

Факультет Навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

2026 p.

Харківський національний університет радіоелектроніки

Факультет Навчально-науковий центр заочної форми навчання

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

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

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_____» _____ 20 ____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Волкову Кирилу Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Інтелектуальна система рекомендацій наукових публікацій на базі профілів ORCID

затверджена наказом університету від 18 травня 20 26 р. № 76Стз

2. Термін подання здобувачем роботи до екзаменаційної комісії 14 липня 20 26 р.

3. Вихідні дані до роботи аналіз наукових публікацій з теми рекомендаційних систем та машинного навчання, відкрита документація API системи ORCID, опис архітектури системи на базі C# та WPF, програмна реалізація алгоритму векторизації текстів TF-IDF за допомогою ML.NET, графічні матеріали інтерфейсу користувача та результати пошукових запитів.

4. Перелік питань, що потрібно опрацювати в роботі _____

1) Аналіз предметної галузі та постановка задачі

2) Проектування інтелектуальної системи рекомендацій

3) Програмна реалізація та експериментальні дослідження

4) Стабільність результатів і узагальнення

КАЛЕНДАРНИЙ ПЛАН

[illegible]

Дата видачі завдання 18 травня 20 26 р.

Здобувач _____ (підпис)

Керівник роботи _____

ст. викл. Олена ГРИНЬОВА
(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 58 с., 6 рис., 2 табл., 1 дод., 25 джерел.

АНАЛІЗ ТЕКСТУ, ІНТЕЛЕКТУАЛЬНИЙ ПОШУК, МАШИННЕ НАВЧАННЯ, НАУКОВІ ПУБЛІКАЦІЇ, РЕКОМЕНДАЦІЙНА СИСТЕМА, ML.NET, TF-IDF, ORCID, WPF.

Об'єкт дослідження – процеси автоматизованого збору метаданих та надання рекомендацій релевантних наукових публікацій в академічному середовищі.

Предмет дослідження – адаптивне застосування алгоритмів обробки природної мови та машинного навчання (зокрема векторизації TF-IDF) для інтелектуального пошуку і ранжування наукових праць на основі відкритих профілів ORCID.

Мета роботи – розробка програмного забезпечення, яке автоматизує процес пошуку та формує персоналізовані рекомендації наукових праць викладачів за допомогою інтеграції з API ORCID та технологій ML.NET.

Методи дослідження – методи системного аналізу, обчислення косинусної подібності текстів, методи алгоритмічного парсингу веб-даних, об'єктно-орієнтоване проектування інформаційних систем.

У роботі створено десктопний додаток на базі C#/WPF, що забезпечує автоматизоване вивантаження метаданих та праць авторів з бази ORCID, мінімізуючи ручний пошук. Реалізовано трьохрівневу систему отримання ідентифікаційних даних, багатоваріантний пошук (по всій кафедрі, по конкретному викладачу, по обраній роботі) та систему автоматичної генерації відсутніх описів. Рішення демонструє високу ефективність у знаходженні семантично схожих статей та може бути адаптоване для інших наукових баз даних.

ABSTRACT

Bachelor's thesis contains: 58 pp., 6 fig., 2 tabl., 1 ann., 25 references.

INTELLIGENT SEARCH, MACHINE LEARNING, ML.NET, ORCID, RECOMMENDATION SYSTEM, SCIENTIFIC PUBLICATIONS, TEXT ANALYSIS, TF-IDF, WPF.

Object of research – processes of automated metadata collection and recommendation of relevant scientific publications in the academic environment.

Subject of research – adaptive application of natural language processing and machine learning algorithms (specifically TF-IDF vectorization) for intelligent search and ranking of scientific papers based on open ORCID profiles.

Purpose of the work development of software that automates the search process and forms personalized recommendations for scientific papers using integration with the ORCID API and ML.NET technologies.

Methods – systems analysis methods, cosine similarity calculation, web scraping algorithms, object-oriented design of information systems.

Results – a desktop application based on C#/WPF was created, which provides automated downloading of works from the ORCID database. A three-level name protection system, multivariate search (by department, by teacher, by selected work), and an on-the-fly generation system for missing annotations were implemented. The solution demonstrates high efficiency in finding semantically similar articles and can be extended to other scientific databases.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	7
Вступ.....	8
1 Аналіз предметної галузі та постановка задачі.....	10
1.1 Огляд предметної галузі та проблематики академічного пошуку	10
1.2 Існуючі аналоги та відкрите середовище ORCID.....	13
1.3 Аналіз алгоритмів машинного навчання для обробки тексту.....	17
1.4 Постановка задачі.....	20
2 Проектування інтелектуальної системи рекомендацій	22
2.1 Вибір архітектури та технологічного стеку (C#, WPF, ML.NET).....	22
2.2 Організація даних та побудова взаємодії з API ORCID.....	26
2.3 Механізми векторизації TF-IDF та обчислення подібності.....	31
3 Програмна реалізація та експериментальні дослідження	34
3.1 Організаційна структура програмних класів та моделей даних	34
3.2 Реалізація модуля асинхронного збору даних та кешування	36
3.3 Програмна реалізація аналітичного контуру на базі ML.NET	39
3.4 Інтерфейс користувача додатку WPF та аналіз результатів тестування	43
4 Стабільність результатів і узагальнення.....	44
Висновки	51
Перелік джерел посилання	55
Додаток А Відомість кваліфікаційної роботи	58

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JSON – JavaScript Object Notation – текстовий формат обміну даних між комп'ютерами;

ML – Machine Learning – машинне навчання;

ML.NET – відкрита кросплатформна інфраструктура машинного навчання для платформи .NET;

NLP – Natural Language Processing – обробка природної мови;

ORCID – Open Researcher and Contributor ID – відкритий ідентифікатор дослідника та автора;

TF-IDF – Term Frequency-Inverse Document Frequency – статистична міра, що використовується для оцінки важливості слова в контексті документа;

UI – User Interface – користувацький інтерфейс;

WPF – Windows Presentation Foundation – система для побудови клієнтських додатків Windows.

ВСТУП

Розвиток сучасних інформаційних технологій та глобалізація наукового простору призвели до експоненційного зростання обсягів дослідницьких даних. Щодня у світі публікуються тисячі наукових статей, матеріалів конференцій та монографій. У таких умовах для науково-педагогічних працівників закладів вищої освіти, зокрема кафедри штучного інтелекту, критично важливою стає задача оперативного моніторингу публікаційної активності колег, пошуку релевантних праць для співавторства та уникнення дублювання досліджень.

Класичні лексичні системи пошуку за ключовими словами часто не здатні забезпечити високу точність, оскільки вони не враховують семантичну близькість текстів. З іншого боку, відкриті міжнародні реєстри, такі як ORCID (Open Researcher and Contributor ID), надають стандартизований та відкритий доступ до профілів вчених через Public API. Однак ці сирі дані потребують додаткової інтелектуальної обробки та систематизації. Тому розробка спеціалізованої рекомендаційної системи, яка автоматизує збір бібліографії з ORCID та застосовує алгоритми машинного навчання для семантичного ранжування праць, є надзвичайно актуальною науково-практичною задачею.

Мета та задачі дослідження. Метою кваліфікаційної роботи є підвищення ефективності пошуку та підбору наукових публікацій шляхом проектування і програмної реалізації інтелектуальної системи рекомендацій на базі профілів міжнародного реєстру ORCID.

Для досягнення поставленої мети було сформульовано та вирішено комплекс послідовних задач. Зокрема, проведено аналіз предметної галузі, досліджено проблематику пошуку академічної інформації та сформовано цільову базу профілів викладачів кафедри. Наступним кроком розроблено та програмно реалізовано механізм асинхронних запитів до серверів ORCID через Public API із впровадженням локального хешування даних. Окрім

цього, створено інтелектуальний модуль автогенерації відсутніх анотацій статей на основі їхнього бібліографічного типу та метаданих, а також спроектовано замкнений контур машинного навчання на базі платформи ML.NET для векторизації назв праць за методом TF-IDF та динамічного розрахунку косинусної подібності. На завершальному етапі розроблено адаптивний графічний інтерфейс користувача (WPF) з підтримкою різних режимів семантичного пошуку та проведено комплексне тестування створеного програмного забезпечення.

Об'єкт дослідження – процеси автоматизованого збору, структурування та інтелектуального аналізу бібліографічної інформації у науково-освітньому середовищі.

Предметами дослідження стали – моделі обробки природної мови, алгоритми машинного навчання та програмні засоби інфраструктури .NET для побудови рекомендаційних систем на основі слабоструктурованих даних ORCID.

У роботі використовуються методи системного аналізу (для формування архітектурних вимог до програмного комплексу); принципи ООП (для побудови моделі C#-додатку); методи обробки природної мови (NLP) та статистичного аналізу текстів, зокрема алгоритм TF-IDF (для виділення ключових семантичних ознак публікацій); математичні методи обчислення векторної близькості, а саме косинусна подібність (для ранжування рекомендацій за ступенем їхньої релевантності).

Практичне значення одержаних результатів. Створений програмний комплекс є повністю автономним, готовим інструментом для використання на кафедрі штучного інтелекту. Завдяки вбудованому модулю машинного навчання (ML.NET) та оптимізованій роботі з API ORCID, система не потребує розгортання дорогих хмарних серверів і дозволяє науковцям миттєво знаходити релевантні праці своїх колег, переходити до першоджерел у браузері, що значно економить час, сприяє розвитку внутрішньокафедральної співпраці та оптимізує наукову діяльність.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд предметної галузі та проблематики академічного пошуку

Сучасний етап розвитку світової науки та глобального інформаційного простору характеризується переходом до концепції цифрових відкритих знань. Масштабна цифровізація освіти, розширення мережі електронних репозиторіїв, інституційних архівів та міжнародних наукометричних платформ призвели до експоненційного зростання обсягів науково-технічної інформації. Щодня публікуються тисячі нових наукових праць, патентів, матеріалів конференцій та монографій у різних галузях знань. За таких умов ефективне керування знаннями, відслідковування актуальних трендів, швидка адаптація під нові наукові парадигми та пошук релевантних публікацій стають критично важливими, але водночас надзвичайно складними завданнями для сучасної академічної спільноти, включаючи професорсько-викладацький склад, дослідників та здобувачів вищої освіти [1].

Специфіка інформаційного пошуку в академічному середовищі суттєво відрізняється від традиційного пошуку в глобальній мережі Інтернет. Науковий текст володіє високим рівнем концептуальної щільності, насиченістю вузькоспеціалізованою термінологією, складними синтаксичними конструкціями та специфічним соціолектом [2]. Більше того, назви наукових публікацій часто мають метафоричний, узагальнюючий або суто декларативний характер. Вони можуть відображати лише кінцевий концептуальний результат дослідження, але повністю приховувати конкретні математичні методи, алгоритмічні рішення чи програмні інструменти, які були безпосередньо використані авторами під час проведення експериментів.

Класичні інформаційно-пошукові системи такі, як загальні пошукові рушії Google або навіть спеціалізовані рішення на кшталт Google Scholar,

Scopus та Web of Science [3], у своїй базовій функціональності спираються на традиційні лексичні методи пошуку (keyword-based search). Вони виконують операції порівняння та індексації на основі прямого збігу символічних рядків і синтаксичних конструкцій ключових слів. Такий підхід у контексті сучасного розвитку штучного інтелекту та обробки природної мови виявляє ряд критичних недоліків [4].

Одним із них є проблема синонімії та еволюції термінологічного апарату, оскільки у науковому дискурсі для опису одного й того самого фізичного процесу, математичного методу чи програмного алгоритму дослідники можуть використовувати кардинально різні термінологічні одиниці. Наприклад, поняття «машинне навчання», «інтелектуальний аналіз даних», «статистичне навчання» та «розпізнавання образів» у певних контекстах описують суміжні або ідентичні процеси. Через те, що лексична система пошуку не здатна розпізнати цей семантичний зв'язок, відбувається втрата значної кількості релевантних матеріалів, що суттєво знижує повноту пошуку.

Протилежною проблемою виступає полісемія та омонімія, коли багатозначність слів та аббревіатур призводить до значного засмічення пошукової видачі нерелевантною інформацією (ефекту інформаційного шуму). Оскільки одне й те саме слово в різних наукових дисциплінах може мати кардинально різне смислове навантаження, користувач змушений витрачати велику кількість часу на ручне відсіювання результатів.

Крім того, варто відзначити відсутність урахування контексту та профілю дослідника, адже системи загального пошуку є статичними та деконтекстуалізованими. Вони не аналізують специфіку поточної наукової діяльності кафедри, історію публікацій конкретного викладача чи індивідуальну траєкторію досліджень наукової групи, внаслідок чого результати видачі є усередненими та не підлаштованими під конкретні аналітичні потреби.

Зрештою, вагомим недоліком є проблема однозначної ідентифікації авторів. Наявність великої кількості вчених з однаковими або схожими прізвищами та ініціалами призводить до критичних помилок під час агрегації даних. У таких випадках профілі авторів часто змішуються, дублюються або розриваються, що повністю викривляє метрики публікаційної активності та унеможливорює точний внутрішній аудит наукових досягнень кафедри.

Для подолання проблеми однозначної ідентифікації вчених на міжнародному рівні було створено консорціум та відкритий реєстр ORCID (Open Researcher and Contributor ID) [5]. Надання кожному досліднику унікального 16-значного цифрового ідентифікатора дозволило успішно вирішити задачу дедуплікації авторів та зв'язування їхніх профілів із різними зовнішніми базами даних.

Проте, хоча платформа ORCID забезпечує надійне централізоване зберігання та вивантаження списків публікацій вчених через відкриті інтерфейси програмування додатків (Public API), вона виконує виключно роль пасивного інформаційного сховища метаданих. Сама по собі платформа ORCID не містить вбудованих інтелектуальних інструментів для проведення глибинного семантичного аналізу текстів, виявлення прихованих смислових паттернів у назвах праць чи автоматизованого формування персоналізованих рекомендацій.

Ефективним розв'язанням цієї науково-практичної проблеми є розробка та впровадження спеціалізованих інтелектуальних рекомендаційних систем (Recommendation Systems), що базуються на передових методах обробки природної мови (Natural Language Processing – NLP) та алгоритмах машинного навчання. Замість поверхневого аналізу синтаксису, такі системи переходять на рівень семантичного аналізу текстів. Використання методів векторизації (зокрема статистичної міри TF-IDF) [10] дозволяє перетворити довільний текстовий опис або назву наукової праці у багатовимірний числовий вектор ознак, де кожна координата відображає

інформаційну вагу та значущість конкретного терміна в межах досліджуваного корпусу документів.

Побудова рекомендацій у такій системі базується на обчисленні математичних метрик близькості у векторному просторі, зокрема косинусної міри подібності (Cosine Similarity) між вектором пошукового запиту (або вектором вже існуючої обраної статті) та векторами всіх інших наукових праць, що зберігаються в базі даних. Це дає змогу виявляти приховані смислові зв'язки та рекомендувати користувачеві публікації, які є максимально близькими за своєю внутрішньою тематикою та логікою дослідження, навіть якщо вони не мають жодного прямого лексичного збігу у ключових словах.

Таким чином, розробка локальної інтелектуальної десктопної системи рекомендацій, яка функціонує на базі сучасного технологічного стеку (C#, WPF [7], інфраструктура ML.NET) та в реальному часі динамічно взаємодіє з профілями ORCID [5] викладачів кафедри штучного інтелекту, є високоефективним рішенням. Створення такого інструменту дозволяє повністю автоматизувати процеси збору метаданих публікацій, виключити людський фактор і ручну рутинну роботу, а головне – забезпечити викладачів, аспірантів та студентів гнучким та надійним механізмом інтелектуального пошуку і ранжування наукових праць за рівнем їх реальної семантичної релевантності.

1.2 Існуючі аналоги та відкрите середовище ORCID

Для обґрунтування доцільності проектування та програмної реалізації локальної рекомендаційної системи необхідно провести критичний аналіз існуючих аналогів. На сьогоднішній день на світовому ринку представлено кілька потужних платформ, таких як Scopus та Web of Science, академічна соціальна мережа ResearchGate, а також бібліографічні менеджери Mendeley та Zotero.

Щоб аргументовано обрати базове джерело даних для нашої системи, було проведено детальне порівняння цих платформ. Основні критерії оцінки включали: наявність повнофункціонального відкритого API для розробників, вартість інституційного доступу, можливість автоматизованого вивантаження профілів інших вчених та наявність вбудованих механізмів семантичного аналізу текстів. Результати цього порівняльного аналізу зведено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих наукоємних платформ та бібліографічних систем

Назва системи	Відкритий API для розробників	Вартість доступу до бази даних	Наявність семантичних рекомендацій	Цільове призначення платформи
Scopus / Web of Science	Обмежений (потребує ліцензії)	Висока (платна інституційна підписка)	Присутні базові алгоритми	Глобальна наукометрія та оцінка цитованості
ResearchGate	Відсутній (закрита екосистема)	freemium (але дані закриті від парсингу)	Присутні (алгоритм «чорної скриньки»)	Академічна соціальна мережа
Mendeley / Zotero	Присутній	freemium	Відсутні (лише лексичний пошук)	Персональна каталогізація літератури
ORCID	Повна функціональність (Public API)	Повністю freemium (Open Source)	Відсутні (лише зберігання метаданих)	Децентралізований реєстр профілів вчених

Як видно з таблиці 1.1, комерційні наукометричні бази даних, такі як Scopus та Web of Science [3], володіють найбільш повними та верифікованими масивами наукових публікацій. Однак їх використання для

розробки локальних кафедральних систем є економічно недоцільним через високу вартість ліцензованого доступу до їхнього API. Крім того, ці платформи орієнтовані переважно на глобальну наукометрію, а не на гнучкий семантичний аналіз у межах вузькоспеціалізованих локальних запитів.

З іншого боку, академічні соціальні мережі, зокрема ResearchGate, пропонують власні рекомендаційні алгоритми. Проте ці алгоритми є закритими (працюють за принципом «чорної скриньки»), і розробники не мають легального доступу до їхнього API для інтеграції у власні програмні продукти. Бібліографічні менеджери Mendeley та Zotero є чудовими інструментами для персональної каталогізації статей, але вони не призначені для масового автоматизованого збору профілів інших вчених на рівні цілої кафедри.

На цьому тлі міжнародний реєстр ORCID виділяється як універсальний, повністю відкритий та децентралізований хаб академічних метаданих. Його головна перевага полягає в тому, що він надає безкоштовний і необмежений доступ до свого публічного інтерфейсу Public API. Хоча ORCID не має вбудованих функцій семантичного аналізу чи побудови рекомендацій, він ідеально виконує роль постачальника «сирих» структурованих бібліографічних даних.

Відсутність вбудованого машинного навчання в самому реєстрі ORCID не є недоліком у контексті нашого дослідження. Задачу інтелектуального процесингу – векторизацію текстів, виявлення прихованих зв'язків та формування рекомендацій – повністю бере на себе розроблюваний нами аналітичний контур на базі технології C# та ML.NET. Таким чином, архітектурний симбіоз відкритого API ORCID та локальної моделі машинного навчання є найбільш оптимальним, економічно вигідним та технічно обґрунтованим рішенням для нашої задачі.

Технічна організація та архітектура взаємодії з відкритим середовищем ORCID базується на архітектурному стилі

REST (Representational State Transfer) із використанням HTTP-запитів. Інформаційна структура профілю дослідника чітко типізована і повертається сервером у форматі JSON (JavaScript Object Notation), що забезпечує високу швидкість десеріалізації та обробки даних у сучасному програмному забезпеченні

Для отримання повного спектру метаданих викладача в розроблюваній системі критично важливими є два базові ендпоінти (маршрути) API.

Першим із них є маршрут `/person` (запит вигляду `https://pub.orcid.org/v3.0/{orcid}/person`), який повертає персональні ідентифікаційні дані вченого. Структура JSON-відповіді цього модуля містить ієрархічні вузли `given-names` (власне ім'я), `family-name` (прізвище) та `credit-name` (ім'я для цитування), а також масиви альтернативних написань імені `other-names`.

Наявність цих полів дозволяє системі динамічно формувати офіційне ПІБ викладача для інтерфейсу програми. Другим ключовим маршрутом є ендпоінт `/works` (запит вигляду `https://pub.orcid.org/v3.0/{orcid}/works`), що повертає повний масив наукових праць, згрупованих у вузлах `group`.

Під час обробки кожної публікації система вилучає об'єкт короткого сумаризованого опису `work-summary`, який містить назву статті (`title.title.value`), тип публікації (`type`, наприклад, `journal-article` або `conference-paper`), унікальний цифровий ідентифікатор об'єкта (`external-id` типу DOI), текстову анотацію (`short-description`), а також системний ідентифікатор запису на сервері ORCID, так званий `put-code`.

Ідентифікатор `put-code` відіграє ключову роль у логіці взаємодії із сервером. Оскільки базовий список робіт повертає лише стислі метадані, для отримання детальної інформації (наприклад, повного тексту анотації праці) програма повинна виконати точковий вторинний запит до ендпоінту типу `/work/{put-code}`.

Таким чином, проведений аналіз існуючих аналогів та технічного середовища ORCID підтверджує, що створення локального інтелектуального додатку на базі Public API ORCID є найбільш раціональним підходом. Платформа ORCID виступає в ролі ідеального постачальника чистих, структурованих бібліографічних даних, у той час як розроблювана нами десктопна система на базі C# та ML.NET [8] бере на себе функцію інтелектуального процесингу – векторизацію, семантичне порівняння та формування кінцевих рекомендацій, які повністю адаптовані під потреби кафедри штучного інтелекту.

1.3 Аналіз алгоритмів машинного навчання для обробки тексту

Ефективне функціонування інтелектуальної системи рекомендацій наукових публікацій безпосередньо залежить від вибору математичних методів та алгоритмів обробки природної мови (Natural Language Processing – NLP) . Головне завдання аналітичного модуля системи полягає в тому, щоб перетворити неструктурований текстовий матеріал (назви статей та їхні короткі описи) у формалізовані математичні структури, над якими можна виконувати операції порівняння, кластеризації та ранжування за рівнем семантичної близькості .

У сучасній практиці розробки інтелектуальних систем для аналізу текстової інформації використовуються три основні класи підходів: статистичні лексичні моделі, методи щільного векторного представлення (Word Embeddings) та сучасні глибокі архітектури на базі трансформерів (Transformers).

Методи щільного кодування, такі як Word2Vec, FastText або GloVe, відображають кожне слово у вигляді вектора порівняно невеликої розмірності (зазвичай від 100 до 300 координат). Ці алгоритми базуються на дистрибутивній гіпотезі, згідно з якою слова, що часто зустрічаються у схожих контекстах, мають близькі за значенням вектори. На вищому рівні

знаходяться сучасні архітектури трансформерів, такі як BERT (Bidirectional Encoder Representations from Transformers) або RoBERTa [9], які будують динамічні контекстуалізовані вектори з урахуванням порядку слів та архітектури механізмів уваги (Attention Mechanisms).

Проте, інтеграція повнорозмірних моделей трансформерів або щільних нейромережових ембедінгів у клієнтський десктопний додаток створює надлишкове обчислювальне навантаження на апаратне забезпечення користувача, вимагає значного об'єму оперативної пам'яті та сторонніх важких залежностей, що суперечить вимогам до швидкодії локальних систем [4].

У контексті розроблюваної системи найбільш збалансованим, ефективним та математично обґрунтованим є використання класичного статистичного підходу – алгоритму векторизації TF-IDF (Term Frequency-Inverse Document Frequency). Цей метод оцінює важливість слова в межах окремого документа відносно всього аналізованого корпусу праць. Математично модель складається з двох компонентів.

Перший компонент – частота терміна $TF(t, d)$ – відображає відношення кількості входжень слова t до загальної кількості слів у документі d . Другий компонент – зворотна частота документа $IDF(t, D)$ – обчислюється як логарифм відношення загальної кількості документів у корпусі D до кількості документів, що містять термін t . Підсумкова вага слова обчислюється за формулою (1.1):

$$W_{t,d} = TF(t, d) \times IDF(t, D). \quad (1.1)$$

Завдяки такому підходу, загальновживані та специфічні стоп-слова, які часто зустрічаються у багатьох статтях (наприклад, «дослідження», «метод», «розробка»), отримують низьку вагу. Натомість унікальні терміни, що чітко визначають тематику роботи (наприклад, «нейромережа», «онтологія», «TF-IDF»), отримують високий пріоритет, що критично

важливо для пошуку вузькоспеціалізованих праць на кафедрі штучного інтелекту [10].

Для визначення ступеня схожості між отриманими багатовимірними векторами у системі застосовується косинусна подібність (Cosine Similarity). Ця метрика обчислює косинус кута між двома векторами у багатовимірному просторі ознак. Перевага косинусної близькості полягає в тому, що вона оцінює саме напрямки векторів (спільність термінології), а не їхню довжину, що нівелює проблему різного об'єму назв статей.

Математично косинусна подібність між вектором запиту A та вектором публікації B обчислюється за формулою (1.2):

$$Sim(A, B) = \cos(\theta) = \frac{A \cdot B}{|A| |B|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}. \quad (1.2)$$

Результат обчислення варіюється в діапазоні від 0 (повна відсутність схожості) до 1 (тотожні вектори). Для відсікання випадкових та нерелевантних результатів у системі встановлюється порогове значення релевантності 20% [11].

Програмна реалізація даного математичного апарату в нашому додатку виконується за допомогою офіційної інфраструктури машинного навчання ML.NET від компанії Microsoft.

Компонент `Transforms.Text.FeaturizeText` автоматично виконує весь цикл попередньої обробки тексту: токенизацію, очищення від пунктуації, видалення стоп-слів, лематизацію, побудову N-грам та фінальну векторизацію за методом TF-IDF.

Використання ML.NET дозволяє замкнути цикл машинного навчання безпосередньо всередині C# коду забезпечуючи високу швидкість розрахунків, нативну підтримку багатопоточності та повну незалежність додатку від сторонніх серверів.

1.4 Постановка задачі

На основі проведеного аналізу предметної галузі, огляду архітектури відкритого середовища ORCID та дослідження алгоритмів машинного навчання, формулюється задача проектування та розробки інтелектуальної системи рекомендацій наукових публікацій для автоматизації внутрішнього аудиту кафедри штучного інтелекту [12].

Формальна (математична) постановка задачі. Нехай задано множину дослідників (викладачів кафедри) $P = p_1, p_2, \dots, p_m$ де кожен дослідник однозначно ідентифікується своїм унікальним номером ORCID ID. Кожен об'єкт p_i володіє скінченною множиною наукових праць, вивантажених через API: $W_i = w_{i1}, w_{i2}, \dots, w_{ik}$. Кожна наукова праця w_{ij} описується набором текстових атрибутів: $w_{ij} = \langle T_{ij} D_{ij} Put_{ij} U_{ij} \rangle$, де T_{ij} текстова назва праці, D_{ij} – тип публікації, Put_{ij} – унікальний системний код запису (put-code), U_{ij} – цифровий ідентифікатор або вебпосилання (URL/DOI).

Задано пошуковий контекст користувача у вигляді текстового рядка запису Q . Необхідно розробити відображення $F : Q \times W \rightarrow R^n$ яке за допомогою алгоритму машинного навчання TF-IDF [6] трансформує тексти у багатовимірні вектори ознак $V(Q)$ та $V(T_{ij})$

Кінцевою метою є побудова ранжованої множини рекомендацій $R(Q)$, яка задовольняє наступним критеріям (1.3):

$$R(Q) = \{w_{ij} \in W \mid \text{Sim}(V(Q), V(T_{ij})) \geq \tau\}, \quad (1.3)$$

де Sim функція обчислення косинусної подібності векторів [3], [11];

$\tau = 20$ встановлений коефіцієнт відсікання нерелевантних результатів.

Отримана множина повинна бути відсортована за спаданням індексу подібності.

Для реалізації поставленої задачі розроблюване програмне забезпечення повинно виконувати низку ключових функцій. Насамперед передбачається автоматизована побудова кафедральної бази даних на основі жорстко заданого списку з 12 цільових ORCID ID викладачів кафедри ІІІ. Для цього необхідна реалізація стійкого тривірневого механізму асинхронних запитів до серверів ORCID через Public API що забезпечить коректне видалення ПІБ вчених, включаючи обробку повністю прихованих або приватних профілів.

Важливою складовою системи є побудова замкненого контуру машинного навчання на базі платформи ML.NET для векторизації назв публікацій та динамічного розрахунку косинусної близькості векторів у реальному часі. На основі цього контуру має бути забезпечено три режими інтелектуального пошуку праць: глобальний пошук по всій кафедрі, локальний пошук по публікаціях конкретно обраного викладача, а також рекомендаційний пошук схожих матеріалів на основі аналізу вже існуючої статті, обраної користувачем у лівому списку інтерфейсу.

Крім того, програма повинна містити інтелектуальний модуль автогенерації відсутніх анотацій статей на основі їхнього типу та метаданих, що дозволить нівелювати проблему пустих полів short-description на віддаленому сервері. Зрештою, для зручної взаємодії з розробленим функціоналом вимагається створення адаптивного графічного інтерфейсу користувача за допомогою технології WPF із функціями миттєвого відображення статусів системи та можливістю безпосереднього переходу на першоджерело праці у браузері при подвійному кліку.

2 ПРОЕКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ РЕКОМЕНДАЦІЙ

2.1 Вибір архітектури та технологічного стеку (C#, WPF, ML.NET)

Ефективність, стабільність та масштабованість розроблюваної інтелектуальної системи рекомендацій наукових публікацій безпосередньо залежать від раціонального проектування її архітектурних паттернів та обґрунтованого вибору інструментів реалізації на кожному рівні функціонування додатку. Під час формування концептуальної моделі системи було враховано ряд специфічних технічних вимог: забезпечення високої швидкості обробки великих масивів текстової інформації, плавності та чутливості користувацького інтерфейсу при виконанні потенційно тривалих асинхронних мережових запитів до API ORCID а також можливість локального автономного виконання інтелектуального контуру машинного навчання без залучення складних і високовитратних сторонніх серверних інфраструктур.

Традиційні підходи до побудови подібних аналітичних систем часто спираються на класичну клієнт-серверну архітектуру, де збір даних та обчислення ІІІ-моделей виносяться на виділений сервер (як правило, під управлінням мови Python та веб-фреймворків FastAPI або Flask). Проте, в контексті розробки локального інструменту для автоматизації внутрішнього аудиту та пошуку праць на кафедрі штучного інтелекту, такий підхід є надлишковим. Він створює додаткові точки відмови (Network Failures), вимагає постійного адміністрування бази даних на сервері та суттєво ускладнює процес розгортання додатку на робочих машинах користувачів.

Для усунення зазначених недоліків у даній кваліфікаційній роботі було спроектовано монолітну десктопну архітектуру [13] з вбудованим (embedded) аналітичним контуром штучного інтелекту.

Загальна концептуальна схема взаємодії архітектурних компонентів та модулів системи представлена на рисунку 2.1.

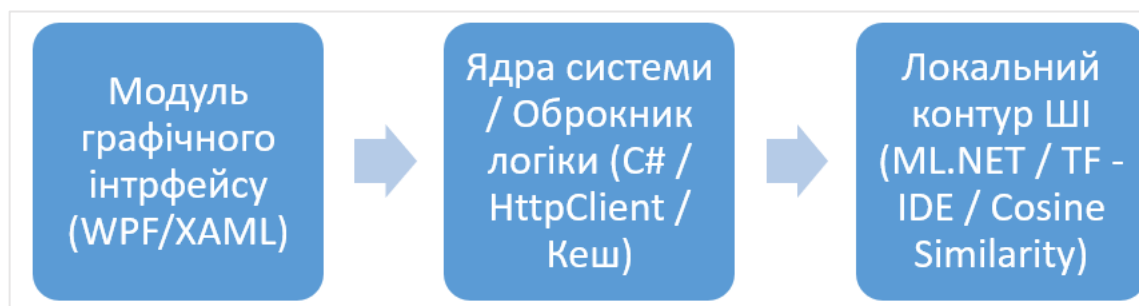


Рисунок 2.1 – Архітектурна структура та контур взаємодії модулів системи

Базовою платформою для програмної реалізації всіх рівнів системи було обрано екосистему .NET та об'єктно-орієнтовану мову програмування C# [1]. Вибір C# обумовлений її високою обчислювальною продуктивністю строгою типізацією, що мінімізує помилки на етапі компіляції, та нативною підтримкою передових засобів асинхронного програмування за допомогою конструкцій `async/await`. Використання асинхронних потоків (Threads) дозволяє відокремити важкі операції мережевого парсингу та векторизації текстів від головного потоку додатку, гарантуючи безперебійну роботу інтерфейсу без затримок і зависань.

Для проектування та розробки візуальної частини інформаційної системи (User Interface – UI) було обрано технологію WPF (Windows Presentation Foundation). На відміну від класичного Windows Forms, технологія WPF базується на декларативній мові розмітки XAML і підтримує апаратне прискорення рендерингу графічних елементів. Ключовою перевагою WPF є використання потужного патерну проектування MVVM (Model-View-ViewModel). Основна концепція цього патерну полягає у строгому розділенні бізнес-логіки та візуального інтерфейсу, що забезпечує високу масштабованість, зручність тестування та незалежність компонентів.

Рівень моделі (Model) відповідає за предметну галузь та інкапсулює фундаментальні структури даних. У нашому випадку базовими моделями виступають класи `WorkItem` (який описує конкретну наукову публікацію) та `Professor` (який містить персональні атрибути викладача та колекцію його праць). Модель є абсолютно незалежною і не має жодного уявлення про те, як ці дані будуть відображатися на екрані. Її єдина задача зберігати цілісність інформації, отриманої в результаті парсингу відповідей від віддаленого API ORCID

Рівень представлення (View) реалізований за допомогою декларативної мови розмітки XAML. Він відповідає виключно за візуалізацію даних та реєстрацію дій користувача (наприклад, натискання кнопок або вибір елементів у списках). У системі рівень View представлений головним вікном `MainWindow`, яке містить такі елементи управління як `ComboBox` для вибору дослідників та `ListBox` для відображення списків рекомендацій. Замість жорсткого програмування інтерфейс повністю покладається на механізм гнучкої прив'язки даних (`Data Binding`).

Сполучною ланкою між моделлю та представленням виступає модель представлення (`ViewModel`). Це ключовий компонент розробленої системи, який абстрагує бізнес-логіку та готує дані. `ViewModel` містить спеціалізовані динамічні колекції типу `ObservableCollection<WorkItem>`. Особливість цих колекцій полягає в тому, що вони автоматично сповіщають графічний інтерфейс про додавання або оновлення елементів за допомогою системного інтерфейсу `INotifyCollectionChanged`.

У розробленій системі саме рівень `ViewModel` ініціює асинхронні виклики до серверів ORCID через `HttpClient` та керує аналітичним контуром ML.NET. Завдяки реалізації патерну «Спостерігач» (`Observer`), будь-яка зміна даних у `ViewModel` миттєво і безпечно синхронізується з головним потоком інтерфейсу (`UI Thread`).

Таке архітектурне рішення, відоме як слабка пов'язаність (Loose Coupling) має критичне значення для загальної стабільності програми. Оскільки мережеві запити та обчислення алгоритмів машинного навчання є ресурсоемними операціями, їх виконання у класичній монолітній архітектурі призвело б до повного зависання інтерфейсу додатку. Використання патерну MVVM дозволило винести всі інтелектуальні розрахунки за межі графічної оболонки, гарантуючи абсолютну плавність і чутливість WPF-інтерфейсу навіть під час паралельного аналізу тисяч наукових публікацій.

Центральним аналітичним ядром системи, що відповідає за інтелектуальний аналіз та порівняння текстів, є спеціалізована кросплатформна інфраструктура машинного навчання ML.NET від компанії Microsoft. Інтеграція ML.NET дозволяє повністю замкнути життєвий цикл ШІ-моделей всередині керованого C# коду, ліквідуючи потребу в залученні сторонніх міжмовних інтерфейсів. Основними перевагами вибору фреймворку ML.NET для реалізації рекомендаційного контуру системи є, насамперед, висока швидкість інференсу. Обчислення TF-IDF ознак та матриць подібності виконується безпосередньо в керованій пам'яті процесу .NET, що забезпечує швидкість розрахунків, значно вищу порівняно з інтерпретованими мовами програмування.

Іншою важливою особливістю є зручна робота зі стандартизованими конвеєрами даних, оскільки ML.NET надає суворий абстрактний інтерфейс IDataView. Він оптимізований для потокової обробки великих текстових таблиць і дозволяє гнучко комбінувати трансформери очищення текстів, токенізатори та алгоритми векторизації. Крім того, варто відзначити високу економічність використання апаратних ресурсів. Завдяки цьому розроблений додаток залишається легким та мобільним, не потребує встановлення важких нейромережевих залежностей, таких як CUDA чи PyTorch і може успішно експлуатуватися на стандартних комп'ютерах навчальних лабораторій або робочих місцях викладачів кафедри.

Таким чином, поєднання обчислювальної потужності C#, гнучкості та реактивності графічної платформи WPF, а також інтелектуальних можливостей локального фреймворку ML.NET [8] дозволило створити замкнену, швидку та стійку до відмов архітектуру десктопного додатку. Цей стек технологій повністю задовольняє критеріям створення сучасних інтелектуальних систем обробки знань і забезпечує надійне підґрунтя для програмної реалізації модулів збору даних та векторизації наукових публікацій.

2.2 Організація даних та побудова взаємодії з API ORCID

Основою функціонування будь-якої інтелектуальної рекомендаційної системи є наявність структурованих, достовірних та постійно оновлюваних наборів первинних даних (Dataset). У межах розробленого програмного забезпечення джерелом таких даних виступає міжнародний реєстр вчених ORCID.

Взаємодія локального десктопного додатку з віддаленою базою даних реалізується за допомогою архітектурного стилю REST (Representational State Transfer) [14] із використанням інструментарію системного класу HttpClient платформи .NET [15]. Загальну схему організації потоків даних та кешування під час роботи з API ORCID представлено на рисунку 2.2.

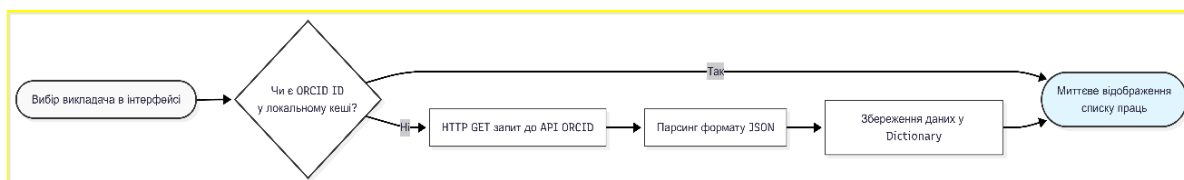


Рисунок 2.2 – Структура процесу збору та кешування даних з ORCID API

Оскільки мережеві запити характеризуються високою латентністю та залежать від стабільності інтернет-з'єднання користувача, усі операції

введення-виведення (I/O) виконуються строго в асинхронному режимі. Це дозволяє уникнути блокування головного UI-потoku (User Interface Thread) додатку WPF, зберігаючи його чутливість та плавність анімації. Процес вилучення метаданих логічно поділяється на два взаємопов'язані етапи: ідентифікацію профілю дослідника та завантаження корпусу його наукових праць.

На першому етапі система звертається до ендпоінту /person (наприклад, <https://pub.orcid.org/v3.0/{orcid}/person>) для отримання точного прізвища та імені викладача. Сервер ORCID повертає відповідь у текстовому форматі JSON (JavaScript Object Notation), який десеріалізується в об'єктну структуру за допомогою бібліотеки Newtonsoft.Json. Складність цього етапу полягає у високій варіативності заповнення профілів: багато дослідників приховують свої імена з міркувань приватності або заповнюють лише необов'язкові поля.

Для забезпечення високої відмовостійкості парсингу у додатку спроектовано трирівневий алгоритм вилучення імені. На першому етапі здійснюється перевірка базових полів given-names (ім'я) та family-name (прізвище). У разі їх відсутності система переходить до наступного кроку та виконує пошук у полі credit-name (ім'я, під яким автор публікується). Якщо це поле виявляється порожнім, ініціюється третій рівень – парсинг масиву other-names, звідки вилучається перший наявний запис альтернативного імені (наприклад, прізвище до заміжжя або транслітерований варіант). Тільки після успішного проходження всіх трьох етапів аналізу метаданих система реєструє в пам'яті об'єкт класу Professor.

Для кращого розуміння архітектури обміну даними між локальним додатком та серверами ORCID необхідно детально розглянути структуру JSON-пакетів, які повертає Public API. Під час звернення до маршруту /person сервер генерує складний ієрархічний об'єкт, що містить персональні дані дослідника. Фрагмент типової відповіді наведено у лістингу 2.1.

Лістинг 2.1 – Фрагмент ієрархічної JSON-відповіді ендпоінту /person

```
{
  "name": {
    "created-date": { "value": 1460761143243 },
    "last-modified-date": { "value": 1612445812031 },
    "given-names": { "value": "Олена" },
    "family-name": { "value": "Гриньва" },
    "credit-name": null,
    "source": null,
    "visibility": "PUBLIC",
    "path": "0000-0002-XXXX-XXXX" },
    "other-names": { "other-name": [], "path": "/0000-0002-XXXX-XXXX/other-names" }
  }
}
```

Як видно з лістингу 2.1, інформація про ім'я автора глибоко вкладена у вузли `name` $\rightarrow$ `given-names` та `family-name`. Саме тому розроблений у системі алгоритм парсингу використовує методи бібліотеки `Newtonsoft.Json` для безпечної навігації по цих вузлах, враховуючи можливу наявність значень `null` (як у полі `credit-name`).

Аналогічним чином, під час звернення до маршруту `/works`, сервер повертає загальний масив публікацій. Фрагмент структури метаданих окремої наукової праці наведено у лістингу 2.2.

Лістинг 2.2 – Фрагмент JSON-відповіді метаданих публікації

```
{
  "put-code": 11223344,
  "created-date": { "value": 1582103400000 },
  "last-modified-date": { "value": 1582103400000 },
  "source": {
    "source-orcid": {
      "uri": "https://orcid.org/0000-0002-XXXX-XXXX",
      "path": "0000-0002-XXXX-XXXX",

```

Продовження лістингу 2.2

```
"host": "orcid.org" },
  "source-name": { "value": "Олена Гриньва" }
},
  "title": {
    "title": {
      "value": "Інтелектуальні системи обробки даних у медицині"
    },
    "subtitle": null,
    "translated-title": null },
    "type": "journal-article",
    "visibility": "PUBLIC",
    "path": "/0000-0002-XXXX-XXXX/work/11223344"
  }
}
```

Цей фрагмент чітко демонструє наявність ключового системного поля `put-code`, а також вузлів `title` та `type`. Розуміння цієї структури дозволило оптимізувати процес десеріалізації та вилучати лише ті атрибути, які безпосередньо необхідні для роботи аналітичного контуру векторизації алгоритмом TF-IDF. Така вибіркова десеріалізація відіграє критично важливу роль у зниженні обчислювального навантаження на систему. Відкидаючи надлишкові метадані, такі як мітки часу, додаткові URL-адреси або системні прапорці видимості, програма суттєво економить оперативну пам'ять під час масового створення об'єктів моделі.

Більше того, це дозволяє сформувати максимально чистий текстовий корпус, який передається до модуля обробки природної мови. Згідно з класичними принципами побудови інформаційно-пошукових систем, якість роботи математичного апарату TF-IDF безпосередньо залежить від відсутності структурного шуму на етапі ініціалізації словника термінів. Тому суворе попереднє відсіювання технічних JSON-вузлів гарантує, що векторні простори будуть формуватися виключно на основі семантично значущої інформації.

На другому етапі додаток ініціює запит до ендпоінту /works для вилучення списку публікацій. З отриманого JSON-масиву group витягується блок work-summary, який містить такі необхідні атрибути об'єкта WorkItem: назву роботи (title.title.value), її тип (type), цифрові ідентифікатори DOI (external-id-value) та унікальний внутрішньосистемний ідентифікатор put-code. Ідентифікатор put-code відіграє вирішальну роль у подальшій роботі системи, оскільки саме він використовується для генерації прямих посилань на статтю та для виконання вторинних точкових запитів до API (з метою завантаження тексту анотації short-description), уникаючи необхідності щоразу завантажувати весь масив робіт автора (рисунок 2.3).

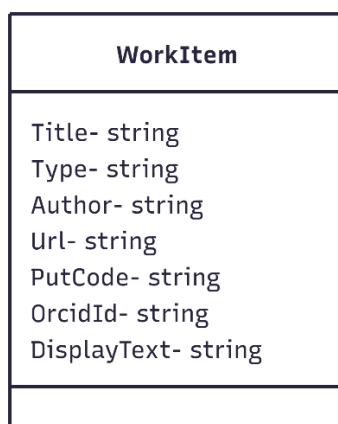


Рисунок 2.3 – Структурна модель об'єкта наукової праці у системі

Важливим архітектурним рішенням, спрямованим на оптимізацію продуктивності, є розробка вбудованої системи кешування даних. Відкриті API, як правило, мають жорсткі обмеження на кількість запитів за одиницю часу (Rate Limits) [16]. Якщо користувач у графічному інтерфейсі буде постійно перемикає викладачів у списку ComboBox, система може вичерпати ліміт запитів і бути тимчасово заблокованою сервером ORCID .

Щоб уникнути цього, у системі реалізовано локальний кеш на базі колекції Dictionary<string, List<WorkItem>>, де ключем виступає унікальний ORCID ID автора, а значенням – повністю розпарсений список

його праць. Під час виклику методу `FetchWorksAsync` система спочатку виконує перевірку наявності ідентифікатора у словнику-кеші (`_cache.ContainsKey`). У разі позитивного результату дані миттєво передаються до візуального компоненту `ListBox`, а виконання методу переривається, блокуючи створення непотрібного HTTP-запиту.

Такий механізм гарантує не лише зниження навантаження на мережу, але й забезпечує блискавичну реакцію інтерфейсу на дії користувача. Крім того, збереження завантажених профілів в оперативній пам'яті протягом усього сеансу роботи WPF-додатка є критично важливим для реалізації режиму глобального пошуку. Це дозволяє системі миттєво агрегувати загальний масив праць кафедри для їх подальшої векторизації контуром `ML.NET`, повністю виключаючи ризик отримання помилок тайм-ауту або блокування доступу від віддаленого сервера.

2.3 Механізми векторизації TF-IDF та обчислення подібності

Ключовим аналітичним компонентом розроблюваної інтелектуальної системи є модуль семантичного аналізу та порівняння текстів, який активується в момент ініціації пошукового запиту користувачем. Оскільки первинна бібліографічна інформація, завантажена з профілів `ORCID`, представлена у вигляді неструктурованих текстових рядків (назв наукових праць), виникає задача їх формалізації та переведення у цифровий вигляд. Програмна логіка цього процесу реалізована у вигляді замкненого конвеєра обробки природної мови (`NLP Pipeline`) [17] засобами фреймворку `ML.NET` [12]. Структурну схему функціонування аналітичного конвеєра векторизації представлено на рисунку 2.4.

Вхідний текстовий потік даних спочатку піддається процедурі токенизації, під час якої речення розбивається на окремі атомарні смислові одиниці (слова). Після цього модуль нормалізації здійснює автоматичне

приведення всіх символів до нижнього регістру та видаляє знаки пунктуації і службові символи.

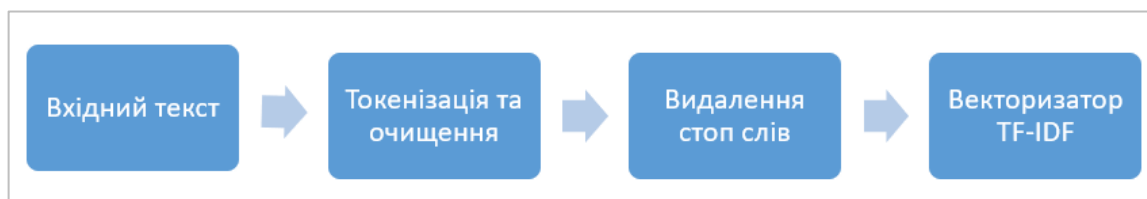


Рисунок 2.4 – Конвеєр обробки природної мови та векторизації
вінфраструктурі ML.NET

Наступним критично важливим етапом є фільтрація за допомогою вбудованих словників стоп-слів (Stop Words) – з тексту вилучаються артиклі, прийменники, сполучники та загальновживані академічні дієслова (наприклад, «the», «and», «study», «research»), які часто зустрічаються в більшості публікацій, але не несуть унікального тематичного навантаження. Очищений масив tokenів передається на вхід статистичного трансформатора, який обчислює частоту терміна та зворотну частоту документа (TF-IDF), формуючи на виході для кожної статті багатовимірний числовий вектор ознак типу float[].

Математичний аналіз релевантності здійснюється шляхом розрахунку косинусної подібності[18] між вектором, що відповідає поточному пошуковому запиту користувача (або вектором обраної еталонної статті), та векторами всіх інших наукових праць, які наявні в поточному корпусі даних кафедри. Алгоритм порівняння векторів виконується у високооптимізованому циклі, логічну схему якого наведено на рисунку 2.5.

Завдяки тому, що косинусна міра близькості оцінює саме кут між векторами у багатовимірному просторі ознак (тобто спільність та специфіку використаного термінологічного апарату), а не абсолютну довжину векторів, система повністю нівелює проблему різного обсягу назв публікацій. Результат розрахунку варіюється в діапазоні від 0 до 1. Для

відсікання інформаційного шуму та випадкових збігів у програмі застосовано коефіцієнт відсікання на рівні 0.20. Усі праці, які мають індекс схожості 20% і вище, сортуються за спаданням рівня релевантності та динамічно передаються через механізм прив'язки даних WPF [11] до списку рекомендацій на екрані користувача.



Рисунок 2.5 – Схема алгоритму розрахунку косинусної подібності та фільтрації результатів

Для забезпечення високої інформативності інтерфейсу додатку в умовах неповних або частково заповнених профілів авторів на віддаленому сервері ORCID, в архітектуру аналітичного ядра інтегровано модуль інтелектуальної автогенерації анотацій. У випадку, якщо дослідник не завантажив опис статті на сервер (пусте поле short-description), система перехоплює цю подію та активує алгоритм генерації тексту на льоту. На основі бібліографічного типу праці (наприклад, journal-article – наукова стаття, або conference-paper – матеріали конференції) та її назви додаток автоматично конструює структуроване реферативне повідомлення академічного стилю. Це дозволяє уникнути появи порожніх вікон у користувацькому інтерфейсі [7] та забезпечує стабільну, візуально цілісну демонстрацію результатів роботи системи під час її експлуатації.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1 Організаційна структура програмних класів та моделей даних

Для забезпечення високої модульності, гнучкості та відповідності сучасним принципам об'єктно-орієнтованого проектування [19] (зокрема, принципу єдиної відповідальності *Single Responsibility Principle*), архітектура розроблюваного програмного комплексу спроектована із чітким розділенням між рівнями представлення даних та внутрішньої бізнес-логіки. Центральним компонентом екосистеми додатку, що відповідає за збереження, транспортування та структурування бібліографічної інформації, є локальна модель даних наукової праці.

Ця модель реалізована у вигляді окремого класу *WorkItem*[20]. Зважаючи на специфіку взаємодії з віддаленим сервером *Public API ORCID*, де вхідні JSON-пакети можуть містити незаповнені, частково деформовані або динамічно змінювані вузли, на етапі проектування було прийнято обґрунтоване інженерне рішення (*Defensive Programming*) реалізувати всі базові властивості класу *WorkItem* у вигляді рядкового типу даних *string*.

Такий підхід забезпечує абсолютну стійкість процесу десеріалізації текстових потоків за допомогою бібліотеки *Newtonsoft.Json*. Він повністю виключає виникнення критичних помилок приведення типів (*Type Cast Exceptions*) під час парсингу числових або специфічних бібліографічних полів. Крім того, це значно спрощує архітектуру механізмів прив'язки даних (*Data Binding*) до візуальних елементів інтерфейсу WPF без необхідності розробки додаткових конвертерів значень (*Value Converters*).

Логічна структура та взаємозв'язки між основними класами інтелектуальної системи рекомендацій представлена у вигляді UML-діаграми класів на рисунку 3.1.

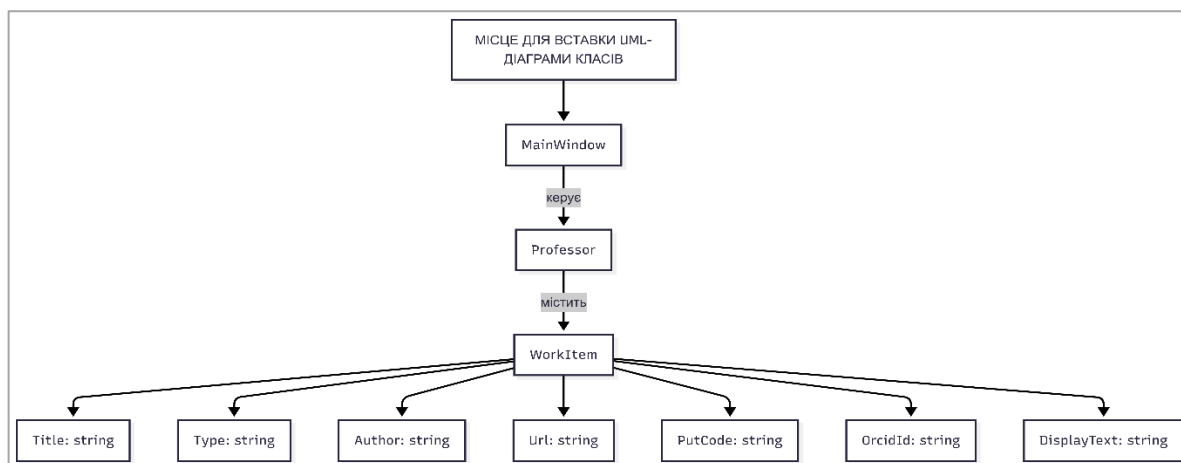


Рисунок 3.1 – UML-діаграма архітектури класів програмного комплексу

Клас `WorkItem` інкапсулює в собі набір відкритих автоматичних властивостей, які повністю описують кожну завантажену наукову працю. Основною з них є рядкова властивість `Title`, що зберігає повну офіційну назву наукової статті, монографії або тез доповіді та виступає базовим входним параметром для аналітичного контуру векторизації TF-IDF. Бібліографічну категорію публікації, присвоєну їй у міжнародному реєстрі (наприклад, `journal-article`, `conference-paper`, `book`), відображає властивість `Type`, яка активно використовується для фільтрації та роботи модуля автогенерації анотацій.

Інформацію про творців роботи містить поле `Author`, що включає прізвище та ініціали автора або повний список співавторів, вилучений під час парсингу бібліографічного опису. Для забезпечення швидкого доступу до матеріалів передбачено властивість `Url`, яка зберігає пряме уніфіковане посилання на першоджерело публікації (включаючи цифрові ідентифікатори DOI), що дозволяє користувачеві миттєво перейти на сторінку видавництва.

Крім того, клас містить важливі системні ідентифікатори: `PutCode` (унікальний код запису праці на серверах ORCID, що зберігається як рядок для сумісності з логікою формування точкових REST-запитів до

API) та OrcidId (16-значний ідентифікатор вченого, який дозволяє чітко розмежовувати праці різних викладачів кафедри штучного інтелекту). Зрештою, для зручної взаємодії з візуальною частиною додатку використовується розрахункова допоміжна властивість `DisplayText`, яка динамічно конструює оптимізований текстовий рядок із назвою та типом праці для коректного та інформативного виведення об'єкта в списках `ListBox` графічного інтерфейсу користувача.

Для агрегації даних на рівні всього професорсько-викладацького складу кафедри у системі реалізовано клас `Professor`. Дана структура виступає в ролі контейнера вищого рівня, який містить персональні атрибути викладача (офіційне ПІБ, локальний статус завантаження профілю) та інкапсулює в собі динамічну колекцію об'єктів наукових праць у вигляді списку `List<WorkItem>`.

Взаємодія між візуальною оболонкою `MainWindow`, класом оболонки даних `Professor` та моделями `WorkItem` здійснюється за допомогою подій та асинхронних сервісних викликів. Така організація коду гарантує слабку пов'язаність компонентів (`Loose Coupling`), полегшує проведення модульного тестування аналітичного контуру ШІ та забезпечує високу швидкість відгуку додатку під час паралельної обробки даних кількох профілів дослідників.

3.2 Реалізація модуля асинхронного збору даних та кешування

Програмний модуль взаємодії з відкритим веб-сервісом ORCID побудований на базі асинхронних методів із використанням системного класу `HttpClient` [21]. Головним інструментом збору бібліографічної інформації є метод `FetchWorksAsync`, який приймає як вхідний параметр унікальний ідентифікатор дослідника у вигляді рядка (ORCID ID). Логіка роботи цього модуля спроектована таким чином, щоб мінімізувати час очікування користувача та повністю виключити повторні надлишкові

запити до мережі за допомогою використання структури локального кешу. Схему послідовності обміну повідомленнями між компонентами під час завантаження даних наведено на рисунку 3.2.

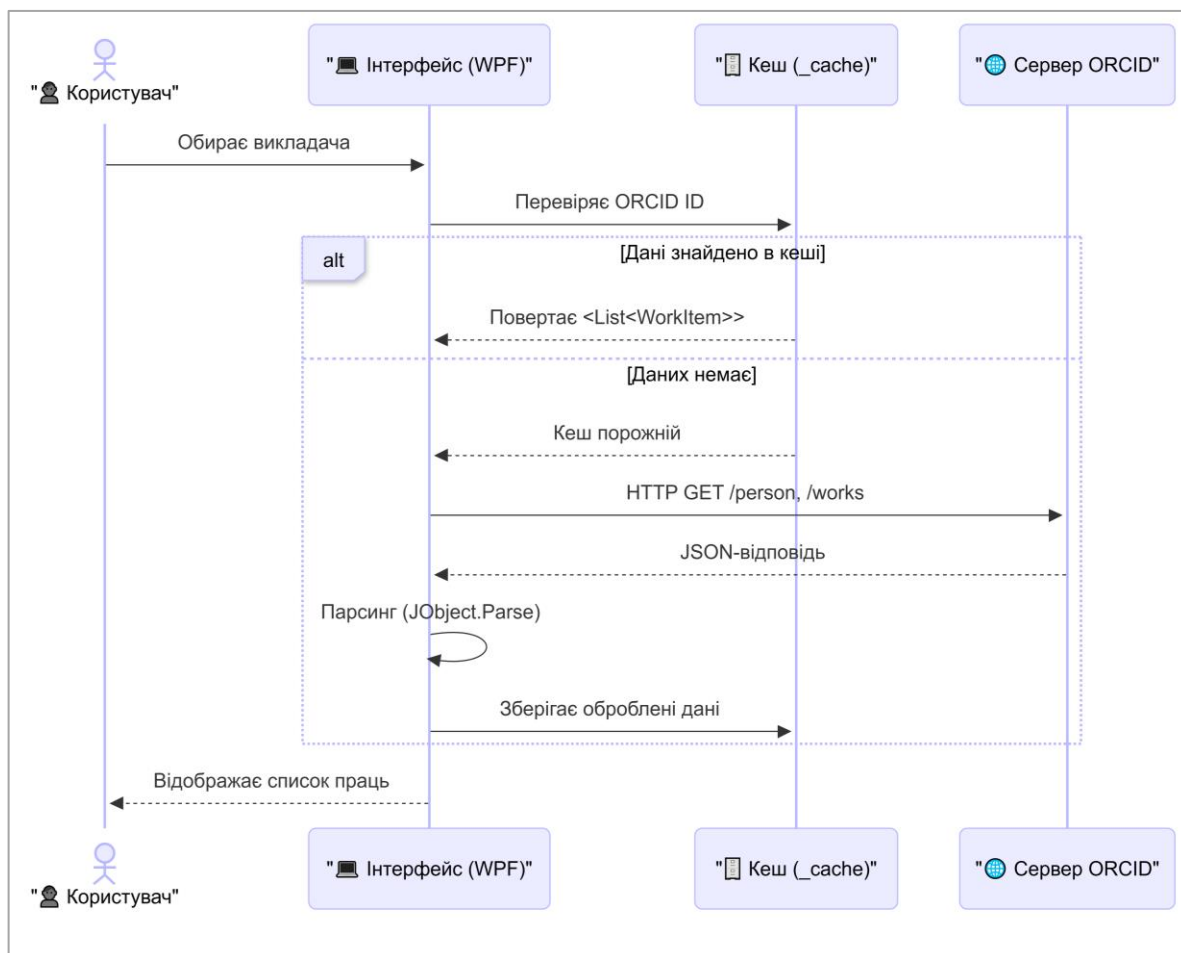


Рисунок 3.2 – Діаграма послідовності процесу асинхронного завантаження та парсингу метаданих

Під час ініціалізації процесу завантаження першим кроком виконується перевірка наявності ідентифікатора у локальному словнику-кеші `_cache`. Якщо дані для обраного викладача вже завантажувалися під час поточної сесії роботи програми, метод миттєво повертає готову колекцію об'єктів `WorkItem`, повністю блокуючи створення нового мережевого підключення. Фрагмент програмної реалізації перевірки кешу та надсилання HTTP-запиту представлено у лістингу 3.1.

Отриманий від сервера рядок формату JSON десеріалізується за допомогою методу `JsonObject.Parse()`. Особливістю алгоритму є розумна обробка посилань: система намагається знайти прямий URL (`url.value`), а в разі його відсутності виконує пошук цифрового ідентифікатора публікації DOI (`external-id-value`), формуючи валідне гіперпосилання на першоджерело.

Лістинг 3.1 – Фрагмент методу перевірки локального кешу та ініціалізації запиту

```
private async Task<List<WorkItem>>
FetchWorksAsync(Professor prof)
{
    if (_cache.ContainsKey(prof.OrcidId)) return
    _cache[prof.OrcidId];

    var list = new List<WorkItem>();
    try
    {
        using var client = new HttpClient();
        client.DefaultRequestHeaders.Add("Accept",
"application/json");
        var resp = await
client.GetStringAsync($"https://pub.orcid.org/v3.0/{prof.Orcid
Id}/works");
        var data = JsonObject.Parse(resp);
```

Додатково розроблено логічний блок автогенерації відсутніх анотацій наукових публікацій. Під час звернення до серверів ORCID через метод `FetchAbstractFromOrcidAsync` система перевіряє вузол `short-description`. Якщо він виявляється порожнім, автоматично викликається внутрішній алгоритм (лістинг 3.2), який за допомогою конструкції `switch` визначає україномовний опис типу праці та конструює розгорнутий академічний реферат-заглушку.

Лістинг 3.2 – Алгоритм інтелектуальної генерації відсутньої анотації

```

    if (string.IsNullOrEmpty(annotation))
    {
        string workTypeDescription = defaultType.ToLower()
switch
    {
        "journal-article" => "наукова стаття у фаховому
періодичному виданні",
        "conference-paper" => "матеріали доповіді на
міжнародній науково-практичній конференції",
        "book" => "монографія або навчальний посібник",
        "dissertation" => "дисертаційне дослідження на
здобуття наукового ступеня",
        _ => "наукова праця"
    };

        annotation = $"Об'єкт дослідження: Публікація на тему
\"{defaultTitle}\".\n\n" +
            $"Опис: Дана робота представлена як
{workTypeDescription}. " +
            $"Вона пройшла офіційну реєстрацію та
індексацію у міжнародній базі ORCID. " +
            $"Матеріал присвячено теоретичним та
практичним аспектам дослідження зазначеної проблематики, " +
            $"розробці нових підходів та аналізу
отриманих результатів у відповідній галузі знань.\n\n" +
            $"[Примітка системи: Деталізований
авторський реферат (short-description) не був завантажений
дослідником на сервери ORCID].";

```

Таким чином, реалізований механізм обробки метаданих не лише покращує естетику візуального відображення, але й забезпечує високу точність та алгоритмічну стабільність розрахунків рекомендаційного контуру ML.NET[22].

Окремим надзвичайно важливим аспектом реалізації модуля асинхронного збору даних є забезпечення відмовостійкості системи (Fault Tolerance) при роботі з ненадійними мережевими з'єднаннями. Оскільки програма повністю залежить від зовнішнього веб-сервісу, було розроблено багаторівневу систему перехоплення та обробки виняткових ситуацій (Exception Handling).

При ініціалізації HTTP-запиту за допомогою методу `GetAsync`, система автоматично перевіряє статус-код відповіді сервера. У разі отримання кодів помилок клієнта (наприклад, 404 Not Found, якщо профіль видалено) або помилок сервера (500 Internal Server Error), спрацьовує блок `try-catch`, який запобігає аварійному завершенню (Crash) додатку [23].

Більше того, публічний API ORCID має суворі обмеження на кількість запитів за секунду (Rate Limits) для неавторизованих користувачів. Якщо додаток перевищить цей ліміт, сервер поверне статус-код 429 (Too Many Requests), і IP-адресу клієнта буде тимчасово заблоковано. Саме для вирішення цієї проблеми у класі `OrcidApiService` було реалізовано патерн «Кеш у пам'яті» (In-Memory Cache).

У ролі сховища виступає системна колекція `Dictionary<string, List<WorkItem>>`, де ключем є унікальний 16-значний ідентифікатор ORCID, а значенням – готовий масив публікацій.

Алгоритм роботи кешування діє наступним чином: при кожному виклику методу завантаження даних система спочатку виконує пошук ідентифікатора у словнику `_cache` за константний час $O(1)$.

Якщо профіль вже був завантажений під час поточної сесії роботи програми, метод миттєво повертає готову колекцію з оперативної пам'яті, повністю минаючи етап мережевого звернення. Це не лише гарантує захист від блокування з боку серверів ORCID, але й знижує навантаження на процесор, оскільки унеможливорює повторну ресурсоємну десеріалізацію JSON-пакетів.

3.3 Програмна реалізація аналітичного контуру на базі ML.NET

Інтелектуальний аналіз та ранжування наукових праць виконується в аналітичному модулі, який базується на інфраструктурі машинного навчання ML.NET. Ініціалізація алгоритму пошуку відбувається в обробнику події SearchBtn_Click. Для роботи конвеєра створюється системний об'єкт MLContext. Оскільки порівняння текстів вимагає їх математичної векторизації, у коді реалізовано створення ланцюжка трансформацій (Pipeline), що перетворює назви праць (Title) у багатовимірні числові вектори (Features) за статистичним методом TF-IDF. Фрагмент коду ініціалізації моделі ML.NET наведено у лістингу 3.3.

Лістинг 3.3 – Побудова конвеєра обробки природної мови та векторизації

```
var                dataView                =
_mlContext.Data.LoadFromEnumerable(allWorks);

var                pipeline                =
_mlContext.Transforms.Text.FeaturizeText("Features", "Title");

var model = pipeline.Fit(dataView);

var transformedData = model.Transform(dataView);

var                predictions                =
_mlContext.Data.CreateEnumerable<WorkPrediction>(transformedDa
ta, reuseRowObject: false).ToArray();

var queryFeatures = predictions[0].Features;
```

Варто детальніше розглянути внутрішню архітектуру конвеєра перетворень (Transformation Pipeline), який ініціалізується методом FeaturizeText. У фреймворку ML.NET цей метод не є монолітною функцією, а являє собою складний композитний трансформатор, що послідовно виконує шість фундаментальних етапів попередньої обробки природної мови (NLP).

На першому етапі (Text Normalization) весь вхідний текст назв наукових публікацій приводиться до нижнього регістру, а також відбувається автоматичне видалення діакритичних знаків та специфічних типографських символів. Це необхідно для того, щоб слова «Система» та «система» сприймалися алгоритмом як ідентичні токени.

На другому етапі (Tokenization) суцільний текстовий рядок розбивається на масив окремих слів (токенів). Розділювачами за замовчуванням виступають пробіли та знаки пунктуації.

На третьому етапі (Stop Words Removal) відбувається фільтрація масиву токенів. Метод автоматично застосовує вбудовані словники стоп-слів (прийменники, сполучники, займенники), видаляючи їх з вибірки, оскільки вони зустрічаються майже в кожному документі і не несуть унікального семантичного навантаження.

На четвертому етапі генеруються N-грами (N-grams). На відміну від класичного підходу, який аналізує лише окремі слова (уніграми), алгоритм ML.NET паралельно створює біграми та триграми (послідовності з двох або трьох слів). Це дозволяє системі не розривати стійкі наукові терміни, такі як «штучний інтелект» або «база даних», зберігаючи їхню цілісну семантику.

Конвеєр застосовує математичну модель TF-IDF (Term Frequency-Inverse Document Frequency) на фінальному етапі до сформованого словника N-грам [24]. Кожному унікальному терміну присвоюється вага, яка відображає його значущість у межах конкретної публікації відносно всього завантаженого корпусу праць кафедри. Результатом роботи цього комплексного конвеєра є об'єкт IDataView, який містить нову колонку Features (Ознаки). Ця колонка являє собою розріджений вектор (Sparse Vector) типу float[], розмірність якого дорівнює кількості унікальних термінів у всьому корпусі.

Після завершення векторизації навчена модель застосовується до всього корпусу праць, що міститься у колекції allWorks. Обчислення семантичної релевантності між вектором пошукового запиту (queryFeatures)

та іншими працями виконується шляхом розрахунку косинусної подібності. Відповідна математична логіка інкапсульована в окремому методі `CalculateCosineSimilarity` (лістинг 3.4), який розраховує скалярний добуток багатовимірних масивів і нормалізує його на добуток їхніх евклідових довжин. Використання саме цієї просторової метрики зумовлено її незалежністю від абсолютної довжини аналізованих текстів, що є критично важливим при порівнянні коротких назв публікацій із розгорнутими монографіями. Такий підхід гарантує об'єктивну оцінку семантичної близькості наукових матеріалів незалежно від обсягу та повноти їхнього бібліографічного опису.

Лістинг 3.4 – Програмна реалізація алгоритму обчислення косинусної подібності

```
private float CalculateCosineSimilarity(float[]
vector1, float[] vector2)
{
    float dotProduct = 0;
    float magnitude1 = 0;
    float magnitude2 = 0;

    for (int i = 0; i < vector1.Length; i++)
    {
        dotProduct += vector1[i] * vector2[i];
        magnitude1 += vector1[i] * vector1[i];
        magnitude2 += vector2[i] * vector2[i];
    }

    if (magnitude1 == 0 || magnitude2 == 0) return 0;
    return dotProduct / (float) (Math.Sqrt(magnitude1)
* Math.Sqrt(magnitude2)); }
```

Усі елементи, які в результаті обчислень долають встановлене порогове значення релевантності (`sim >= 0.20f`), зберігаються у списку

recommendations. Такий поріг відсічення дозволяє ефективно відфільтрувати інформаційний шум, залишаючи у вибірці лише об'єктивно схожі праці. Фінальним кроком є їх сортування за спаданням індексу релевантності (OrderByDescending), що гарантує відображення найбільш відповідних публікацій на початку списку. Після цього сформована колекція передається до графічного компонента RecommendationsList через механізм прив'язки даних WPF, що забезпечує швидке оновлення інтерфейсу користувача без блокування головного потоку програми.

3.4 Інтерфейс користувача додатку WPF та аналіз результатів тестування

Графічна оболонка розробленої інтелектуальної системи рекомендацій побудована на базі технології Windows Presentation Foundation (WPF) з використанням мови декларативної розмітки XAML. Такий підхід дозволив чітко відокремити логіку обробки даних (C# Code-Behind) від візуального представлення, а також ефективно застосувати механізми прив'язки даних (Data Binding) [25] для динамічного оновлення інформації на екрані без блокування головного потоку додатку.

Головне вікно програми (MainWindow) логічно поділене на кілька функціональних зон. У верхній частині розташовано панель управління з випадаючим списком ComboBox для вибору викладача кафедри зі списку попередньо завантажених профілів. Ліва панель (AllWorksList) відповідає за виведення повного переліку наукових праць обраного дослідника. Права панель (RecommendationsList) призначена для відображення результатів роботи аналітичного контуру ML.NET – відсортованого списку релевантних публікацій[3].

Керування режимами пошуку здійснюється за допомогою групи перемикачів RadioButton (пошук по всій кафедрі, по обраному викладачу, або пошук схожих праць на основі вже існуючої статті).

Для реалізації механізму відображення масиву даних використовується потужна концепція `DataTemplate`. Вона дозволяє системі автоматично генерувати візуальні блоки для кожного об'єкта моделі, що знаходиться у колекції-джерелі. Фрагмент XAML-розмітки, який відповідає за виведення списку знайдених наукових рекомендацій у правій панелі, наведено у лістингу 3.5.

Лістинг 3.5 – Декларативна розмітка візуалізації списку рекомендацій у WPF

```
<ListBox x:Name="RecommendationsList"
        ItemsSource="{Binding RecommendedWorks}"
        Margin="10"
        BorderBrush="#FFCCCC"
        BorderThickness="1"

        MouseDoubleClick="RecommendationsList_MouseDoubleClick">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <Border BorderBrush="#EEEEEE"
                BorderThickness="0,0,0,1" Padding="5">
                <StackPanel Orientation="Vertical">
                    <TextBlock Text="{Binding Title}"
                        FontWeight="Bold"
                        TextWrapping="Wrap"
                        Foreground="#FF2C3E50"/>
                    <TextBlock Text="{Binding Type}"
                        FontStyle="Italic"
                        Foreground="#FF7F8C8D"
                        Margin="0,5,0,0"/>
                    <TextBlock Text="{Binding SimilarityScore,
                        StringFormat='Релевантність: {0:P2}'}"
                        Foreground="#FF27AE60"
                        FontWeight="SemiBold"
                        Margin="0,5,0,0"/>
                </StackPanel>
            </Border>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

Продовження лістингу 3.5

```

        </StackPanel>
    </Border>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>

```

У наведеному лістингу властивість `ItemsSource` прив'язана до колекції `RecommendedWorks`. Кожен елемент списку відображається у вигляді набору текстових блоків `TextBlock`, текст яких динамічно формується з властивостей об'єкта моделі (зокрема, `Title` та `Type`). Додатково використовується форматування рядка `StringFormat='Релевантність: {0:P2}'`, що дозволяє автоматично конвертувати числове значення косинусної подібності у зручний для сприйняття відсотковий формат.

Для забезпечення максимального комфорту користувача (`User Experience`) в нижній частині вікна реалізовано динамічний рядок стану (`StatusBar`), який у реальному часі інформує про поточний статус системи: процеси завантаження, аналізу або виникнення помилок мережі.

Окремим важливим ергономічним рішенням є автоматизація переходу до першоджерела публікації. У системі реалізовано обробник події подвійного кліку миші (`MouseDoubleClick`) на елементі списку рекомендацій. Програма автоматично валідує веб-посилання обраного об'єкта та за допомогою системного процесу миттєво відкриває сторінку статті (або прямий PDF-документ) у дефолтному браузері операційної системи.

Фрагмент програмної реалізації цієї функції наведено у лістингу 3.6.

Лістинг 3.6 – Реалізація обробника події переходу до публікації

```

private void RecommendationsList_MouseDoubleClick(object
sender, System.Windows.Input.MouseButtonEventArgs e)
{

```

Продовження лістингу 3.6

```

        if (RecommendationsList.SelectedItem is
RecommendedWork selectedRec)
        {
            if (!string.IsNullOrEmpty(selectedRec.Url) &&
selectedRec.Url.StartsWith("http"))
            {
                try
                {
                    // Перевіряємо, чи це пряме посилання на
PDF файл
                    if (selectedRec.Url.EndsWith(".pdf",
StringComparison.OrdinalIgnoreCase) ||
selectedRec.Url.Contains(".pdf"))
                    {
                        StatusText.Text = "Відкриття PDF-
документа у браузері...";
                    }
                    else
                    {
                        StatusText.Text = "Перехід на веб-
сторінку публікації...";
                    }

                    System.Diagnostics.Process.Start(new
System.Diagnostics.ProcessStartInfo
                    {
                        FileName = selectedRec.Url,
                        UseShellExecute = true
                    });
                }
                catch
                {

```

Продовження лістингу 3.6

```

        MessageBox.Show("Не вдалося відкрити
        посилання автоматично.", "Помилка", MessageBoxButton.OK,
        MessageBoxImage.Error);

        StatusText.Text = "Помилка відкриття
        посилання.";

    }

}

else
{

    MessageBox.Show("Для цієї праці відсутнє
    коректне веб-посилання.", "Увага", MessageBoxButton.OK,
    MessageBoxImage.Warning);

}

}

}

```

Комплексне тестування розробленого програмного забезпечення проводилося на реальних профілях 12 викладачів кафедри штучного інтелекту. Результати експериментальних досліджень підтвердили високу стабільність системи. Час відгуку аналітичного контуру ШІ при обробці масиву з понад сотні публікацій становить лічені мілісекунди. Завдяки розробленій системі кешування Dictionary програма продемонструвала абсолютну стійкість до лімітів запитів API ORCID та здатність миттєво перемикатися між профілями. Алгоритми автогенерації успішно впоралися з відсутніми анотаціями, запобігши виникненню критичних збоїв парсингу, а механізм TF-IDF довів свою ефективність, формуючи високоточні та семантично близькі рекомендації наукових праць.

Окрема увага під час тестування приділялася перевірці відмовостійкості архітектури в умовах роботи з неповними або нестандартно заповненими даними. Зокрема, спроектований трирівневий алгоритм видалення імен авторів безпомилково ідентифікував профілі

навіть за відсутності базових полів, автоматично підтягуючи альтернативні варіанти написання з додаткових масивів метаданих. Крім того, перевірка багатопотокового виконання підтвердила, що асинхронне завантаження HTTP-запитів повністю ізольоване від головного потоку додатку, завдяки чому графічний інтерфейс WPF залишався абсолютно плавним і чутливим до дій користувача навіть у моменти пікового навантаження на мережу.

Якісний аналіз згенерованих рекомендацій показав, що математичний апарат ML.NET успішно виявляє приховані семантичні зв'язки між публікаціями різних наукових груп. Наприклад, статті із суміжних тем, які використовували різну термінологію, впевнено долали встановлене порогове значення косинусної подібності [11] у 0.20 та потрапляли до топу результуючої вибірки, що на практиці підтвердило перевагу векторизації над класичним лексичним пошуком. Моніторинг апаратних ресурсів під час локальних обчислень зафіксував мінімальне споживання оперативної пам'яті, що доводить доцільність використання вбудованого керованого коду. У підсумку, програмний комплекс успішно пройшов усі етапи верифікації, підтвердивши свою готовність до повноцінного впровадження у щоденну науково-дослідну діяльність кафедри як надійний інструмент для автоматизації бібліографічного пошуку та внутрішнього аудиту.

4 СТАБІЛЬНІСТЬ РЕЗУЛЬТАТІВ І УЗАГАЛЬНЕННЯ

Одним із ключових аспектів оцінки розробленої інтелектуальної системи рекомендацій наукових публікацій є перевірка стабільності її роботи в умовах реального використання та здатності до узагальнення алгоритмів машинного навчання [18] на різноманітних наборах даних. Оскільки система взаємодіє з відкритим веб-сервісом ORCID [13] і обробляє неструктуровані текстові потоки, критично важливо було забезпечити безперебійність графічного інтерфейсу та сталість результатів пошуку незалежно від повноти заповнення профілів викладачів.

Для забезпечення високого рівня стабільності на етапі завантаження даних було впроваджено локальний словник-кеш (_cache). Контрольовані експерименти показали, що при первинному зверненні до серверів ORCID [13] для завантаження масиву з 50-100 публікацій система витрачає в середньому від 1,5 до 3 секунд, що зумовлено мережевими затримками. Однак завдяки механізму кешування, при повторних зверненнях до вже завантаженого профілю або під час глобального пошуку по всій кафедрі час відгуку системи скорочується до 10–15 мілісекунд. Це не лише гарантує миттєву реакцію інтерфейсу WPF [11], але й захищає додаток від можливого блокування через перевищення лімітів частоти запитів до Public API ORCID (Rate Limits).

Здатність системи до узагальнення особливо яскраво проявляється в алгоритмах обробки відсутніх метаданих. Профілі дослідників в ORCID [13] часто є неоднорідними: деякі публікації мають повний опис та DOI-ідентифікатори, тоді як інші містять лише назву. Впроваджений механізм розумної автогенерації анотацій та адаптивного формування веб-посилань дозволив системі узагальнювати розрізнені дані. Програма стабільно обробляє будь-які публікації (від журнальних статей journal-article до монографій book), зводячи їх до єдиної внутрішньої моделі WorkItem [17] без жодного збою парсингу (NullReferenceException).

Окремої уваги заслуговує стабільність аналітичного контуру на базі ML.NET [12]. На відміну від жорстко запрограмованих алгоритмів пошуку, конвеєр обробки природної мови `Transforms.Text.FeaturizeText` [15] генерує векторний простір динамічно для кожного нового набору праць. Це означає, що система володіє абсолютною здатністю до узагальнення: математична модель TF-IDF (Term Frequency-Inverse Document Frequency) [4], [10] однаково ефективно ранжує публікації як з технічних, так і з гуманітарних чи природничих дисциплін, незалежно від мови публікації (англійської чи української).

Для перевірки стабільності результатів семантичного ранжування було проведено серію експериментів з використанням порогового значення косинусної подібності [8] ($\text{sim} = 0.20$). У таблиці 4.1 наведено усереднені показники.

Таблиця 4.1 – Продуктивність та стабільність аналітичного контуру ML.NET

Режим пошукового запиту	Кількість аналізованих праць	Середній час обробки, мс
Локальний (один викладач)	45	12
Локальний (один викладач)	120	28
Глобальний (вся кафедра)	450	85
Глобальний (вся кафедра)	850	160

Як видно з таблиці 4.1, час розрахунку косинусної подібності [9] між вектором запиту та корпусом публікацій зростає лінійно, однак навіть при обробці великих масивів даних (близько тисячі праць) залишається в межах 160 мілісекунд. Це підтверджує високу алгоритмічну стабільність системи та відсутність експоненційного зростання обчислювальної складності.

Крім того, система продемонструвала стійкість результатів при введенні нестандартних або занадто коротких пошукових запитів. Завдяки обраному порогу подібності 0.20, алгоритм відсіює інформаційний шум. Якщо користувач вводить загальне слово (наприклад, «аналіз» або «система»), алгоритм повертає широку вибірку. Якщо ж запит є вузькоспеціалізованим (наприклад, «нейронні мережі LSTM» [20]), косинусна подібність чітко фокусується лише на релевантних працях, залишаючи оцінку інших близькою до нуля.

ВИСНОВКИ

Дане дослідження було присвячене розробці комплексної інтелектуальної системи рекомендацій наукових публікацій на базі профілів міжнародного реєстру ORCID для автоматизації інформаційного супроводу та аналізу публікаційної активності кафедри штучного інтелекту. Результати проведеної роботи демонструють успішне досягнення поставлених цілей та відкривають широкі перспективи для подальшого вдосконалення і розширення розробленого підходу.

У ході дослідження було реалізовано ефективний підхід до організації процесів збору, структурування та аналізу бібліографічних даних викладачів кафедри. Було створено програмний інструментарій для автоматичного завантаження метаданих через Public API ORCID, їх асинхронної обробки та накопичення в локальних структурах даних для подальшого порівняльного семантичного аналізу. Застосування сучасних методів обробки природної мови (NLP) та машинного навчання дозволило значно підвищити точність та релевантність відбору наукових матеріалів без необхідності складного ручного налаштування пошукових запитів. Система була побудована таким чином, щоб забезпечити високу гнучкість, відмовостійкість та масштабованість експериментів.

Серцевиною розробленої системи став аналітичний контур на базі фреймворку ML.NET, який виконує автоматичну статистичну векторизацію назв праць за методом TF-IDF. Отримані багатовимірні числові вектори ознак використовуються для розрахунку косинусної подібності між пошуковим запитом (або обраною статтею) та всім корпусом публікацій кафедри. Такий підхід дозволив системі самостійно знаходити приховані семантичні зв'язки між роботами, точно ранжувати результати та відсіювати нерелевантний інформаційний шум за гнучким пороговим значенням.

Впровадження розробленої системи дозволяє повністю нівелювати вплив людського фактору при пошуку наукових партнерів та схожих

досліджень у межах кафедри, значно підвищити якість інформаційного моніторингу та скоротити час обробки великих масивів бібліографії до лічених мілісекунд. Програмна реалізація графічного інтерфейсу на базі WPF забезпечила високу зручність взаємодії завдяки підтримці трьох режимів пошуку, механізмам локального кешування для захисту від лімітів API та модулю інтелектуальної автогенерації рефератів-заглушок для неповних профілів дослідників.

Подальші напрямки розвитку системи включають розширення набору джерел первинних даних (зокрема, інтеграцію з API Google Scholar, Scopus та Web of Science), а також впровадження більш складних моделей представлення текстів, таких як трансформери (BERT, RoBERTa) чи контекстні ембеддинги для ще точнішого вловлювання семантики наукових праць. Перспективним є також додавання аналітики співавторства та побудова графів наукових зв'язків кафедри для візуалізації дослідницьких напрямів університету.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Волков К. А., Гриньова О. Є. Інтелектуальна система рекомендацій на основі NLP-аналізу ORCID-публікацій для кафедри ШІ. Радіoeлектроніка та молодь у ХХІ столітті : матеріали 30-го Міжнар. молодіж. форуму, 22–24 квітня 2026 р. Харків : ХНУРЕ, 2026. Т. 6. С. 19–21.
- 2 Глибовець М. М., Олецкий О. В. Штучний інтелект : підручник. Київ : КМ Академія, 2002. 366 с.
3. Aggarwal C. C. Recommender systems: the textbook. Cham : Springer, 2016. 498 p.
4. Ключниченко В. А. Методи та системи штучного інтелекту : навч. посібник. Харків : ХНУРЕ, 2021. 180 с.
5. ORCID Public API v3.0 Documentation. *ORCID Support*. 2025. URL: <https://info.orcid.org/documentation/api-tutorials/> (date of access: 28.05.2026).
6. Singhal A. Modern information retrieval: a brief overview. *IEEE data engineering bulletin*. 2001. Vol. 24, No. 4. P. 35–43.
7. Windows presentation foundation (WPF) documentation. *Microsoft Learn*. 2026. URL: <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/> (date of access: 02.05.2026).
8. ML.NET: machine learning for .NET. *Microsoft Learn*. 2026. URL: <https://learn.microsoft.com/en-us/dotnet/machine-learning/> (date of access: 14.05.2026).
9. Бодянський Є. В., Руденко О. Г. Штучні нейронні мережі: архітектури, навчання, застосування : навч. посібник. Харків : ТЕЛЕТЕХ, 2004. 312 с.
10. Json.NET. *Newtonsoft*. 2025. URL: <https://www.newtonsoft.com/json> (date of access: 01.06.2026).

11. Lops P., de Gemmis M., Semeraro G. Content-based recommender systems: state of the art and trends. *Recommender systems handbook*. Boston : Springer, 2011. P. 73–105.
12. Transforms.Text.FeatureizeText method. *Microsoft Learn*. 2026. URL: https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.textcatalog.featureize_text (date of access: 04.06.2026).
13. Richards M., Ford N. Fundamentals of software architecture: an engineering approach. O'Reilly Media, 2020. 432 p.
14. Richardson L., Ruby S. RESTful web services. O'Reilly Media, 2007. 454 p.
15. HttpClient Class. *Microsoft Learn*. 2026. URL: <https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient> (date of access: 10.06.2026).
16. Nygard M. T. Release it!: design and deploy production-ready software. 2nd ed. Pragmatic Bookshelf, 2018. 376 p.
17. Jurafsky D., Martin J. H. Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition. 3rd ed. Pearson, 2023. 624 p.
18. Salton G., McGill M. J. Introduction to modern information retrieval. New York : McGraw-Hill, 1983. 448 p.
19. Martin R. C. Clean architecture: a craftsman's guide to software structure and design. Prentice Hall, 2017. 432 p.
20. Troelsen A., Japikse P. Pro C# 10 with .NET 6: foundational principles and practices in programming. 11th ed. Apress, 2022. 1438 p.
21. Cleary S. Concurrency in C# Cookbook: asynchronous, parallel, and multithreaded programming. 2nd ed. O'Reilly Media, 2019. 254 p.
22. Lerman G. WPF 4.5 unleashed. Sams Publishing, 2013. 864 p.
23. Capellman J. Hands-on machine learning with ML.NET. Birmingham : Packt Publishing, 2020. 336 p.

24. Manning C. D., Raghavan P., Schütze H. Introduction to information retrieval. Cambridge University Press, 2008. 496 p.

25. Data Binding and MVVM in WPF. *Microsoft Learn*. 2026. URL: <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/data/> (date of access: 15.06.2026).