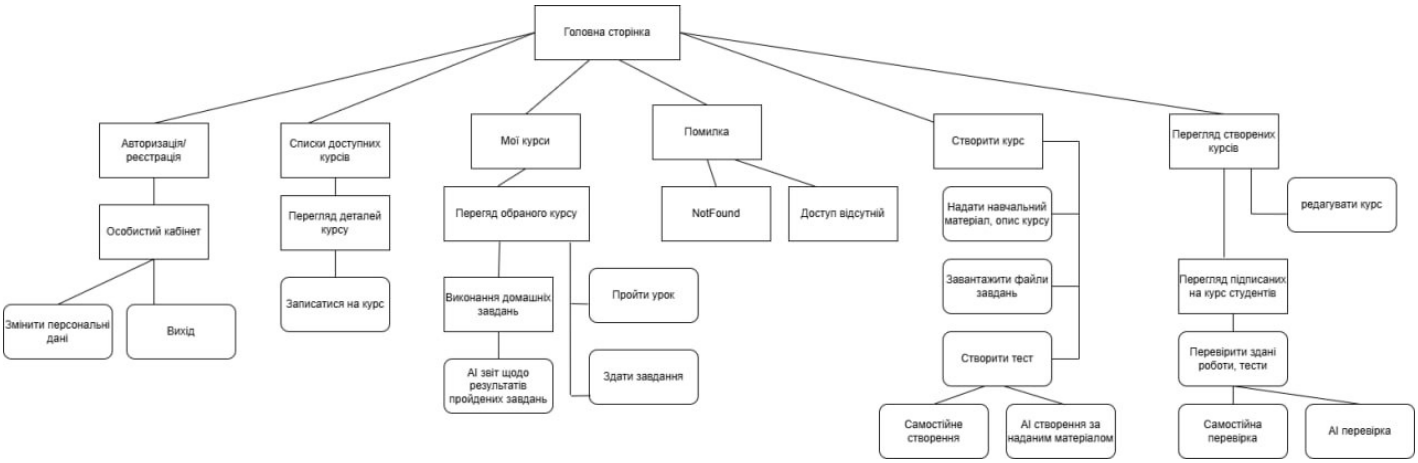


Рисунок 3.1 – Карта веб-сторінок потенційної системи керування навчанням з інтегрованим інтелектуальним модулем

Реалізація API-модуля для інтеграції LLM у LMS передбачає поступову розбудову системи від базового прототипу до масштабованого сервісу, здатного обробляти значну кількість навчальних матеріалів і запитів користувачів. LMS не повинна виконувати ресурсомісткі операції самостійно, а передати їх окремому інтелектуальному API-модулю. Загальну схему архітектури подано на рисунку 3.2

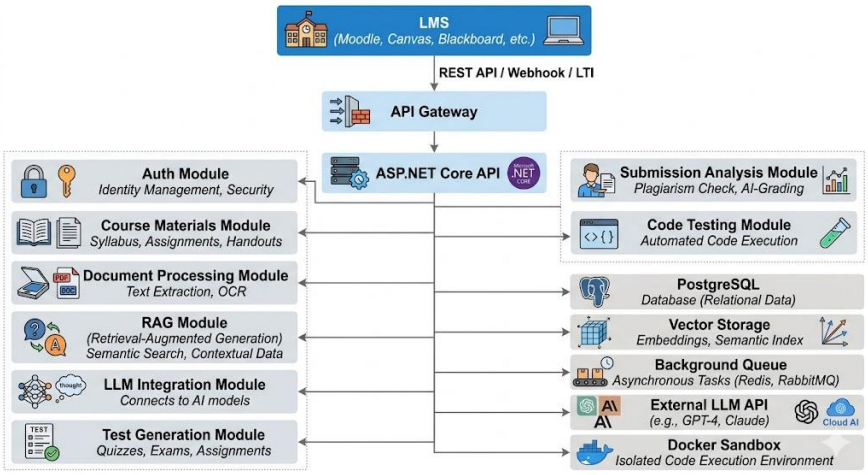


Рисунок 3.2 – Схематична архітектура проєкту

Реалізація розпочинається з розгортання ASP.NET Core Web API, що визначає контракт взаємодії з LMS через ендпоінти завантаження та генерації контенту. Паралельно впроваджується модуль обробки матеріалів, який видобуває текст із файлів та сегментує його на фрагменти зі збереженням метаданих, створюючи надійний фундамент для контекстної роботи системи.

Наступним кроком є підключення RAG-конвеєра, де текстові сегменти конвертуються у векторні представлення для семантичного пошуку, що критично знижує ризик «галюцинацій» ШІ. Взаємодія з LLM виноситься в окремий сервіс формування промптів, що дозволяє гнучко змінювати постачальників мовних моделей без переписування основної бізнес-логіки.

П’ятий етап передбачає перехід до асинхронної обробки задач. Ресурсомісткі операції, як-от індексація великих PDF-файлів чи створення тестів, виконуються у фоновому режимі, а користувач отримує ідентифікатор для відстеження статусу. Такий підхід гарантує стабільність API та високу швидкість відгуку системи незалежно від обсягу даних.

Приклад логіки взаємодії:

а) надсилання LMS файлу лекції до системи;

б) створення API відповідної задачі обробки;

в) отримання користувачем унікального ідентифікатора taskId;

г) вилучення тексту та генерація embeddings фоновим сервісом;

д) збереження фінального результату в базі даних;

е) отримання LMS готового конспекту або тесту за ідентифікатором taskId.

Шостим етапом є реалізація модуля аналізу студентських робіт. Для текстових робіт система може перевіряти відповідність темі, повноту відповіді, наявність ключових понять і логічність викладу. Для програмних робіт доцільно використовувати ізольоване середовище запуску, наприклад Docker-контейнер. У ньому студентський код перевіряється автоматичними тестами, після чого LLM може сформувати зрозуміле пояснення результатів для студента або викладача.

Окрему увагу потрібно приділити масштабуванню. На початковому етапі всі модулі можуть бути реалізовані в межах одного ASP.NET Core-проєкту як

модульний моноліт. Це спрощує розробку, тестування і налагодження. Проте внутрішня структура має бути побудована так, щоб окремі частини системи можна було винести в незалежні сервіси. Поступове масштабування може виглядати як у таблиці 3.1.

Таблиця 3.1 – Потенційні етапи масштабування проєкту

Етап	Назва етапу	Опис
1	Модульний моноліт	API, бізнес-логіка, робота з БД і LLM-інтеграція в одному застосунку
2	Додавання фонових задач	Винесення довгих операцій у background worker
3	Окреме векторне сховище	Підключення pgvector, Qdrant або іншої vector database
4	Винесення LLM/RAG в окремий сервіс	Окремий сервіс для embeddings, пошуку і формування промптів
5	Горизонтальне масштабування	Запуск кількох екземплярів API та worker-сервісів

3.2 Порівняльний експеримент ML.NET проти Python

Для обґрунтування вибору технологічного стеку в межах розроблюваного API-модуля було проведено порівняльний аналіз двох підходів до реалізації інтелектуальної обробки тексту: використання ML.NET у середовищі C# та застосування Python-бібліотек, зокрема scikit-learn, NLTK і spaCy. Такий експеримент є важливим для проєкту, оскільки API-модуль повинен не лише інтегруватися з LLM, а й виконувати попередню обробку навчальних матеріалів, семантичне структурування текстів, оцінювання якості згенерованого контенту та підготовку даних для RAG-конвеєра.

Python традиційно вважається одним із найпоширеніших інструментів для задач машинного навчання та NLP завдяки великій кількості бібліотек і гнучкості експериментування. Водночас для розроблюваного проєкту важливими є не лише дослідницькі можливості, а й стабільність, інтеграція з ASP.NET Core API, підтримка типізації, зручність розгортання та передбачуване використання ресурсів. Саме тому ML.NET розглядається як потенційно ефективніший варіант для промислової реалізації модуля в екосистемі .NET.

Основою експерименту став метод TF-IDF [20, 21], який використовується для визначення важливості термінів у документі відносно всього корпусу текстів. У контексті LMS цей метод може застосовуватися для виділення ключових понять лекції, попередньої класифікації матеріалів, тематичного пошуку та побудови інтерпретованих ознак перед подальшою обробкою LLM або RAG-модулем.

У ML.NET реалізація TF-IDF може виконуватися через вбудований трансформер FeaturizeText, який перетворює текст у числове представлення в межах єдиного конвеєра обробки даних:

```
var ml = new MLContext();
var data = ml.Data.LoadFromEnumerable(documents);

var pipeline = ml.Transforms.Text.FeaturizeText(
    outputColumnName: "Features",
    inputColumnName: "Text");

var model = pipeline.Fit(data);
var transformedData = model.Transform(data);
```

У Python аналогічний підхід найчастіше реалізується через TfidfVectorizer з бібліотеки scikit-learn:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
```

```
tfidf_matrix = vectorizer.fit_transform(documents)
```

Обидва підходи дозволяють отримати векторне представлення текстів, однак відрізняються за рівнем контролю, способом інтеграції та експлуатаційними характеристиками. Python надає більшу гнучкість у налаштуванні параметрів токенізації, n-грам, стоп-слів, лематизації та додаткової NLP-обробки [23, 24]. ML.NET, зі свого боку, забезпечує зручну інтеграцію з C#, ASP.NET Core, Entity Framework Core та іншими компонентами .NET-інфраструктури.

Для більш повної оцінки текстових результатів у другому експериментальному напрямі було використано набір лінгвістичних метрик: коефіцієнт водянистості тексту, Type-Token Ratio, MATTR, MTLD, Factual Accuracy Score, Manipulation Score та Coherence Score. Ці показники дозволяють оцінювати не лише швидкість обробки тексту, а й якість самого контенту: надмірність, лексичне багатство, зв’язність, фактичну достовірність і потенційну маніпулятивність.

Експеримент виконувався в однакових умовах для обох середовищ. Для коректності порівняння використовувався один і той самий корпус текстів, однакова логіка попередньої обробки та однакові набори метрик. Порівняння проводилося за такими критеріями:

- час виконання обробки документа;
- використання оперативної пам’яті;
- навантаження на CPU;
- стабільність результатів;
- зручність інтеграції в API;
- придатність до масштабування;
- гнучкість налаштування NLP-конвеєра.

Результати експерименту показали, що ML.NET демонструє більш стабільну поведінку під час виконання типових задач текстової аналітики. У дослідженні з TF-IDF середній час обробки в ML.NET становив приблизно 150 мс на документ, тоді як Python-реалізація на scikit-learn потребувала від 1,6 до 3,2 с залежно від

документа та складності обробки. Також ML.NET показав нижче споживання оперативної пам’яті: близько 10 МБ проти приблизно 97 МБ у Python-середовищі. За показником використання CPU ML.NET також виявився більш передбачуваним, перебуваючи приблизно в межах 4-13 %, тоді як Python демонстрував вищу варіативність навантаження.

Отримані результати свідчать, що ML.NET краще відповідає вимогам систем, які мають працювати як частина стабільного серверного API. Для розроблюваного модуля це є суттєвою перевагою, оскільки обробка навчальних матеріалів повинна виконуватися не як окремий дослідницький скрипт, а як частина повноцінного backend-сервісу, здатного приймати запити від LMS, обробляти їх асинхронно та повертати результат у стандартизованому форматі.

Водночас Python має важливі переваги на етапі дослідження та прототипування. Його екосистема містить велику кількість готових інструментів для NLP, глибокого навчання, лематизації, токенізації, роботи з трансформерами та візуалізації результатів. Тому Python доцільно використовувати для експериментальної перевірки нових моделей, підбору параметрів, швидкого тестування гіпотез або роботи з бібліотеками, які ще не мають повноцінних аналогів у .NET. У таблиці 3.2 – стисло узагальнений результат порівняльного експерименту.

Для цього проєкту найбільш доцільним є гібридний підхід. Основний API-модуль варто реалізовувати на базі ASP.NET Core та ML.NET, оскільки це забезпечує стабільність, типобезпечність, зручність підтримки та просту інтеграцію з LMS. Python при цьому може використовуватися як допоміжне середовище для дослідження, підготовки експериментальних моделей або перевірки окремих NLP-алгоритмів.

З практичної точки зору ML.NET [29] краще підходить для функцій, які мають бути вбудовані безпосередньо в серверну логіку модуля: класифікація навчальних матеріалів, TF-IDF-аналіз, базова оцінка тексту, попередня фільтрація та підготовка даних до RAG-конвеєра. Python доцільніше застосовувати там, де потрібна максимальна гнучкість: складна лінгвістична обробка, експерименти з

трансформерними моделями, тонке налаштування NLP-пайплайнів або порівняння альтернативних алгоритмів.

Таблиця 3.2 – Порівняльна характеристика двох підходів

Критерій	ML.NET	Python
Інтеграція з API	Висока, нативна для .NET	Потребує окремого сервісу
Продуктивність	Стабільна	Залежить від бібліотек
Пам’ять	Нижче споживання	Вище споживання
Гнучкість NLP	Помірна	Дуже висока
Підтримка прототипування	Обмеженіша	Дуже зручна
Розгортання	Простіше в .NET-системах	Часто потребує FastAPI/Flask
Підтримка LLM/RAG	Можлива через API/SDK	Широка екосистема бібліотек
Придатність до LMS API	Висока	Середня або висока як окремий сервіс

Проведений порівняльний експеримент підтвердив доцільність використання ML.NET [30] як основного інструменту для реалізації текстової аналітики в межах розроблюваного API-модуля. Його переваги полягають у стабільній продуктивності, меншому споживанні ресурсів, простішому розгортанні та природній інтеграції з .NET-інфраструктурою. Python зберігає значення як дослідницький і допоміжний інструмент, однак для промислової реалізації LMS-орієнтованого API [27, 28] перевага надається ML.NET[30, 32]. Саме такий підхід дозволяє поєднати інтерпретовані методи аналізу тексту, метрики якості навчального контенту та масштабовану серверну архітектуру в межах єдиної системи.

3.3 Тестування та оптимізація роботи модуля

Тестування API-модуля є важливим етапом практичної реалізації, оскільки система працює з різними типами запитів: завантаженням навчальних матеріалів, генерацією конспектів, відповідями ШІ-помічника, створенням тестів та аналізом студентських робіт. Для такого проєкту недостатньо перевіряти лише коректність окремих методів. Необхідно протестувати повний цикл роботи: від HTTP-запиту до обробки даних, взаємодії з LLM [32], збереження результату та повернення відповіді користувачу LMS.

У межах проєкту доцільно використовувати кілька рівнів тестування: ручне API-тестування через Swagger UI, автоматизоване тестування через Postman/Newman, інтеграційні тести на рівні ASP.NET Core, а також окреме тестування модуля перевірки студентського коду.

Swagger UI використовується на етапі розробки як інтерактивна документація API. Він дозволяє бачити всі доступні endpoint-и, структуру вхідних даних, можливі відповіді сервера та швидко виконувати запити без написання окремого клієнтського застосунку. Для API-модуля це особливо зручно, оскільки викладач або розробник LMS може перевірити роботу методів завантаження лекцій, генерації конспектів і звернення до ШІ-помічника без додаткових інструментів.

Swagger UI більше підходить для ручної перевірки та демонстрації API, тоді як Postman дозволяє створювати повноцінні набори тестів. У Postman кожен запит може мати тестовий сценарій, який перевіряє код відповіді, структуру JSON, наявність ідентифікатора задачі або коректність повідомлення про помилку.

Приклад тесту в Postman для ендпоінта генерації конспекту:

```
pm.test("Status code is 202", function () {
    pm.response.to.have.status(202);
});

pm.test("Response contains taskId", function () {
```

```
    const json = pm.response.json();
    pm.expect(json.taskId).to.exist;
});
```

Окрім зовнішніх API-тестів, важливо реалізувати інтеграційні тести безпосередньо в .NET-проєкті [26]. Для цього можна використати xUnit та Microsoft.AspNetCore.Mvc.Testing. Такі тести запускають API у тестовому середовищі та перевіряють поведінку ендпоінтів через HttpClient:

```
public class AssistantApiTests
    : IClassFixture<WebApplicationFactory<Program>>
{
    private readonly HttpClient _client;

    public AssistantApiTests(WebApplicationFactory<Program> factory)
    {
        _client = factory.CreateClient();
    }

    [Fact]
    public async Task Ask_ReturnsSuccessfulResponse()
    {
        var request = new
        {
            courseId = "ai-course",
            question = "Що таке RAG?"
        };

        var response = await _client.PostAsJsonAsync(
            "/api/assistant/ask",
            request);

        response.EnsureSuccessStatusCode();

        var json = await
response.Content.ReadFromJsonAsync<AssistantResponse>();

        Assert.NotNull(json);
        Assert.False(string.IsNullOrWhiteSpace(json.Answer));
    }
}
```

Оптимізація роботи модуля потрібна через те, що LLM-запити, обробка PDF і семантичний пошук можуть бути ресурсомісткими. Першим напрямом оптимізації є асинхронна обробка. Замість того щоб чекати завершення генерації в межах одного HTTP-запиту, API створює задачу, повертає taskId, а сама обробка виконується у фоновому сервісі:

```
    [HttpPost("summary")]
    public IActionResult GenerateSummary([FromBody] SummaryRequest request)
    {
        var taskId = Guid.NewGuid().ToString();

        _backgroundQueue.Enqueue(new SummaryJob
        {
            TaskId = taskId,
            MaterialId = request.MaterialId
        });

        return Accepted(new { taskId });
    }
```

Другим напрямом є кешування результатів. Якщо користувач повторно ставить однакове питання до того самого матеріалу, система може повернути вже обчислений результат або повторно використати знайдені RAG-фрагменти. Це зменшує кількість звернень до LLM і вартість роботи системи.

```
    public async Task<string> GetAnswerAsync(string question, string courseId)
{
    string cacheKey = $"answer:{courseId}:{question}";

    if (_cache.TryGetValue(cacheKey, out string cachedAnswer))
```

```
        return cachedAnswer;

    var answer = await _llmService.GenerateAnswerAsync(question, courseId);

    _cache.Set(cacheKey, answer, TimeSpan.FromMinutes(30));

    return answer;
}
```

Третім напрямом є обмеження кількості запитів. Оскільки LLM-запити можуть бути дорогими, API повинен захищати систему від надмірного навантаження. Для цього використовується rate limiting.

Ще одним методом оптимізації є вимірювання продуктивності окремих частин системи. Наприклад, можна порівняти швидкість розбиття тексту на фрагменти, пошуку релевантних embeddings або генерації промпту. Для цього доцільно використовувати BenchmarkDotNet.

Такі вимірювання дозволяють не оптимізувати систему навмання, а знаходити конкретні вузькі місця: надто повільне читання файлів, зайве повторне створення embeddings, неефективні запити до бази даних або надмірний розмір контексту, який передається до LLM.

Таким чином, тестування та оптимізація API-модуля мають охоплювати як функціональну коректність, так і стабільність під навантаженням. Swagger UI забезпечує зручну ручну перевірку й документування API, Postman дозволяє автоматизувати сценарії тестування, xUnit використовується для інтеграційних тестів у .NET. Оптимізація досягається за рахунок асинхронної обробки, кешування, обмеження частоти запитів і вимірювання продуктивності критичних частин системи. У сукупності це робить модуль придатним не лише для демонстраційного прототипу, а й для подальшого масштабування в реальному освітньому середовищі.

3.4 Економічне обґрунтування впровадження інтелектуального модуля

Економічне обґрунтування впровадження інтелектуального API-модуля полягає у визначенні доцільності його використання в освітньому середовищі з погляду витрат на розробку, експлуатацію та очікуваного ефекту для закладу освіти. Основна цінність запропонованого рішення полягає не лише в автоматизації окремих операцій, а й у зменшенні трудомісткості рутинної роботи викладача: підготовки конспектів, створення тестів, первинного аналізу студентських робіт та формування відповідей на типові питання студентів.

У традиційній моделі значна частина часу викладача витрачається на повторювані дії. Наприклад, після підготовки лекційного матеріалу необхідно окремо створити короткий конспект, сформувати контрольні питання, підготувати тестові завдання, перевірити частину студентських робіт і надати пояснення студентам. Інтелектуальний модуль дозволяє автоматизувати частину цих процесів, залишаючи за викладачем функцію контролю, редагування та остаточного затвердження результатів.

До основних витрат на впровадження модуля належать витрати:

- на розробку програмного забезпечення;
- на серверну інфраструктуру;
- на використання LLM API або локальних моделей;
- на зберігання даних і векторної бази;
- на підтримку, оновлення та адміністрування системи.

На етапі прототипу система може бути реалізована як модульний ASP.NET Core API із PostgreSQL та векторним сховищем. У такому випадку витрати на інфраструктуру залишаються помірними, оскільки всі компоненти можуть працювати на одному сервері або в хмарному середовищі мінімальної конфігурації. Надалі, зі збільшенням навантаження, окремі частини системи можуть масштабуватися незалежно: API-сервер, фонові обробники документів, векторне сховище та модуль взаємодії з LLM.

Для оцінки економічного ефекту можна порівняти витрати часу викладача до та після впровадження модуля. Наприклад, якщо підготовка конспекту, тестових завдань та короткого опису лекції вручну займає в середньому 2 години, а після використання системи викладач витрачає 30 хвилин лише на перевірку та редагування результату, економія становить 1,5 години на одну лекцію.

Додатково можна оцінити економію часу на створенні тестів. Якщо ручна підготовка набору з 20 тестових питань займає приблизно 1–1,5 години, то за допомогою LLM система може сформувати первинний варіант тесту за кілька хвилин. Викладач у такому випадку не створює тест з нуля, а лише перевіряє коректність питань, редагує формулювання та затверджує фінальний варіант.

Окремим джерелом економічного ефекту є автоматизація первинного аналізу студентських робіт. Для текстових робіт модуль може визначати відповідність темі, наявність ключових понять і загальну структуру відповіді. Для програмних завдань система може запускати автоматичні тести та формувати звіт про результат. Це не замінює викладача повністю, але дозволяє швидше виявляти типові помилки та зменшити навантаження під час перевірки великої кількості робіт.

Витрати на експлуатацію модуля залежать від способу розгортання. Якщо використовується зовнішній LLM API, основною змінною витратою є кількість запитів і обсяг переданого тексту. Якщо використовується локальна модель, витрати зміщуються в бік серверного обладнання, GPU та технічної підтримки. Для магістерського проєкту доцільним є гібридний підхід: на етапі прототипу використовувати зовнішній API, а для подальшого розвитку передбачити можливість підключення локальної або корпоративної моделі.

Для зменшення експлуатаційних витрат у проєкті передбачено кілька оптимізаційних рішень. По-перше, використовується кешування результатів для повторюваних запитів. Якщо студент ставить типове питання до вже обробленої лекції, система може повторно використати готову відповідь або знайдені RAG-фрагменти. По-друге, застосовується асинхронна обробка, що дозволяє ефективніше використовувати серверні ресурси. По-третє, обмеження частоти

запитів запобігає надмірному використанню LLM і неконтрольованому зростанню витрат.

Економічний ефект від впровадження модуля проявляється не лише у прямій економії часу. Важливими є також непрямі переваги:

- підвищення швидкості підготовки навчальних матеріалів;
- зменшення навантаження на викладача;
- швидше надання зворотного зв’язку студентам;
- уніфікація якості тестових завдань;
- можливість повторного використання оброблених матеріалів;
- масштабування рішення на різні курси та освітні програми.

З погляду закладу освіти запропонований API-модуль є економічно доцільним, оскільки не потребує повної заміни наявної LMS. Він інтегрується як зовнішній сервіс і може бути підключений поступово: спочатку для генерації конспектів і тестів, потім для ШІ-помічника, а надалі для аналізу студентських робіт та автоматичного тестування програмного коду. Такий підхід зменшує ризики впровадження та дозволяє оцінювати ефективність системи поетапно. Отже, система дозволяє скоротити витрати часу на рутинні педагогічні процеси, підвищити швидкість підготовки навчального контенту та забезпечити масштабованість освітніх сервісів. Найбільший ефект досягається при використанні модуля на кількох курсах або кафедрах, де повторюваність задач є високою, а автоматизація навіть частини процесів дає помітну економію ресурсів.

ВИСНОВКИ

У цій роботі розв’язано актуальне науково-прикладне завдання, пов’язане з проєктуванням і реалізацією методів інтелектуальної обробки та автоматизованої генерації навчального контенту для сучасних LMS. Проведене дослідження показало, що освітні платформи поступово мають переходити від простого зберігання навчальних файлів до їх семантичної обробки, аналізу та повторного використання. На цій основі було обґрунтовано розробку API-модуля, призначеного для автоматизації рутинних методичних процесів і зменшення навантаження на викладача.

Порівняльний аналіз наявних LMS і освітніх сервісів засвідчив наявність розриву між традиційними платформами, орієнтованими переважно на розміщення матеріалів, та інтелектуальними рішеннями, які підтримують персоналізацію і автоматизовану взаємодію зі студентом. Це підтвердило доцільність створення зовнішнього незалежного модуля, який може інтегруватися з різними освітніми системами без глибокого втручання в їхню внутрішню архітектуру.

Використання екосистеми .NET, хмарних сервісів та інструментів інтеграції LLM дозволило спроєктувати рішення, придатне для стабільної роботи під навантаженням і подальшого масштабування. Основою запропонованого підходу є RAG-архітектура, що поєднує векторне представлення навчальних матеріалів, семантичний пошук і генеративні можливості великих мовних моделей. Завдяки цьому відповіді системи формуються не лише на основі загальних знань моделі, а з урахуванням перевірених матеріалів конкретного навчального курсу, що знижує ризик помилкових або нерелевантних результатів.

Розроблений модуль забезпечує асинхронну обробку навчальних матеріалів, зокрема вилучення тексту з PDF-документів, поділ його на логічні фрагменти, формування структурованих конспектів і генерацію тестових завдань. Окрему увагу приділено створенню якісних дистракторів, які є семантично близькими до правильної відповіді та дозволяють ефективніше перевіряти рівень засвоєння матеріалу. Також передбачено можливість аналізу студентських робіт і програмного

коду, що розширює функціональність модуля від генерації контенту до підтримки оцінювання.

Оцінювання результатів за допомогою метрик ROUGE, BLEU та Precision@K підтвердило доцільність використання запропонованого підходу для задач генерації навчального тексту й контекстного пошуку. Практичне значення роботи полягає у створенні універсального API-рішення, яке може скоротити час підготовки дидактичних матеріалів, підвищити швидкість надання зворотного зв’язку студентам і стати основою для подальшого розвитку адаптивних освітніх сервісів на базі LLM.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Miles R. C# Programming Yellow Book : Cheese Edition 8.1. Hull : University of Hull, 2019. 217 p.

2. Bayer T., Polley T. The API Gateway Handbook : Your Practical Guide to API Gateway Setup, Security, and Operation. Bonn : predic8 GmbH, 2025. 277 p.

3. Mertens O., Van Baelen B. Azure Data and AI Architect Handbook : Adopt a structured approach to designing data and AI solutions at scale on Microsoft Azure. Birmingham : Packt Publishing, 2023. 284 p.

4. Ahmed Z., Amizadeh S., Bilenko M. et al. Machine Learning at Microsoft with ML.NET. arXiv. 2019. URL: https://arxiv.org/pdf/1905.05715 (дата звернення: 22.03.2026). DOI: https://doi.org/10.1145/3292500.3330667.

5. Likness J. Microsoft Agent Framework – Building Blocks for AI Part 3. .NET Blog. 2026. URL: https://devblogs.microsoft.com/dotnet/microsoft-agent-framework-building-blocks-for-ai-part-3/ (дата звернення: 08.02.2026).

6. .NET Documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/ (дата звернення: 18.03.2026).

7. What is ML.NET and how does it work? Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/machine-learning/mldotnet-api (дата звернення: 23.03.2026).

8. Chettier T. M., Boyina V. A. K. Integrating Machine Learning with Fullstack Development Using ML.NET. International Journal of Computer Sciences and Engineering. 2025. Vol. 13, no. 1. P. 17–23.

9. Esposito D., Esposito F. Programming ML.NET. 1st ed. Redmond : Microsoft Press, 2022. 145-168 pp.

10. Martin R. C. Clean Code : A Handbook of Agile Software Craftsmanship. 1st ed. Boston : Pearson, 2008. 464 p.

11. Martin R. C. Clean Architecture : A Craftsman’s Guide to Software Structure and Design. 1st ed. Boston : Pearson, 2017. 432 p.

12. Esposito D. Programming ASP.NET Core. 1st ed. Redmond : Microsoft Press, 2018. 416 p.

13. Горбатенко М. С., Білова Т. Г. Критерії вибору архітектур веб-застосунків з високою доступністю. Радіоелектроніка та молодь у XXI столітті : матеріали 29-го Міжнар. молодіж. форуму, 16–19 квітня 2025 р. Харків : ХНУРЕ, 2025. Т. 6. С. 395–397

14. Harris C. Microservices vs. monolithic architecture. Atlassian. URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith (дата звернення: 15.02.2025).

15. Tymoshchuk T. Monolith vs. Microservices vs. Serverless architecture. Geniusee. 2023. URL: https://geniusee.com/single-blog/monolith-vs-microservices-vs-serverless-architecture (дата звернення: 04.04.2025).

16. Newman S. Monolith to Microservices : Evolutionary Patterns to Transform Your Monolith. Sebastopol : O'Reilly Media, 2019. 272 p.

17. Морозов В. Ю. Теорія штучного інтелекту. Харків : Ранок, 2021. 304 с.

18. Монахов В. М. Основи наукових досліджень : навч. посіб. Одеса : ОНУ, 2020. 186 с.

19. Text classification with ML.NET. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/machine-learning/how-to-guides/text-analysis-ml-net (дата звернення: 02.04.2026).

20. Langer S., Gipp B. TF-IDF : A novel term-weighting scheme for user modeling based on users' personal document collections. Proceedings of the iConference 2017. Wuhan, 2017. DOI: https://doi.org/10.9776/17306.

21. Understanding TF-IDF (Term Frequency-Inverse Document Frequency). GeeksforGeeks. URL: https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/ (дата звернення: 01.05.2026).

22. Goyal P., Pandey S., Jain K. Deep Learning for Natural Language Processing : Creating Neural Networks with Python. Berkeley : Apress, 2018. 277 p.

23. Top NLP Algorithms & Concepts. Data Science Central. URL: https://www.datasciencecentral.com/top-nlp-algorithms-amp-concepts/ (дата звернення: 13.03.2026).

24. Jurafsky D., Martin J. H. Speech and Language Processing. 3rd ed. draft. Stanford University. URL: https://web.stanford.edu/~jurafsky/slp3/ (дата звернення: 28.04.2026).

25. Newman S. Building Microservices : Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 612 p.

26. Khorikov V. Unit Testing Principles, Practices, and Patterns. Shelter Island : Manning Publications, 2020. 304 p.

27. Evans E. Domain-Driven Design : Tackling Complexity in the Heart of Software. Boston : Addison-Wesley Professional, 2003. 560 p.

28. Freeman E., Robson E. Head First Design Patterns. 2nd ed. Sebastopol : O'Reilly Media, 2020. 672 p.

29. Kanjilal J. Integrating AI and ML in .NET Core with ML.NET. CODE Magazine. 2025. URL: https://www.codemag.com/Article/2511051/Integrating-AI-and-ML-in-.NET-Core-with-ML.NET (дата звернення: 02.03.2026).

30. Watt A. Building Modern SaaS Applications with C# and .NET : Build, deploy, and maintain professional SaaS applications. Birmingham : Packt Publishing, 2023. 346 p.

31. McKinney W. Python for Data Analysis : Data Wrangling with pandas, NumPy, and Jupyter. 3rd ed. Sebastopol : O'Reilly Media, 2022. 582 p.

32. Building Production-Ready LLM Applications with .NET : A Practical Guide. ABP.IO. URL: https://abp.io/community/articles/building-production-ready-llm-applications-with-net-ya7qemfa (дата звернення: 20.03.2026).