

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук
(повна назва)
Кафедра _____ Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____ перший (бакалаврський)

Інтелектуальна система синтезу персональних освітніх планів засобами
семантичного мапінгу, трансформерних NLP-моделей та онтологічного
інжинірингу
(тема)

Виконав:
здобувач _____ четвертого _____ року навчання,
групи _____ ІТШ-22-1

_____ Володимир ВОЛОДІН
(власне ім'я, прізвище)

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

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

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

Допускається до захисту

Завідувач кафедри ШІ

_____ Лариса ЧАЛА
(підпис) (власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук _____

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 _____ Володіну Володимиру Андрійовичу _____
(прізвище, ім'я, по батькові)

1. Тема роботи _____ Інтелектуальна система синтезу персональних освітніх планів засобами семантичного мапінгу, трансформерних NLP-моделей та онтологічного інжинірингу _____

затверджена наказом університету від 5 червня 2026 р. № 488Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 9 липня 2026 р.

3. Вихідні дані до роботи _____ Силабуси навчальних дисциплін освітніх програм кафедри штучного інтелекту ХНУРЕ, структурно-логічна схема програм (передумови та компетенції дисциплін), онтологія освітньо-професійної програми, попередньо натренована трансформерна модель MiniLM. _____

4. Перелік питань, що потрібно опрацювати в роботі _____

1) Аналіз предметної галузі та постановка задачі _____

2) Обґрунтування технологій, методів та архітектури при проєктуванні системи _____

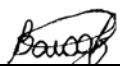
3) Програмна реалізація та алгоритмічне забезпечення системи _____

4) Інтерфейс користувача та тестування системи _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.06.2026	виконано
2	Аналіз предметної галузі	09.06.2026	виконано
3	Огляд та порівняльний аналіз існуючих рішень	09.06.2026	виконано
4	Постановка задачі та проєктування системи	10.06.2026	виконано
5	Формування онтологічної бази знань	10.06.2026	виконано
6	Програмна реалізація модулів системи	11.06.2026	виконано
7	Розроблення інтерфейсу та тестування системи	18.06.2026	виконано
8	Написання пояснювальної записки	22.06.2026	виконано
9	Перевірка на академічний плагіат	23.06.2026	виконано
10	Нормоконтроль	25.06.2026	виконано
11	Підготовка презентації та доповіді	27.06.2026	виконано
12	Попередній захист, рецензування	02.06.2026	виконано
13	Захист	09.07.2026	

Дата видачі завдання 8 червня 2026 р.

Здобувач 
(підпис)

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

РЕФЕРАТ

Пояснювальна записка: 101 с., 10 рис., 2 табл., 7 дод., 22 джерела.

ВЕБ-ОРІЄНТОВАНА СИСТЕМА, ГРАФ ПЕРЕДУМОВ, ОНТОЛОГІЧНИЙ ІНЖИНІРИНГ, ОСВІТНІЙ ПЛАН, ПЕРСОНАЛІЗАЦІЯ НАВЧАННЯ, ПОЯСНЮВАНИЙ ШТУЧНИЙ ІНТЕЛЕКТ, СЕМАНТИЧНИЙ МАПІНГ, ТОПОЛОГІЧНЕ СОРТУВАННЯ, ТРАНСФОРМЕРНІ МОДЕЛІ.

Об'єкт дослідження – процес автоматизованого синтезу персональних освітніх планів у сфері інформаційних технологій.

Предмет дослідження – методи та засоби семантичного аналізу неформалізованих запитів природною мовою й алгоритми обходу графів передумов на основі онтологічних моделей знань.

Мета роботи – розробка та програмна реалізація інтелектуальної системи, яка на основі довільного запиту користувача природною мовою формує логічно впорядкований персональний план вивчення дисциплін із поясненням кожного кроку.

Методи дослідження – онтологічний інжиніринг засобами OWL і ризонера HermiT, обробка природної мови на основі трансформерної моделі-енкодера, графові алгоритми (обхід у глибину, топологічне сортування), принципи пояснюваного штучного інтелекту.

Розроблено OWL-онтологію освітньо-професійної програми кафедри штучного інтелекту ХНУРЕ та NLP-конвеєр семантичного зіставлення запитів з її концептами. Графовий модуль будує персональний план, а механізм пояснення формує верифіковані шаблонні траси виведення – на відміну від генеративних пояснень. Компоненти інтегровано у веб-орієнтований комплекс з інтерактивною візуалізацією графа передумов.

ABSTRACT

Bachelor's thesis contains: 101 p., 10 fig., 2 tabl., 7 ann., 22 references.

EDUCATIONAL PLAN, EXPLAINABLE ARTIFICIAL INTELLIGENCE, LEARNING PERSONALIZATION, ONTOLOGICAL ENGINEERING, PREREQUISITE GRAPH, SEMANTIC MAPPING, TOPOLOGICAL SORTING, TRANSFORMER MODELS, WEB-BASED SYSTEM.

The object of research is the process of automated synthesis of personalized educational plans in the field of information technology.

The subject of research is the methods and tools of semantic analysis of unstructured natural-language queries and algorithms for traversing prerequisite graphs based on ontological knowledge models.

The aim of the work is the development and software implementation of an intelligent system that, based on an arbitrary natural-language query, generates a logically ordered personal learning plan with an explanation of the disciplines' study sequence.

Research methods – ontological engineering using OWL and the HermiT reasoner, natural language processing based on a transformer encoder model, graph algorithms (depth-first traversal, topological sorting), explainable AI principles.

An OWL ontology of the educational programme of the NURE Department of Artificial Intelligence, an NLP pipeline for semantic matching of queries with ontology concepts, and a graph module for building personal learning plans were developed. An explanation module based on template-based inference traces was implemented, ensuring result verifiability – unlike generative explanations. The components were integrated into a web-based system with interactive prerequisite-graph visualization.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	8
Вступ.....	9
1 Аналіз предметної галузі та постановка задачі.....	11
1.1 Сучасний стан предметної галузі	11
1.2 Огляд та порівняльний аналіз існуючих рішень.....	13
1.2.1 Платформи масового онлайн-навчання.....	13
1.2.2 Платформи з ринково-орієнтованою персоналізацією	14
1.2.3 Великі мовні моделі загального призначення	15
1.2.4 Академічні онтологічні системи	15
1.2.5 Гібридні нейро-символьні системи.....	17
1.2.6 Порівняльний аналіз розглянутих рішень.....	18
1.3 Мета роботи та постановка задачі.....	19
2 Обґрунтування технологій, методів та архітектури при проєктуванні системи	22
2.1 Загальна архітектура системи	22
2.2 Модель онтології освітньо-професійної програми.....	23
2.3 Проєктування конвеєра обробки природномовного запиту	27
2.4 Проєктування модуля побудови плану.....	30
2.5 Проєктування модуля пояснення	32
3 Програмна реалізація та алгоритмічне забезпечення системи.....	34
3.1 Засоби та технології розробки	34
3.2 Формування онтологічної бази знань	35
3.2.1 Парсинг силабусів.....	35
3.2.2 Побудова онтології та нечітке зіставлення передумов	36
3.2.3 Перевірка узгодженості та індекс ембедингів	39
3.3 Реалізація модуля обробки запиту	40
3.4 Реалізація модуля побудови плану.....	42
3.5 Реалізація модуля пояснення	45

4 Інтерфейс користувача та тестування системи	47
4.1 Інтерфейс користувача	47
4.2 Демонстрація роботи системи	49
4.2.1 Звичайний запит.....	49
4.2.2 Запит із відомою ціллю	51
4.2.3 Обробка некоректного запиту	52
4.3 Тестування та оцінювання	53
Висновки	55
Перелік джерел посилання	57
Додаток А Код парсингу силабусів.....	60
Додаток Б Код формування онтології.....	70
Додаток В Код побудови індексу ембедингів.....	78
Додаток Г Код обробки запиту	81
Додаток Д Код побудови плану.....	89
Додаток Е Код формування пояснень.....	96
Додаток Ж Відомість кваліфікаційної роботи	101

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОПП – освітньо-професійна програма;

ШІ – штучний інтелект;

DAG – Directed Acyclic Graph – орієнтований ациклічний граф;

DFS – Depth-First Search – пошук у глибину;

LLM – Large Language Model – велика мовна модель;

NER – Named Entity Recognition – розпізнавання іменованих сутностей;

NLP – Natural Language Processing – обробка природної мови;

OWL – Web Ontology Language – мова веб-онтологій;

RDF – Resource Description Framework – модель опису ресурсів;

SPARQL – SPARQL Protocol and RDF Query Language – мова запитів до RDF-даних;

XAI – Explainable Artificial Intelligence – пояснюваний штучний інтелект.

ВСТУП

Галузь інформаційних технологій характеризується одним із найвищих темпів технологічного оновлення серед сучасних галузей: цикл оновлення базового стеку розробки скорочується до 18–24 місяців, а машинне навчання, обробка природної мови та інженерія даних щороку набувають статусу самостійних освітніх напрямів зі своїми вимогами. У цих умовах здобувач, який прагне опанувати конкретний напрям ІТ, постає перед задачею ширшою, ніж власне навчання: серед значної кількості курсів і дисциплін потрібно визначити мінімально достатню послідовність вивчення, що приведе до бажаної компетенції, оскільки опанування спеціалізованої дисципліни без попереднього засвоєння фундаментальних блоків знижує якість розуміння матеріалу.

Формальним джерелом інформації про коректну послідовність вивчення є освітньо-професійна програма (ОПП) у вигляді табличного переліку дисциплін і передумов. Для здобувача без професійної підготовки такий формат залишається малозручним: повна схема містить десятки взаємопов'язаних компонентів, а визначити в ній цільовий маршрут вручну непросто. Додатковим бар'єром є термінологічна неузгодженість між лексикою, якою здобувач формулює освітню мету, та академічною номенклатурою дисциплін.

Існуючі підходи до автоматизації побудови освітніх маршрутів – платформи онлайн-навчання, рекомендаційні системи, великі мовні моделі загального призначення та формалізовані онтологічні моделі – частково вирішують поставлену задачу, проте жоден не поєднує одночасно гнучкий природномовний інтерфейс, верифікованість рекомендацій та строгий облік передумов [1]. Перспективним напрямом є гібридні архітектури, у яких трансформерні NLP-моделі забезпечують зв'язок між неформалізованим запитом і формальним поданням знань, а онтологічна компонента відповідає за коректність результату [2], [3], доповнені реалізацією

принципу пояснюваного штучного інтелекту [4]. Підґрунтя для подібних розробок у вітчизняній науковій традиції закладено дослідженнями концептуальних графів навчальних курсів [5].

Актуальність роботи зумовлена потребою в інструменті, що автоматизує побудову персонального освітнього плану на основі довільно сформульованого запиту, гарантуючи логічну впорядкованість плану та прозорість його обґрунтування. Об'єктом дослідження є процес автоматизованого синтезу персональних освітніх планів у сфері інформаційних технологій, а предметом – методи та засоби семантичного аналізу неформалізованих запитів природною мовою й алгоритми обходу графів передумов на основі онтологічних моделей знань.

Метою роботи є розробка та програмна реалізація інтелектуальної системи, що на основі довільного запиту користувача природною мовою формує логічно впорядковану послідовність дисциплін із поясненням кожного кроку. Досягнення мети передбачало проєктування онтологічної моделі ОПП засобами OWL, реалізацію NLP-конвеєра семантичного зіставлення запиту з концептами онтології, розробку графового модуля виявлення передумов і побудови плану, реалізацію механізму пояснення рекомендацій на основі формальних зв'язків онтології та інтеграцію компонентів у веб-орієнтований програмний комплекс із подальшою експериментальною перевіркою.

Практичне значення роботи полягає у можливості застосування реалізованої системи як навігатора по освітній програмі – здобувачами під час побудови індивідуального плану та кафедрою для автоматизованого консультування абітурієнтів щодо структури ОПП.

Результати роботи оприлюднено в тезах доповіді на конференції «Радіотехніка та молодь у XXI столітті» [6].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Сучасний стан навчання в галузі інформаційних технологій

Інформаційні технології залишаються однією з галузей із найбільшим попитом на фахівців. В Україні ІТ-сфера також є одним із провідних напрямів, що приваблює людей з інших професій. Водночас технології в цій галузі швидко змінюються: мови програмування, бібліотеки та інструменти оновлюються кожні кілька років. Через це навчання для ІТ-фахівця не закінчується разом із дипломом – щоб лишатися затребуваним, він мусить опановувати нове постійно.

Самостійне навчання вже стало звичним шляхом у галузь, а не винятком. За даними щорічного опитування розробників Stack Overflow, лише близько половини з них опанували програмування у межах формальної освіти, тоді як переважна більшість користувалася онлайн-ресурсами – документацією, відеокурсами та навчальними платформами [7]. Те саме опитування показує, що диплом з комп'ютерних наук уже не є єдиним шляхом у професію: дедалі більше людей навчаються гнучко, самостійно добираючи матеріал під конкретну потребу. Тобто мільйони людей самі складають собі програму навчання, не маючи поруч викладача, який підказав би послідовність.

Доступність навчальних матеріалів за останні роки різко зросла. Онлайн-платформи накопичили величезні каталоги курсів майже з будь-якої теми, а поряд із ними діють відеоуроки, інтерактивні тренажери та професійна документація. Наприклад, платформа Coursera на кінець 2025 року налічувала близько 197 мільйонів зареєстрованих слухачів, а щороку до неї приєднується понад двадцять мільйонів нових [8]. Онлайн-навчання перестало бути тимчасовим явищем і перетворилося на сталий масовий формат.

Унаслідок цього змінився характер головної проблеми. Раніше складно було знайти якісний матеріал – тепер його радше забагато. Людина, що хоче опанувати певний напрям, стикається із сотнями курсів і не знає, з чого почати, що з цього справді потрібне та в якому порядку це вивчати. Наприклад, охочий навчитися машинного навчання бачить окремі курси з програмування, статистики, лінійної алгебри та нейронних мереж, але сам не визначить, що вивчати першим, а що – лише після опанування основ. Без орієнтиру початківець або витрачає час на зайве, або береться за складну тему, не маючи потрібної підготовки.

Неправильно обраний порядок навчання дорого коштує. Час, витрачений на непотрібний чи надто складний курс, не повертається, а постійні невдачі знижують мотивацію – частина початківців через це взагалі полишає навчання. Тому інструмент, який підказав би людині розумну послідовність кроків саме під її мету, має не лише навчальну, а й практичну цінність.

Для організованого навчання відповідь на питання послідовності вже існує. Заклади вищої освіти описують зміст підготовки в освітньо-професійній програмі. Така програма перелічує дисципліни й указує, що треба вивчити раніше, а що пізніше, тобто фіксує передумови між дисциплінами. Подібну програму має, зокрема, кафедра штучного інтелекту ХНУРЕ. Проте освітньо-професійна програма складена для зарахованого студента й подана у вигляді формальних таблиць. Сторонній людині важко самостійно вчитати з неї саме свій шлях до конкретної мети.

Отже, склалася суперечлива ситуація. З одного боку, навчального матеріалу вдосталь, а формальні описи послідовності дисциплін існують. З іншого – звичайна людина не може легко перетворити бажання «хочу навчитися певної технології» на чіткий перелік курсів у правильному порядку. Саме це й намагаються розв'язати різні інструменти – від навчальних платформ до систем на основі штучного інтелекту. Детальний

порівняльний аналіз цих інструментів за здатністю розв'язувати окреслену задачу наведено в підрозділі 1.2.

1.2 Огляд та порівняльний аналіз існуючих рішень

Завдання, окреслене в підрозділі 1.1, намагаються розв'язати рішення кількох типів. Щоб порівняти їх між собою, оцінимо кожне за чотирма ознаками, важливими для нашого користувача: чи приймає воно мету, сформульовану звичайними словами; чи вибудовує правильний порядок вивчення з урахуванням передумов; чи зважає на те, що людина вже знає; і чи пояснює свою пораду так, щоб її можна було перевірити. Нижче розглянуто п'ять груп рішень, а наприкінці підрозділу їх зведено в порівняльну таблицю.

1.2.1 Платформи масового онлайн-навчання

Найпоширеніший тип рішень – платформи масового онлайн-навчання, наприклад, Coursera, edX, Udemy, Pluralsight. Для побудови послідовності навчання вони використовують два механізми. Перший тип – готові набори курсів, які експерти заздалегідь склали в певному порядку (спеціалізації Coursera, програми MicroMasters на edX, навчальні шляхи Pluralsight). Другий – рекомендації за схожістю поведінки користувачів: система пропонує те, що проходили інші люди зі схожими інтересами. Приклад готової послідовності курсів показано на рисунку 1.1.

Сильні сторони таких платформ очевидні: величезний обсяг контенту, готові програми для популярних професій і визнані роботодавцями сертифікати.

Проте для нашої задачі вони мають три суттєві вади. Готові послідовності однакові для всіх – вони не враховують, що частину матеріалу людина вже знає. Рекомендації за поведінкою підбирають схоже, а не

логічно правильне за порядком: те, що два курси часто проходять разом, не означає, що один є передумовою іншого. Нарешті, платформа не пояснює, чому радить саме це, – користувач має довіряти їй на слово.

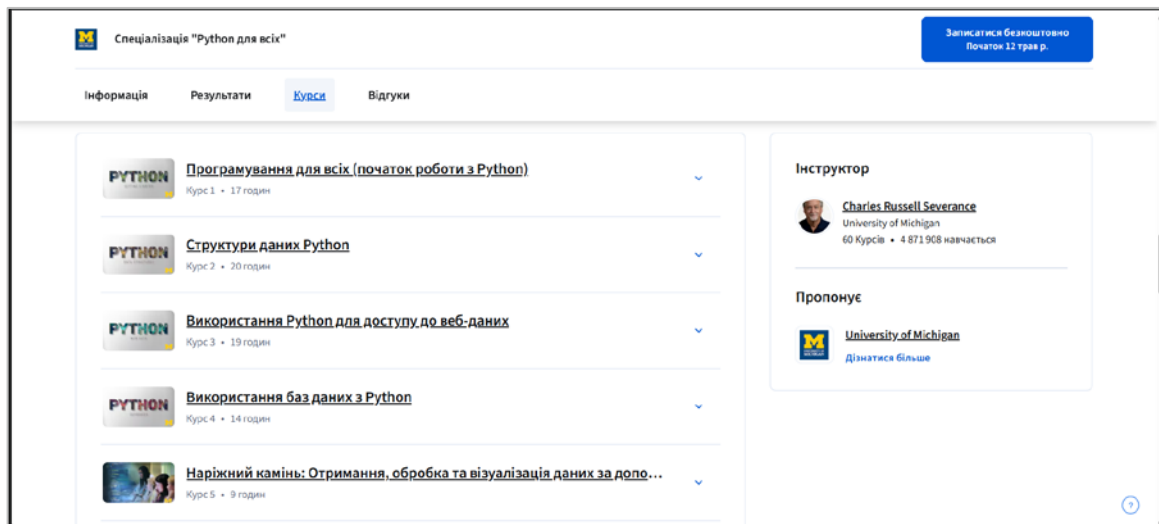


Рисунок 1.1 – Приклад готової навчальної послідовності у спеціалізації платформи Coursera

1.2.2 Платформи з ринково-орієнтованою персоналізацією

Окремий тип – рішення, що поєднують навчальний контент із даними ринку праці. Найвідоміше з них, LinkedIn Learning, аналізує вакансії та профілі фахівців і для обраної професії визначає набір потрібних навичок, а потім пропонує курси, які ці навички закривають.

Перевага підходу – актуальність, рекомендації спираються на поточні вимоги ринку, а не на сталу освітню програму.

Проте порядок засвоєння тут не вибудовується. Навички подано простим переліком, без указівки на те, що вивчати першим, а зв'язку з академічним змістом немає. Пояснення рекомендацій, як і в попередній групі, не надається.

1.2.3 Великі мовні моделі загального призначення

Після поширення великих мовних моделей (ChatGPT, Claude, Gemini) частина людей почала радитися щодо навчання саме з ними. Приваблює тут передусім те, що можна просто описати мету звичайними словами й одразу отримати розгорнуту відповідь майже з будь-якої теми.

Однак як єдиний інструмент побудови плану такі моделі мають принципову ваду. Вони складають відповідь на основі статистичних закономірностей у текстах, на яких навчалися, і не звіряють її з жодною формальною базою знань. Через це модель може впевнено написати неправду – порадити неіснуючий курс, послатися на застарілу версію технології чи порушити реальний порядок передумов. Це явище називають галюцинаціями.

Наскільки ненадійні моделі в задачах зі строгою структурою, показує бенчмарк Spider4SPARQL – набір зі 166 баз знань для перевірки, чи здатна модель правильно скласти формальний запит мовою SPARQL за описом звичайною мовою. У режимі без прикладів модель GPT-3.5 дала лише близько 8% правильних запитів, а навіть із десятима прикладами в підказці – близько 45% [1]. Для задач, де результат має бути логічно гарантованим, така точність недостатня.

Окрема вада стосується пояснень. Текст, яким модель обґрунтовує свою пораду, – це продукт тієї самої генерації, що й сама порада. Він може звучати переконливо, але не є перевіреною трасою міркування, тож покладатися на нього не можна.

1.2.4 Академічні онтологічні системи

Принципово інший підхід пропонують академічні дослідження, які описують освітні програми за допомогою онтологій. Онтологія – це формальний опис предметної галузі, тобто перелік понять і зв'язків між

ними, записаний так, що комп'ютер може робити з нього логічні висновки. Завдяки цьому будь-який результат можна перевірити, бо кожне твердження впливає з явно заданих фактів. Слабке місце таких систем – вони майже завжди потребують, щоб користувач звертався до них формальною мовою, а не звичайними словами.

Однією з відомих робіт цього напрямку є дослідження Ю Ляна Чі [2]. У ньому послідовність вивчення курсів будується на основі OWL-онтології, у якій навчальні елементи та зв'язки між ними описано формально, а правила впорядкування задано спеціальною мовою логічних правил SWRL. Логічне виведення й перевірку несуперечливості виконує онтологічний резонатор у середовищі Protégé, тому отриманий порядок є машинно-перевірним, а не складеним вручну. Завдяки цьому підхід дає строгий, логічно обґрунтований результат, у якому кожна залежність між темами впливає з явно заданих фактів онтології. Водночас він має суттєве обмеження. Якість порядку повністю залежить від того, наскільки повно й коректно експерт заповнив онтологію, і будь-яка прогалина в ній одразу позначається на результаті. До того ж користувач не може звернутися до такої системи звичайною мовою, адже взаємодія відбувається в термінах формальної моделі.

Робота Назирової зі співавторами [9] ближча до нашої задачі, адже вона присвячена саме перевірці узгодженості передумов і результатів навчання. Автори побудували онтологію освітньої програми й перевіряють резонатором, чи немає в ній суперечностей у порядку дисциплін. Серед виявлених труднощів – складність графа для багаторічних програм і незручність поєднання такої онтології зі звичайними системами керування навчанням.

Робота Ванга зі співавторами описує систему OSCCA [10], цікаву тим, що онтологію курсів комп'ютерних наук там наповнюють напівавтоматично, обробляючи тексти каталогів курсів засобами обробки природної мови. Це показує, що онтологію можна не лише складати вручну,

а й розширювати з неструктурованих джерел. Близькі за духом дослідження застосування семантичних технологій до навчальних програм узагальнено й в інших оглядах галузі [11].

Спільна сильна сторона всіх цих систем – перевірність, будь-який висновок логічно обґрунтований у межах онтології. Спільні вади – відсутність зручного входу звичайною мовою та велика ручна праця на створення самої онтології.

1.2.5 Гібридні нейро-символьні системи

Окрему групу, що активно розвивається, становлять гібридні нейро-символьні системи. Вони поєднують дві парадигми: трансформерну модель обробки природної мови – для зручного входу звичайними словами – та онтологію чи граф знань – для перевірності висновку. Саме до цієї групи найближча система, що розробляється в цій роботі.

Робота Банерджі зі співавторами [3] показує, що трансформер може перекладати запит звичайною мовою у формальний запит SPARQL. На відомих наборах даних точність такого перекладу за метрикою F1 сягає 0,88–0,92, а покращений підхід TSET [12] з додатковим попереднім навчанням моделі – 0,94–0,95. Це підтверджує, що трансформер придатний як «перекладач» між звичайною мовою і мовою формальних запитів.

Найближча до нашої задачі – побудова впорядкованих послідовностей навчальних елементів. Робота Лю зі співавторами [13] поєднує кілька кроків: визначає прогалини в знаннях користувача, ранжує матеріал нейронною мережею, перевпорядковує його трансформером і насамкінець застосовує топологічне сортування. Оглядова стаття Лі зі співавторами [14] узагальнює галузь і прямо називає обхід у глибину, обхід у ширину та топологічне сортування класичними алгоритмами для цієї задачі, а серед головних труднощів – швидке зростання простору пошуку у великих графах і залежність якості результату від повноти графа.

Робота Муна [15] показує, як у такій системі реалізують перевірене пояснення. Трансформер перекладає запитання у послідовність SPARQL-запитів, які виконуються над графом, а користувач може сам переглянути цю послідовність і крок за кроком простежити, як система дійшла відповіді. Пояснення тут не згенероване, а є реальною трасою виконаних запитів.

Огляд способів пояснення для графових рекомендаційних систем [4] поділяє їх на кілька типів і відзначає, що пояснення у вигляді траси виведення найкраще відповідає вимозі прозорості, хоча його складніше реалізувати. Попри ці напрацювання, наявні гібридні системи здебільшого розв'язують суміжні задачі – відповіді на запитання чи переклад у SPARQL – і не спеціалізовані саме на побудові персонального освітнього плану з прив'язкою до структурно-логічної схеми конкретної освітньо-професійної програми.

До цього ж напрямку належать і вітчизняні дослідження. Зокрема, у роботі [5] концептуальні графи застосовано для подання структури масових відкритих онлайн-курсів (МООС) – підхід, близький до використаного в цій роботі, де структура освітньої програми так само подається у вигляді графа понять і зв'язків між ними.

1.2.6 Порівняльний аналіз розглянутих рішень

Підсумок розгляду зведено в таблицю 1.1. Кожну групу рішень оцінено за чотирма ознаками з початку підрозділу, а також за тим, чи спирається вона на структуровану освітню програму.

Аналіз таблиці показує головну закономірність: жодна з наявних груп не задовольняє одразу всіх потрібних ознак. Платформи дають доступ до контенту, ринкові рішення додають персоналізацію за даними ринку, але не вибудовують порядок. Великі мовні моделі розуміють звичайну мову, проте не можуть гарантувати правильність, бо не спираються на формальну базу знань. Академічні онтологічні системи дають строгий і перевірний

результат, але не приймають запит звичайною мовою. Гібридні системи поєднують сильні сторони, однак розв'язують суміжні задачі й не прив'язані до структурно-логічної схеми конкретної освітньої програми.

Таблиця 1.1 – Порівняння існуючих рішень за ключовими ознаками

Група рішень	Природна мова	Облік передумов	Адаптація	Перевірне пояснення	Прив'язка до ОПП
Масові онлайн-платформи	–	±	–	–	–
Ринково-орієнтовані платформи	–	–	±	–	–
LLM загального призначення	+	–	±	–	–
Академічні онтологічні системи	–	+	+	+	+
Гібридні нейро-символьні системи	+	+	±	±	–
Запропонована система	+	+	+	+	+

Звідси випливає висновок – поставлену задачу найкраще розв'язує гібридний підхід, спеціально побудований під побудову освітнього плану – той, що приймає запит звичайною мовою, спирається на онтологію освітньо-професійної програми, вибудовує порядок дисциплін графовими алгоритмами та пояснює результат перевірною трасою.

1.3 Мета роботи та постановка задачі

Проведений аналіз показав, що наявні рішення не повністю задовольняють потреби користувача, тому найкращим є спеціалізований гібридний підхід. Метою роботи є розробка та програмна реалізація інтелектуальної системи, яка на основі запиту користувача природною мовою формує логічно впорядкований персональний план вивчення

дисциплін із поясненням кожного кроку. Для цього визначено вхідні та вихідні дані системи.

Вхідними даними є три складові: формалізований опис освітньо-професійної програми (перелік дисциплін, зв'язки передумов, компетенції та базові знання), запит користувача звичайною мовою про те, що він хоче вивчити, та необов'язковий перелік уже відомого користувачу для виключення цих елементів із плану.

На основі цих даних система знаходить дисципліну, що найкраще відповідає запиту (мета плану), збирає всі її дисципліни-передумови та вибудовує їх у порядку логічного слідування. Кожна дисципліна позначається за важливістю (кількістю залежних курсів) і отримує пояснення щодо доцільності її включення.

Результат подається як впорядкований перелік дисциплін від базових до цільової із зазначенням їх важливості, передумов, наступних відкритих курсів та обґрунтування включення. Базові знання подаються окремим списком, що забезпечує користувача не лише послідовністю кроків, а й розумінням кожного з них.

Із цього випливають функціональні вимоги до системи:

- приймати запит звичайною мовою та знаходити відповідну йому дисциплін;
- будувати повну й несуперечливу послідовність передумов для обраної мети;
- позначати дисципліни за рівнем важливості в межах плану;
- надавати для кожної дисципліни пояснення на основі формальних зв'язків програми;
- виключати з плану дисципліни, які користувач уже опанував.

Нефункціональні вимоги передбачають, що кожен зв'язок у плані має бути перевіримим онтологічним резонатором, а результат – відтворюваним (однаковий запит за тих самих даних завжди дає однаковий

план). Також має бути забезпечена можливість розширювати онтологію без переписування алгоритмів.

Постановка задачі спирається на припущення, що система працює в межах однієї заздалегідь формалізованої програми, базові математичні знання вважаються набутими, а запит опрацьовується українською мовою для однієї цільової дисципліни (одночасне опрацювання кількох цілей не передбачено).

Межі задачі окреслюють, що базові методи (онтологічний інжиніринг, трансформерні моделі NLP, графові алгоритми та принципи ХАІ) є відомими й заново не розробляються. Власним внеском є їх поєднання в єдину систему для синтезу освітнього плану за схемою конкретної програми з оцінкою важливості дисциплін та перевірними поясненнями. Постійне відстеження прогресу користувача виходить за межі мети й може стати напрямом подальшого розвитку.

Узагальнити постановку задачі зручно на прикладі типового сценарію використання. Користувач, який прагне опанувати певний напрям, наприклад нейронні мережі, описує свою мету звичайними словами. Система зіставляє цей запит із дисциплінами програми, визначає цільову дисципліну й збирає всі її передумови – від базової математики та програмування до спеціальних дисциплін. Зібрані дисципліни вибудовуються в такому порядку, щоб кожна стояла після тих, на яких вона ґрунтується. Для кожної дисципліни система зазначає рівень її важливості та причину включення до плану, а те, що користувач уже опанував, з плану вилучає. Саме таку поведінку й повинна забезпечувати розроблювана система.

2 ОБҐРУНТУВАННЯ ТЕХНОЛОГІЙ, МЕТОДІВ ТА АРХІТЕКТУРИ ПРИ ПРОЄКТУВАННІ СИСТЕМИ

2.1 Загальна архітектура системи

Система побудована за гібридним принципом, обґрунтованим у розділі 1. Природномовний запит обробляє нейромережева складова, а правильність плану та перевірність пояснень забезпечує символна складова – онтологія та графові алгоритми над нею. Робота системи поділяється на дві фази: попередню підготовку даних, що виконується один раз, і обслуговування запиту користувача, що виконується щоразу.

У фазі підготовки система перетворює вихідні навчальні документи на формальну базу знань. Спершу з PDF-силабусів дисциплін витягуються структуровані дані – назва, передумови, модулі, теми та компетенції кожної дисципліни. На їх основі будується онтологія освітньо-професійної програми у форматі OWL, причому передумови, записані в силабусах вільним текстом, автоматично зіставляються з дисциплінами онтології. Готову онтологію перевіряє резонатор HermiT на несуперечливість і відсутність циклів. Окремо для кожної дисципліни обчислюється векторне представлення її змісту – числовий вектор, який згодом використовується для семантичного зіставлення із запитом. Результатом цієї фази є онтологія та індекс векторів, що надалі використовуються без повторного перерахунку.

У фазі обслуговування запит користувача проходить через три послідовні модулі, що працюють над спільною онтологією. Перший модуль обробляє природномовний запит і визначає дисципліну, яка найкраще йому відповідає, – вона стає метою плану. Другий модуль обходить граф передумов цієї мети, збирає всі потрібні дисципліни та вибудовує їх у коректному порядку з позначенням важливості. Третій модуль формує для

кожної дисципліни пояснення, складене з фактів онтології. Загальну схему роботи системи показано на рисунку 2.1.

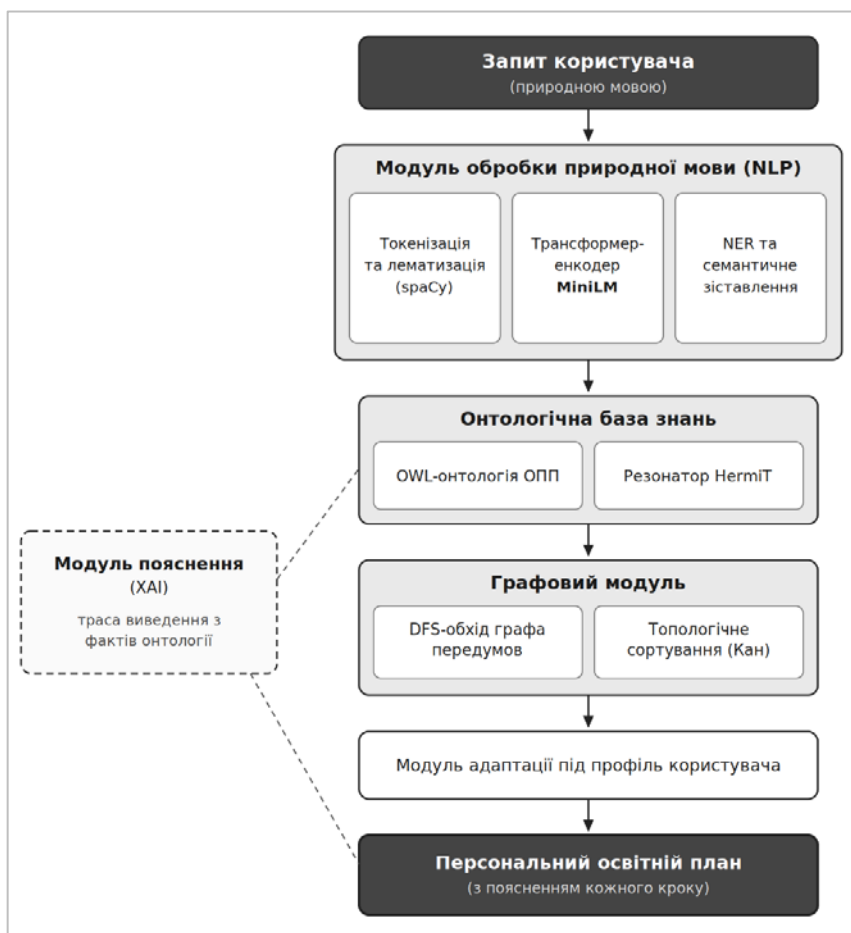


Рисунок 2.1 – Загальна архітектура системи

Доступ до системи реалізовано у двох варіантах: через інтерфейс командного рядка для перевірки роботи та через веб-інтерфейс для зручної взаємодії користувача. Програмно систему реалізовано мовою Python, онтологію опрацьовано бібліотекою owlready2, обробку природної мови виконано за допомогою бібліотек spaCy та sentence-transformers, а веб-частину – фреймворком Flask. Такий набір інструментів обрано через зрілу екосистему Python для обробки природної мови та інженерії онтологій, що дало змогу зосередитися на логіці самої системи, а не на низькорівневих деталях реалізації.

2.2 Модель онтології освітньо-професійної програми

Ядром системи є онтологія – формальний опис освітньо-професійної програми, на якому ґрунтуються побудова плану та пояснення. Онтологію реалізовано мовою OWL із кількох причин. По-перше, OWL має строгу логічну семантику, тому її вміст можна автоматично перевіряти на несуперечливість [16]. По-друге, для неї існують готові резонатори, зокрема HermiT [17], та зручні засоби розробки – редактор Protégé й бібліотека owlready2, використана в цій роботі. По-третє, онтологію легко розширювати новими дисциплінами без зміни алгоритмів, що працюють над нею.

Онтологія складається з двох частин: схеми, яка описує типи понять і відношення між ними, та наповнення, яке містить конкретні дисципліни програми. Спершу розглянемо схему.

Схема визначає шість класів понять. Клас Course відповідає навчальній дисципліні, Module – модулю в її складі, Topic – окремій темі модуля. Клас Competency описує компетенцію, яку формує дисципліна; до нього належать загальні (ЗК) та фахові (ФК) компетенції, а також програмні результати навчання (ПРН). Клас Specialization відповідає освітньо-професійній програмі, а клас BaseKnowledge – базовим знанням, що передують програмі й вивчаються до вступу. Усі шість класів є самостійними: вони не утворюють ієрархії підкласів, а пов'язуються між собою через відношення.

Зв'язки між поняттями задають сім об'єктних відношень. Центральним є hasPrerequisite – воно з'єднує дисципліну з її передумовою й утворює граф, яким працює модуль побудови плану. Зворотне до нього відношення isPrerequisiteOf автоматично вказує, для яких дисциплін певний курс є передумовою; воно використовується для побудови шляху до цілі в поясненнях. Відношення containsModule та containsTopic задають декомпозицію дисципліни на модулі й теми. Відношення

developsCompetency пов'язує дисципліну з компетенціями, leadsToSpecialization – з освітньою програмою, а requiresBaseKnowledge – з базовими знаннями поза програмою. Усі ці відношення показано на рисунку 2.2.

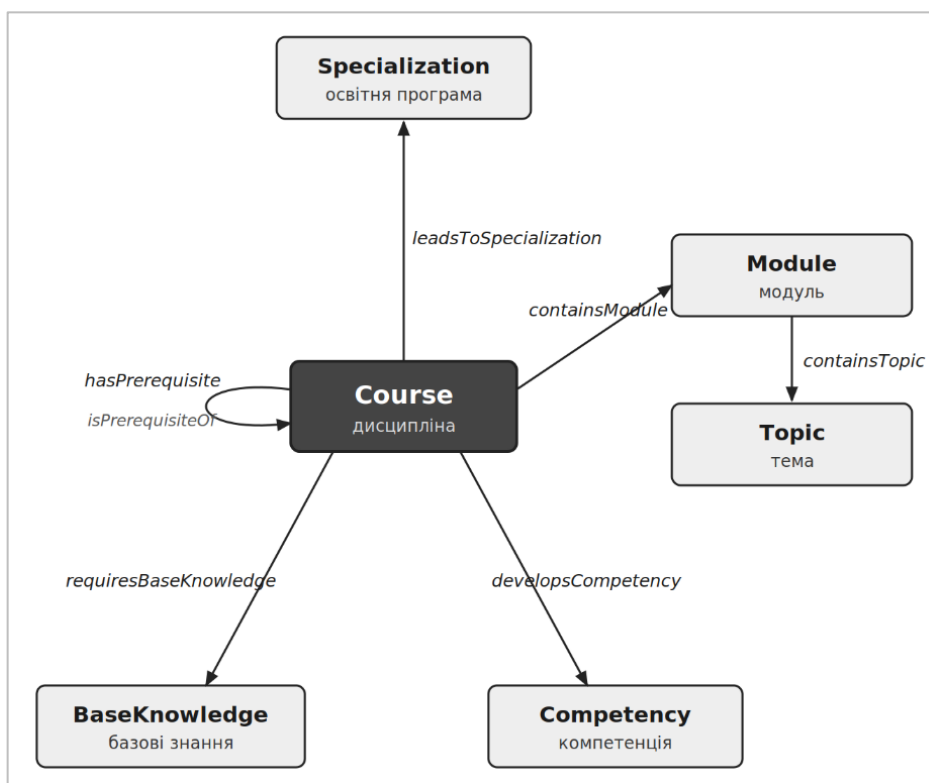


Рисунок 2.2 – Структура класів і відношень онтології.

Властивості самих понять задають шість атрибутів даних. Дисципліна має код (hasCode), кількість кредитів (hasCredits), семестр (hasSemester), рівень (hasLevel) та повний текст опису (hasFullText); компетенція має власний код (competencyCode). Слід зазначити, що кредити й семестр зберігаються лише як довідкові атрибути дисципліни – для впорядкування плану вони не використовуються, оскільки, як показано в підрозділі 1.1, для цільового користувача значущим є порядок передумов, а не календарний графік.

Повне оголошення класів і властивостей засобами owlready2 наведено в лістингу 2.1.

Наповнення онтології формується автоматично з даних, отриманих під час обробки силабусів. Побудована онтологія містить 53 дисципліни, 161 модуль і 566 тем, 41 компетенцію та три освітньо-професійні програми – «Штучний інтелект», «Науки про дані» та «Системи штучного інтелекту».

Лістинг 2.1 – Оголошення класів і властивостей онтології (owlready2)

with onto:

```
class Specialization(Thing): pass
class Course(Thing): pass
class Module(Thing): pass
class Topic(Thing): pass
class Competency(Thing): pass
class BaseKnowledge(Thing): pass

class hasPrerequisite(ObjectProperty):
    domain = [Course]; range = [Course]
class isPrerequisiteOf(ObjectProperty):
    domain = [Course]; range = [Course]
    inverse_property = hasPrerequisite
class containsModule(ObjectProperty):
    domain = [Course]; range = [Module]
class containsTopic(ObjectProperty):
    domain = [Module]; range = [Topic]
class developsCompetency(ObjectProperty):
    domain = [Course]; range = [Competency]
class leadsToSpecialization(ObjectProperty):
    domain = [Course]; range = [Specialization]
class requiresBaseKnowledge(ObjectProperty):
    domain = [Course]; range = [BaseKnowledge]

class hasCredits(DataProperty, FunctionalProperty):
    domain = [Course]; range = [float]
class hasSemester(DataProperty, FunctionalProperty):
    domain = [Course]; range = [str]
class hasCode(DataProperty, FunctionalProperty):
    domain = [Course]; range = [str]
class hasLevel(DataProperty, FunctionalProperty):
    domain = [Course]; range = [str]
class hasFullText(DataProperty, FunctionalProperty):
    domain = [Course]; range = [str]
class competencyCode(DataProperty, FunctionalProperty):
    domain = [Competency]; range = [str]
```

Окремої уваги потребує наповнення відношення передумови. У силабусах передумови вказані вільним текстом і часто не збігаються дослівно з офіційними назвами дисциплін – трапляються скорочення, друкарські помилки та різні варіанти написання. Тому кожен такий запис автоматично зіставляється з назвами дисциплін онтології методом нечіткого порівняння рядків. Спершу застосовується порівняння за відсортованими токенами з порогом подібності 80%, а якщо воно не дає результату, то використовується порівняння за входженням підрядка з вищим порогом 90%. Якщо для запису знайдено відповідну дисципліну, між ними створюється відношення `hasPrerequisite`; якщо відповідника немає, запис вважається базовим знанням поза програмою й заноситься до класу `BaseKnowledge`. У такий спосіб в онтології утворилося 139 зв'язків передумови між дисциплінами та 15 базових знань.

Готову онтологію перевірено резонатором `HermiT`. Перевірка підтвердила, що онтологія несуперечлива й не містить циклів у відношенні передумови. Відсутність циклів означає, що граф передумов є орієнтованим ациклічним графом, – а це необхідна умова коректної роботи топологічного сортування, на якому ґрунтується побудова плану.

2.3 Проєктування конвеєра обробки природномовного запиту

Завдання модуля обробки природної мови – за вільним запитом користувача визначити одну дисципліну онтології, яка найкраще йому відповідає. Складність полягає в тому, що користувач формулює запит власною лексикою, яка не збігається з академічними назвами дисциплін (підрозділ 1.1). Тому зіставлення виконується не за збігом слів, а за змістом. І запит, і дисципліни переводяться у числові вектори, близькість яких відображає смислову близькість тексту.

Запит проходить через послідовність кроків обробки, показану на рисунку 2.3. Перші кроки готують текст, центральні переводять його у

вектор і знаходять найближчу дисципліну, останній формує формальний запит до онтології.

На підготовчому етапі запит послідовно нормалізується, токенизується, лематизується та аналізується на наявність доменних термінів. Нормалізація замінює поширені скорочення ІТ-галузі на повні форми, наприклад: «ML» на «машинне навчання», «NLP» на «обробка природної мови», – щоб аббревіатури не заважали зіставленню.

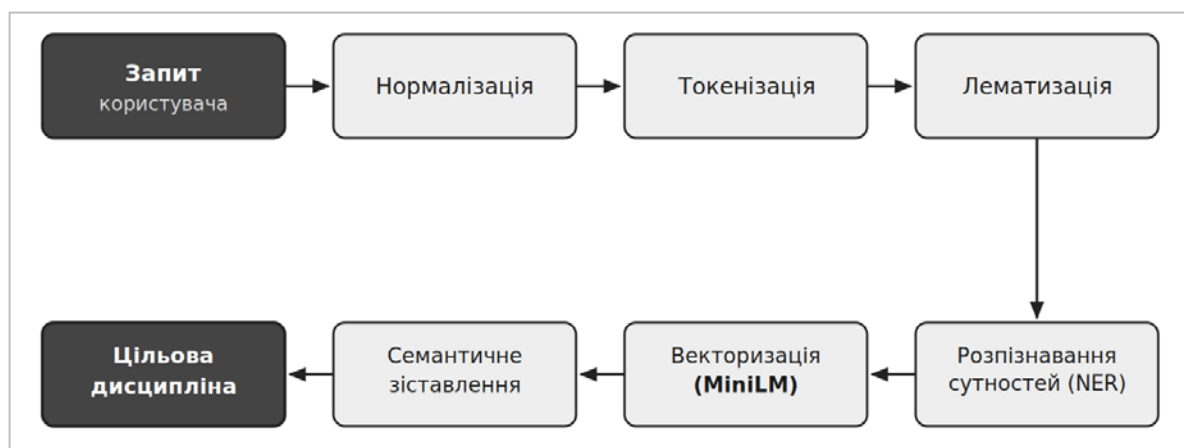


Рисунок 2.3 – Конвеєр обробки природномовного запиту.

Токенізація розбиває текст на окремі слова, а лематизація зводить їх до початкової форми; обидві операції виконує модель української мови `uk_core_news_sm` бібліотеки `spaCy`. Завдяки лематизації запити «нейронних мережах» і «нейронні мережі» сприймаються однаково. Розпізнавання іменованих сутностей виявляє в запиті відомі доменні терміни – спочатку засобами `spaCy`, а потім пошуком за словником предметних термінів [18].

На центральному етапі очищений запит переводиться у числовий вектор. Для цього використано трансформерну модель-енкодер `paraphrase-multilingual-MiniLM-L12-v2` з бібліотеки `sentence-transformers` [19]. Її обрано з трьох міркувань: вона багатомовна й підтримує українську, компактна та швидка, а на виході дає нормований вектор фіксованої довжини у 384 виміри, зручний для порівняння за подібністю.

Кожну дисципліну онтології переведено в такий самий вектор заздалегідь, на етапі підготовки даних, тому під час запиту повторне обчислення не потрібне.

На етапі семантичного зіставлення вектор запиту порівнюється з вектором кожної дисципліни за косинусною подібністю, яку визначають за формулою (2.1). Оскільки всі вектори нормовано до одиничної довжини, їхні норми дорівнюють одиниці, і косинусна подібність зводиться до скалярного добутку векторів. Це істотно спрощує обчислення, адже подібність до всіх дисциплін знаходиться одним множенням матриці їхніх векторів на вектор запиту [20]. Дисципліна з найбільшою подібністю стає метою плану.

$$\cos (q, c) = \frac{\sum_{i=1}^n q_i c_i}{\sqrt{\sum_{i=1}^n q_i^2} \cdot \sqrt{\sum_{i=1}^n c_i^2}} = \sum_{i=1}^n q_i c_i, \quad (2.1)$$

де q – вектор запиту;

c – вектор дисципліни;

n – розмірність векторів ($n = 384$).

Знайдений збіг приймається лише тоді, коли його подібність перевищує поріг. Поріг є адаптивним: за замовчуванням він становить 0,40, але знижується до 0,25, якщо на кроці розпізнавання сутностей у запиті виявлено доменний термін. Розпізнаний термін сам по собі є додатковим свідченням того, що збіг змістовний, тому для таких запитів можна прийняти трохи нижчу подібність. Це зменшує кількість хибних відмов для коротких, але однозначних запитів.

На останньому кроці для знайденої дисципліни формується шаблонний запит мовою SPARQL – це пов'язує природномовний вхід із формальним рівнем онтології та дає змогу за потреби отримати з неї додаткові факти. Результатом роботи модуля є ідентифікована цільова дисципліна разом зі значенням подібності та її ідентифікатором.

2.4 Проєктування модуля побудови плану

Маючи цільову дисципліну, визначену модулем обробки запиту, модуль побудови плану формує впорядковану послідовність дисциплін, потрібних для її досягнення. Робота модуля складається з трьох дій: зібрати всі передумови цілі, упорядкувати їх і позначити важливість кожної; додатково результат адаптується під профіль користувача.

Збирання передумов виконується обходом графа. Починаючи з цільової дисципліни, модуль рухається відношенням передумови вглиб – від дисципліни до її передумов, далі до передумов цих передумов, і так збирає всі дисципліни, від яких ціль залежить прямо чи опосередковано. Обхід виконується у глибину, але реалізований не рекурсивно, а через явний стек. Такий вибір зроблено навмисно, бо рекурсія на довгих ланцюгах передумов могла б призвести до переповнення стека викликів, тоді як ітеративний обхід працює зі скільки завгодно глибоким графом. Під час обходу передумови, що лежать поза програмою, збираються окремо як базові знання, а для кожної дисципліни запам'ятовується її глибина відносно цілі; якщо дисципліна досяжна кількома шляхами, береться найбільша глибина.

Зібрані дисципліни утворюють підграф, який потрібно вибудувати в лінійну послідовність так, щоб кожна дисципліна стояла після всіх своїх передумов. Це задача топологічного сортування; її розв'язано алгоритмом Кана [21]. За цим алгоритмом серед дисциплін обираються ті, що не мають невиконаних передумов, вони додаються до результату й вилучаються з графа, після чого крок повторюється, доки граф не спорожніє. Обчислювальна складність такого сортування лінійна й визначається за формулою (2.2). Якщо алгоритм не може впорядкувати всі дисципліни, це означає наявність циклу, і результат позначається як некоректний. Оскільки відсутність циклів підтверджено резонатором *HermiT* (підрозділ 2.2), для побудованої онтології такої ситуації не виникає.

$$T(n) = O(V + E), \quad (2.2)$$

де V – кількість дисциплін у плані (вершини графа);

E – кількість зв'язків передумови між ними (дуги графа).

Після формування послідовності кожній дисципліні присвоюється рівень важливості. Він визначається не семестром чи кількістю кредитів, а структурною роллю дисципліни в плані – тим, скільки інших дисциплін плану безпосередньо на неї спираються. Рівень важливості обчислюється за формулою (2.3), де $d(c)$ – кількість дисциплін плану, для яких дисципліна c є безпосередньою передумовою. Цільова дисципліна позначається окремо. Дисципліна, на якій ґрунтується багато інших, є вузловою, і прогалина в ній блокує цілу гілку плану, тому її варто опанувати насамперед. Саме ця метрика замінює впорядкування за семестрами, недоцільність якого для цільового користувача обґрунтовано в підрозділі 1.1.

$$\text{importance}(c) = \begin{cases} \text{критичний,} & d(c) \geq 3 \\ \text{важливий,} & 1 \leq d(c) \leq 2, \\ \text{опційний,} & d(c) = 0 \end{cases} \quad (2.3)$$

де $d(c)$ – кількість дисциплін плану, для яких дисципліна c є безпосередньою передумовою.

Якщо користувач вказав дисципліни, які вже опанував, вони вилучаються з плану ще до обчислення важливості, тож план містить лише те, що користувачеві ще потрібно. Відповідність між указаними відомими дисциплінами та дисциплінами плану встановлюється за частковим збігом назв, що дає змогу враховувати відмінності у формулюванні.

Залежно від запиту та профілю модуль формує один із трьох варіантів результату. У звичайному стані ціль користувачеві ще невідома, і будується повний план її досягнення. Якщо ж користувач уже знає цільову дисципліну, замість плану модуль рекомендує наступні дисципліни, що

спираються на неї через зворотне відношення `isPrerequisiteOf`. Нарешті, якщо користувач знає всі передумови цілі, у плані лишається тільки сама ціль, і до неї можна братися одразу. Ці стани роблять систему корисною не лише для початківців, а й для користувачів із попередньою підготовкою.

Результатом роботи модуля є впорядкований план: цільова дисципліна, послідовність дисциплін із позначками важливості, окремий перелік базових знань, стан плану, а для стану з відомою ціллю – перелік рекомендованих наступних дисциплін.

2.5 Проєктування модуля пояснення

Завдання модуля пояснень полягає в тому, щоб обґрунтувати включення кожної дисципліни до плану та сформувати довіру користувача до отриманих результатів. Ключове проєктне рішення полягає в тому, що ці роз'яснення не генеруються мовною моделлю. Натомість їх складено з фактів онтології за наперед заданими шаблонами. Це зроблено навмисно. Шаблон, заповнений фактом онтології, гарантує, що кожне твердження пояснення спирається на конкретний зв'язок у графі знань, а отже, його можна перевірити. Саме такої гарантії не може дати генеративна модель, відповідь якої, як показано в підрозділі 1.2, може виявитися вигадкою.

Пояснення кожної дисципліни складається з кількох частин. Воно містить рівень важливості дисципліни; стислу причину, чому вона в плані; перелік передумов із плану, на які вона спирається; перелік дисциплін плану, які на неї спираються; компетенції, які вона розвиває; і шлях від неї до цілі. Усі ці частини читаються безпосередньо з відношень онтології: передумови – з відношення `hasPrerequisite`, залежні дисципліни – зі зворотного `isPrerequisiteOf`, компетенції – з `developsCompetency`.

Причину сформульовано фіксованою фразою, яку обирають за роллю дисципліни в плані – ціль, критична, важлива чи опційна. Обраний шаблон заповнюється конкретними числами з графа, наприклад, для критичної

дисципліни фраза повідомляє, скільки наступних курсів на ній ґрунтуються. Оскільки ці числа різні, фраза узгоджує граматичну форму слова (один курс, два–чотири курси, п'ять і більше курсів), щоб звучати природно українською.

Шлях від дисципліни до цілі будується пошуком у ширину за зворотним відношенням `isPrerequisiteOf` у межах плану. Результатом є конкретний ланцюг дисциплін, що веде від заданої дисципліни до мети, наприклад «Чисельні методи → Теорія ймовірності → Машинне навчання → Штучні нейронні мережі». Користувач бачить не загальне твердження, а точний логічний маршрут.

Кожна частина пояснення – передумови, залежні дисципліни, компетенції, шлях – є прямим прочитанням відношення, яке існує в онтології й може бути перевірене в редакторі *Protégé* або підтверджене резонатором *HermiT*. Тому пояснення є трасою перевірених фактів, а не згенерованою розповіддю. У цьому й полягає принципова відмінність від генеративних пояснень: обґрунтування мовної моделі може звучати переконливо, але виявитися хибним, тоді як тут хибне твердження неможливе за побудовою, бо кожную фразу складено лише з наявного факту онтології.

Результатом роботи модуля є повне пояснення описаної структури для кожної дисципліни плану; разом вони утворюють пояснений план, який повертається користувачеві.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Засоби та технології розробки

Систему реалізовано мовою Python версії 3.12. Цю мову обрано тому, що вона має зрілі бібліотеки для всіх потрібних у роботі напрямів – роботи з онтологіями, обробки природної мови та створення веб-застосунків, – що дало змогу реалізувати всі компоненти в межах однієї технології. Розробку виконано в середовищі PyCharm з ізольованим віртуальним оточенням, яке відокремлює залежності проєкту від системних.

Для роботи з онтологією використано бібліотеку owlready2 [22]. Вона дає змогу створювати класи, екземпляри та відношення онтології безпосередньо з коду Python, зберігати її у файл формату OWL і запускати онтологічний резонатор. Перевірку несуперечливості й відсутності циклів виконує резонатор HermiT, який викликається через цю саму бібліотеку. Текст із PDF-силабусів видобуто бібліотекою pdfplumber, а нечітке зіставлення передумов із назвами дисциплін реалізовано засобами бібліотеки rapidfuzz.

Обробку природної мови побудовано на двох бібліотеках. Токенізацію, лематизацію та розпізнавання іменованих сутностей виконує бібліотека spaCy з україномовною моделлю uk_core_news_sm. Векторне подання тексту створює бібліотека sentence-transformers, у якій використано трансформерну модель-енкодер MiniLM.

Веб-частину реалізовано легким фреймворком Flask, що відповідає за приймання запитів і обмін даними з клієнтом у форматі JSON. Інтерактивний граф передумов у браузері будує бібліотека vis.js, а решту інтерфейсу виконано засобами HTML, CSS і JavaScript без додаткових фреймворків.

Вхідними даними для системи є силабуси навчальних дисциплін освітніх програм кафедри штучного інтелекту ХНУРЕ у форматі PDF та структурно-логічна схема цих програм – передумови між дисциплінами й перелік компетенцій. Саме з цих джерел будується онтологічна база знань.

Реалізацію системи поділено на дві фази. Перша – підготовка бази знань. Вона виконується один раз і перетворює силабуси на готову онтологію та індекс векторів. Друга – обслуговування запиту, що виконується щоразу й послідовно задіює три модулі: обробки запиту, побудови плану та пояснення.

3.2 Формування онтологічної бази знань

Онтологічна база знань – це фундамент системи, від якого залежать і побудова плану, і пояснення. Її формування становить першу фазу роботи системи й виконується один раз. Ця фаза перетворює неструктуровані вихідні документи – силабуси дисциплін у форматі PDF – на формальну онтологію, придатну для автоматичного опрацювання. Процес складається з трьох послідовних кроків: спершу з силабусів видобуваються структуровані дані, потім на їх основі будується онтологія з усіма зв'язками між дисциплінами, і нарешті готова онтологія перевіряється на коректність, а для кожної дисципліни обчислюється векторне подання її змісту.

3.2.1 Парсинг силабусів

Вихідними документами є силабуси дисциплін – PDF-файли з напівструктурованим викладом, де відомості подано пронумерованими розділами: назва, рівень, кредити, передумови, анотація з модулями й темами, перелік компетенцій. Завдання парсера – перетворити кожен такий документ на структурований запис, придатний для побудови онтології.

Обробка починається з видобування тексту. Кожен PDF відкривається, і його вміст зчитується посторінково зі збереженням розташування тексту, що допомагає коректно відділити пронумеровані поля одне від одного. Реалізацію цього кроку наведено в лістингу 3.1.

Лістинг 3.1 – Видобування тексту із PDF-силабуса

```
def extract_text(pdf_path: Path) -> str:
    text_parts = []
    with pdfplumber.open(pdf_path) as pdf:
        for page in pdf.pages:
            t = page.extract_text(layout=True)
            if t:
                text_parts.append(t)
    return "\n".join(text_parts)
```

З видобутого тексту парсер витягує потрібні поля за шаблонами: назву дисципліни, спеціальність, рівень, кількість кредитів, семестр, передумови у вигляді вільного тексту, а також модулі з їхніми темами та перелік компетенцій.

Оскільки силабуси оформлені не цілком однаково, для частини полів передбачено запасні шаблони пошуку, а кілька документів із нестандартним форматуванням обробляються за наперед заданими винятками. Це забезпечує коректний розбір усього набору без ручного редагування самих PDF-файлів.

Результатом кроку є набір структурованих записів для всіх 55 силабусів, який зберігається у файлі формату JSON. Повний код модуля парсингу силабусів представлено у додатку А.

3.2.2 Побудова онтології та нечітке зіставлення передумов

З отриманих на попередньому кроці структурованих записів будується онтологія. Для кожної дисципліни створюється екземпляр класу

Course, якому встановлюються атрибути – код, кредити, семестр, рівень і повний текст анотації, – а також додаються зв'язки з освітньою програмою (leadsToSpecialization) і компетенціями (developsCompetency). Так само створюються екземпляри модулів і тем та поєднуються з дисципліною відношеннями containsModule й containsTopic. Два документи, що не є навчальними дисциплінами, – «Кваліфікаційна робота» та «Практика» – до онтології не вносяться, тож із 55 силабусів лишається 53 дисципліни.

Створення цих екземплярів і зв'язків є прямим, адже кожне поле запису перетворюється на відповідну властивість онтології. Нетривіальним є лише наповнення відношення передумови, оскільки передумови в силабусах записані вільним текстом і не збігаються дослівно з назвами дисциплін. Тому кожен такий запис зіставляється з назвами дисциплін методом нечіткого порівняння рядків, реалізованим у функції з лістингу 3.2.

Лістинг 3.2 – Нечітке зіставлення передумови з назвами дисциплін

```
def fuzzy_match_prerequisite(prereq_norm, normalized_names_list,
threshold):
```

```
    r1 = process.extractOne(prereq_norm, normalized_names_list,
                             scorer=fuzz.token_sort_ratio)
```

```
    if r1 and r1[1] >= threshold:
```

```
        return r1[0], r1[1], "token_sort"
```

```
    r2 = process.extractOne(prereq_norm, normalized_names_list,
                             scorer=fuzz.partial_ratio)
```

```
    if r2 and r2[1] >= max(threshold + 10, 90):
```

```
        return r2[0], r2[1], "partial"
```

```
    best = max([r for r in [r1, r2] if r], key=lambda x: x[1],
default=None)
```

```
    if best:
```

```
        return None, best[1], None
```

```
    return None, 0, None
```

Функція працює у два проходи. Спершу застосовується порівняння за відсортованими токенами – воно стійке до різного порядку слів, – і збіг приймається, якщо подібність не нижча за поріг 80%. Якщо цей прохід не дав результату, виконується порівняння за входженням підрядка з підвищеним порогом 90%, що відсіює випадкові часткові збіги. Якщо жоден прохід не перевищив порогу, відповідник вважається не знайденим.

Результат зіставлення застосовується так, як показано в лістингу 3.3. Якщо для запису знайдено дисципліну, між поточною дисципліною й знайденою створюється відношення `hasPrerequisite`. Якщо ж відповідника немає, запис вважається базовим знанням поза програмою й заноситься до класу `BaseKnowledge`. Випадки, коли дисципліна зіставляється сама із собою, відкидаються (лістинг 3.3).

Лістинг 3.3 – Створення зв'язків передумови за результатом зіставлення

```
for raw_prereq_text in c["prerequisites"]:
    prereq_text = clean_prereq_text(raw_prereq_text)
    prereq_norm = normalize_name(prereq_text)
    matched_norm, similarity, method = fuzzy_match_prerequisite(
        prereq_norm, normalized_names_list, fuzzy_threshold)
    if matched_norm:
        matched_file = name_to_file[matched_norm]
        if matched_file == c["file"]:
            continue
        matched_course = course_individuals[matched_file]
        course_ind.hasPrerequisite.append(matched_course)
    else:
        if prereq_text not in base_knowledge_individuals:
            bk = BaseKnowledge("bk_" + safe_iri_name(prereq_text),
                               namespace=onto)
            bk.label = [locstr(prereq_text, lang="uk")]
            base_knowledge_individuals[prereq_text] = bk
        course_ind.requiresBaseKnowledge.append(
            base_knowledge_individuals[prereq_text])
```

У такий спосіб в онтології утворилося 53 дисципліни та 139 зв'язків передумови між ними; передумови, що не належать до програми, винесено в окремий клас базових знань. Готова онтологія зберігається у файл формату OWL. Повний код формування онтології представлено у додатку Б.

3.2.3 Перевірка узгодженості та індекс ембедингів

Збережену онтологію перевіряє резонатор HermiT. Перевірку запускають через бібліотеку owlready2 одним викликом, який ініціює міркування над онтологією й виявляє в ній можливі суперечності.

Резонатор підтвердив, що онтологія несуперечлива, а граф передумов не містить циклів. Відсутність циклів гарантує, що цей граф є орієнтованим ациклічним, – необхідна умова коректної роботи топологічного сортування (підрозділ 2.4). Резонатор HermiT потребує середовища виконання Java; якщо воно недоступне, ту саму перевірку можна виконати в редакторі Protégé.

Другий крок цієї фази – побудова індексу векторних представлень, потрібного для семантичного зіставлення (підрозділ 2.3). Текст, що представляє дисципліну, – її назва, спеціальність і анотація – переводиться трансформерною моделлю MiniLM у вектор розмірністю 384. Вектори нормуються до одиничної довжини, завдяки чому косинусна подібність зводиться до скалярного добутку, як описано в підрозділі 2.3. Реалізацію цього кроку наведено в лістингу 3.4.

Окрім 53 дисциплін, до індексу внесено також вектори спеціальностей і тем. Увесь індекс зберігається у файл і завантажується під час обслуговування запиту, тож кодування виконується один раз і надалі не повторюється. Повний код побудови індексу ембедингів представлено у додатку В.

Лістинг 3.4 – Обчислення векторних представлень дисциплін

```
model = SentenceTransformer(
    "sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2")
embeddings = model.encode(
    texts,
    convert_to_numpy=True,
    normalize_embeddings=True,
)
np.savez_compressed(output_path, embeddings=embeddings,
                    iris=iris, labels=labels)
```

3.3 Реалізація модуля обробки запиту

Модуль обробки запиту реалізує конвеєр, спроектований у підрозділі 2.3: він приймає запит користувача й повертає дисципліну, що найкраще йому відповідає. Кроки конвеєра виконуються послідовно.

Спершу запит проходить підготовку. Нормалізація замінює ІТ-аббревіатури на повні форми за словником відповідників. Далі модель spaCy виконує токенізацію, лематизацію та розпізнавання іменованих сутностей. Після лематизації зі списку слів вилючаються службові (стоп-слова), а решта значущих лем об'єднується в очищений запит – саме його, а не вихідний текст, передають на кодування.

Очищений запит кодується у нормований вектор розмірністю 384 тією самою моделлю MiniLM, що й дисципліни в індексі. Потім цей вектор порівнюється з векторами всіх дисциплін, як показано в лістингу 3.5.

Порівняння виконується одним множенням матриці векторів на вектор запиту. Оскільки всі вектори нормовано, цей добуток одразу дає косинусну подібність запиту до кожної сутності (підрозділ 2.3). Результати фільтруються за типом – залишаються лише дисципліни – і впорядковуються за спаданням подібності; перша з них є кандидатом на ціль.

Лістинг 3.5 – Семантичне зіставлення запиту з дисциплінами

```
def semantic_match(self, query_vector, top_n=5, filter_type=None):
    scores = self.embeddings @ query_vector
    if filter_type:
        indices = np.where(self.entity_types == filter_type)[0]
        top_local = np.argsort(-scores[indices])[:top_n]
        top_indices = indices[top_local]
    else:
        top_indices = np.argsort(-scores)[:top_n]
    return [MatchResult(iri=str(self.iris[i]),
                        label=str(self.labels[i]),
                        similarity=float(scores[i]))
            for i in top_indices]
```

Кандидат приймається лише тоді, коли його подібність перевищує поріг. Поріг адаптивний, як показано в лістингу 3.6. Якщо на кроці розпізнавання сутностей у назві кандидата виявлено доменний термін, застосовується знижений поріг 0,25, інакше – стандартний 0,40.

Лістинг 3.6 – Адаптивний поріг прийняття збігу

```
best = matches[0] if matches else None
confidence_ok = False
if best:
    ner_hit = self.check_ner_match(entities, best.label)
    if ner_hit:
        threshold = CONFIDENCE_THRESHOLD_NER
    else:
        threshold = CONFIDENCE_THRESHOLD
    confidence_ok = best.similarity >= threshold
```

Результатом роботи модуля є знайдена цільова дисципліна разом зі значенням подібності та згенерованим формальним запитом. Повний код модуля обробки запиту представлено у додатку Г.

3.4 Реалізація модуля побудови плану

Модуль побудови плану реалізує дії, спроектовані в підрозділі 2.4: збирання передумов, їх упорядкування, позначення важливості та адаптацію під профіль користувача.

Збирання передумов реалізовано обходом графа у глибину, наведеним у лістингу 3.7.

Лістинг 3.7 – Збирання передумов обходом у глибину

```
def collect_prerequisites_dfs(self, target_course):
    all_courses, base_knowledge, depth_map = set(), set(), {}
    stack = [(target_course, 0)]
    while stack:
        current, depth = stack.pop()
        if self.onto.BaseKnowledge in current.is_a:
            base_knowledge.add(current)
            continue
        if self.onto.Course not in current.is_a:
            continue
        if current in all_courses:
            depth_map[current] = max(depth_map[current], depth)
            continue
        all_courses.add(current)
        depth_map[current] = depth
        for prereq in current.hasPrerequisite:
            stack.append((prereq, depth + 1))
        for bk in current.requiresBaseKnowledge:
            base_knowledge.add(bk)
    return all_courses, depth_map, base_knowledge
```

Обхід виконується не рекурсивно, а через явний стек, кожен елемент якого – пара «дисципліна, глибина». Для поточної дисципліни в стек додаються всі її передумови з більшою на одиницю глибиною; базові знання збираються окремо, а для дисциплін, досяжних кількома шляхами,

зберігається найбільша глибина. Стек замість рекурсії обрано навмисно, щоб уникнути переповнення стека викликів на глибоких графах.

Зібрані дисципліни впорядковуються алгоритмом Кана, наведеним у лістингу 3.8. Спершу для кожної дисципліни підраховується кількість її передумов у межах підграфа – вхідний степінь. Дисципліни без передумов потрапляють у чергу; з черги по черзі вибирається дисципліна, додається до результату, а у її залежних дисциплін вхідний степінь зменшується, і коли він сягає нуля, дисципліна теж додається до черги.

Лістинг 3.8 – Топологічне впорядкування за алгоритмом Кана

```
def topological_sort_kahn(self, courses):
    course_set = set(courses)
    in_degree = {c: 0 for c in courses}
    adj = defaultdict(list)
    for c in courses:
        for prereq in c.hasPrerequisite:
            if prereq in course_set:
                adj[prereq].append(c)
                in_degree[c] += 1
    queue = deque([c for c in courses if in_degree[c] == 0])
    ordered = []
    while queue:
        current = queue.popleft()
        ordered.append(current)
        for next_course in adj[current]:
            in_degree[next_course] -= 1
            if in_degree[next_course] == 0:
                queue.append(next_course)
    is_valid = len(ordered) == len(courses)
    return ordered, is_valid
```

Якщо в результат потрапило менше дисциплін, ніж було на вході, у графі є цикл; на побудованій онтології цього не трапляється, бо відсутність циклів підтверджено HermiT. Складність алгоритму лінійна (формула 2.2).

Після впорядкування кожній дисципліні присвоюється рівень важливості, як показано в лістингу 3.9. Для цього спершу для кожної дисципліни підраховується, скільки інших дисциплін плану безпосередньо на неї спираються, а потім за цим числом визначається рівень: три й більше залежних – критична, одна-дві – важлива, жодної – опційна. Це реалізує метрику з формули 2.3. Підрахунок ведеться по остаточному плану, тобто вже після вилучення відомих користувачеві дисциплін.

Адаптацію під профіль і три стани плану – звичайний, з відомою ціллю та з відомими передумовами – реалізовано так, як описано в підрозділі 2.4.

Лістинг 3.9 – Обчислення критичності дисциплін плану

```
dependents_count = {c: 0 for c in final_sequence}
for c in final_sequence:
    for prereq in c.hasPrerequisite:
        if prereq in final_set:
            dependents_count[prereq] += 1

for c in final_sequence:
    dep_count = dependents_count[c]
    if dep_count >= 3:
        importance = "critical"
    elif dep_count >= 1:
        importance = "important"
    else:
        importance = "optional"
```

Результатом роботи модуля є впорядкований план із позначками важливості, окремим переліком базових знань і станом плану. Повний код модуля представлено у додатку Д.

3.5 Реалізація модуля пояснення

Модуль пояснення реалізує підхід, обґрунтований у підрозділі 2.5: для кожної дисципліни плану він складає пояснення цілком із фактів онтології, без жодної генерації тексту. Пояснення подається структурою з полями – рівень важливості, причина, передумови, залежні дисципліни, компетенції та шлях до цілі.

Більшість полів – це прямі прочитання відношень онтології, як показано в лістингу 3.10. Залежні дисципліни беруться з відношення `isPrerequisiteOf`, передумови – з `hasPrerequisite`, компетенції – з `developsCompetency`; у кожному випадку враховуються лише ті сутності, що входять до плану. Саме тому пояснення перевіряє, бо кожен його пункт відповідає конкретному зв'язку, який можна знайти в онтології.

Лістинг 3.10 – Збирання фактів пояснення з відношень онтології

```
enables = [str(d.label[0]) for d in course.isPrerequisiteOf
            if d.name in plan_set and d.label]
requires = [str(p.label[0]) for p in course.hasPrerequisite
            if p.name in plan_set and p.label]
competencies = []
for comp in course.developsCompetency:
    code = str(comp.label[0]) if comp.label else comp.name
    competencies.append((code, self._get_competency_label(code)))
```

Текстову причину сформульовано шаблоном, який обирають за роллю дисципліни й заповнюють числом із графа (наприклад, для критичної дисципліни – «критичний базовий курс – на ньому ґрунтуються N наступних курсів вашого плану»). Допоміжна функція відмінювання добирає правильну форму слова для будь-якого N. Жодну з цих фраз не згенеровано мовною моделлю – це фіксовані шаблони, заповнені фактами.

Шлях від дисципліни до цілі будується пошуком у ширину за відношенням `isPrerequisiteOf` у межах плану, як показано в лістингу 3.11.

Обхід рухається від заданої дисципліни до залежних від неї, доки не досягне цілі, а знайдений ланцюг ідентифікаторів перетворюється на послідовність назв дисциплін.

Лістинг 3.11 – Побудова шляху до цілі пошуком у ширину

```
def _build_path_to_target(self, course_iri, plan_iris):
    target_iri = plan_iris[-1]
    if course_iri == target_iri:
        return []
    visited = {course_iri}
    queue = deque([(course_iri, [course_iri])])
    plan_set = set(plan_iris)
    while queue:
        current_iri, path = queue.popleft()
        current = self._find_individual_by_iri(current_iri)
        for dependent in current.isPrerequisiteOf:
            if dependent.name not in plan_set or dependent.name in
visited:
                continue
            new_path = path + [dependent.name]
            if dependent.name == target_iri:
                return
[
    str(self._find_individual_by_iri(i).label[0])
        for i in new_path]
            visited.add(dependent.name)
            queue.append((dependent.name, new_path))
    return []
```

Оскільки кожне поле пояснення – причина, передумови, залежні дисципліни, компетенції та шлях – отримане безпосередньо з онтології, усе пояснення є перевіримим, адже будь-яке його твердження можна підтвердити відповідним фактом у графі знань. Результатом роботи модуля є набір таких пояснень для всіх дисциплін плану. Повний код модуля пояснення представлено у додатку Е.

4 ІНТЕРФЕЙС КОРИСТУВАЧА ТА ТЕСТУВАННЯ СИСТЕМИ

4.1 Інтерфейс користувача

Систему використовують через веб-інтерфейс у браузері. Взаємодія має два основні екрани: початковий, де користувач вводить запит, і екран результату, де відображається побудований план із поясненнями.

Початковий екран наведено на рисунку 4.1. У центрі сторінки розміщено поле для введення запиту, куди користувач звичайними словами вписує, що хоче вивчити, і кнопку запуску пошуку. Під полем подано кілька прикладів запитів, які можна обрати одним натисканням. Нижче розташовано окремий блок, у якому користувач може вказати дисципліни, що вже опанував, – вони будуть виключені з плану. Цей блок є точкою входу до адаптації плану під профіль користувача.

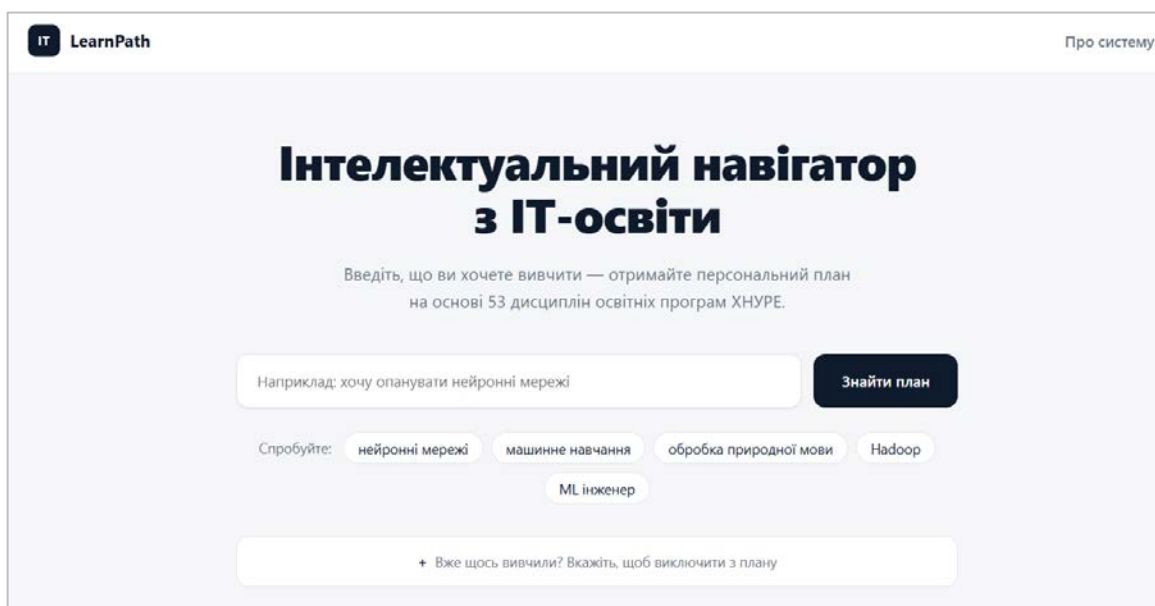


Рисунок 4.1 – Головна сторінка системи

Після введення запиту й запуску пошуку відкривається екран результату, зведену частину якого наведено на рисунку 4.2. Угорі показано

знайдену цільову дисципліну та значення її семантичної подібності до запиту. Нижче розміщено лічильники, що підсумовують план: загальну кількість дисциплін і їх розподіл за рівнями важливості – критичні, важливі та опційні. Поряд подано легенду, яка пояснює, що означає кожен рівень.

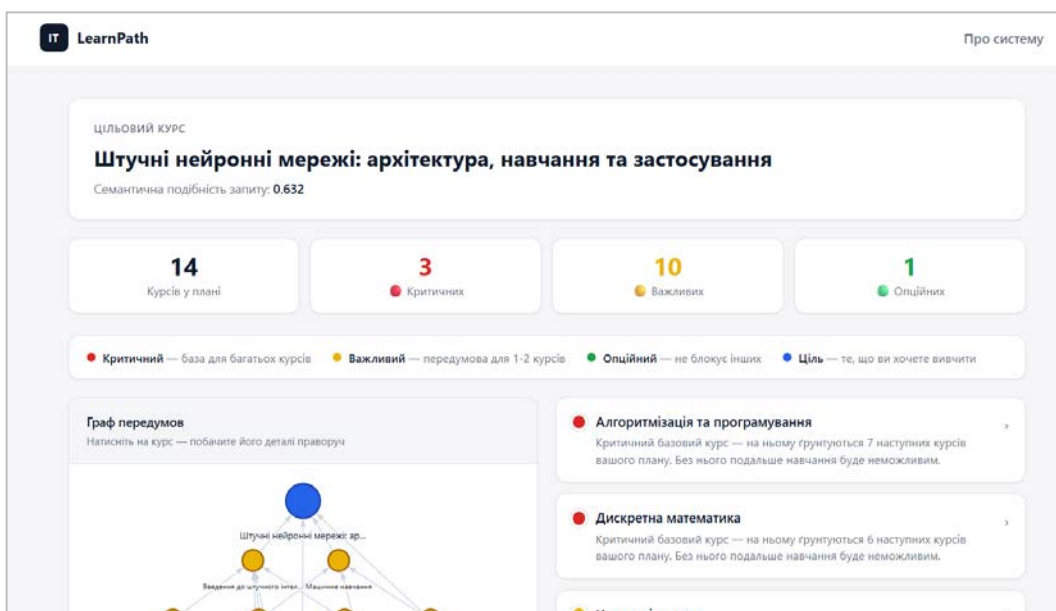


Рисунок 4.2 – Зведена панель побудованого плану

Сам план показано у дві колонки. Ліва колонка відображає план як інтерактивний граф передумов, у якому вершини відповідають дисциплінам, колір вершини – рівню важливості, а напрям стрілок – порядку вивчення. Права колонка показує ті самі дисципліни у вигляді карток із поясненнями. Обидві колонки синхронізовано між собою.

Взаємодію з планом наведено на рисунку 4.3. Коли користувач наводить курсор на дисципліну або клацає по ній – у графі чи на картці, – відповідний елемент у другій колонці підсвічується, а в графі виділяється шлях від обраної дисципліни до цілі. Картка дисципліни при цьому розкриває повне пояснення: рівень важливості, передумови, на які дисципліна спирається, дисципліни, які вона відкриває далі, компетенції та шлях до цілі.

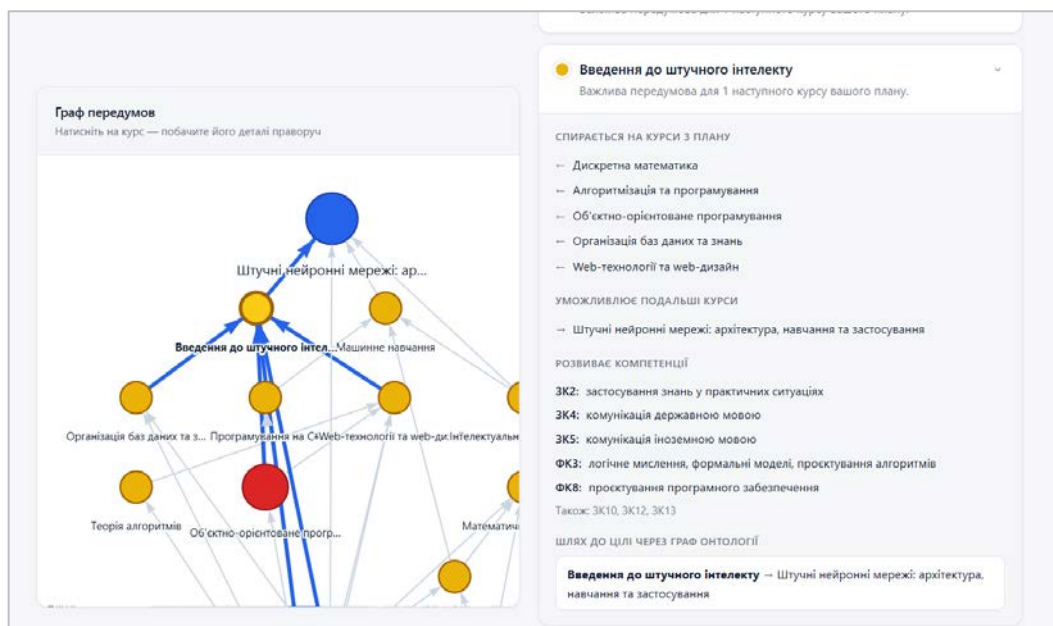


Рисунок 4.3 – Картка дисципліни з поясненням і виділеним шляхом до цілі

Завдяки такому поданню користувач може переходити від наочної структури плану в графі до докладного пояснення окремої дисципліни, не втрачаючи зв'язку між ними.

4.2 Демонстрація роботи системи

Роботу системи продемонстровано на трьох сценаріях: звичайному запиті, запиті, ціль якого користувач уже опанував, та некоректному запиті. Перший показує основний режим роботи, два інші – поведінку системи в межових випадках.

4.2.1 Звичайний запит

Розглянемо типовий сценарій. Користувач вводить запит природною мовою – «Хочу навчитися нейронним мережам».

На першому кроці модуль обробки запиту виконує семантичне зіставлення. Найближчою за змістом виявляється дисципліна «Штучні

нейронні мережі: архітектура, навчання та застосування» з подібністю 0,632. Це значення перевищує поріг 0,40, тож дисципліну прийнято за мету плану.

На другому кроці графовий модуль збирає передумови цілі, упорядковує їх і визначає важливість кожної. Побудований план містить 14 дисциплін: 3 критичні, 10 важливих і 1 опційну, а також 3 базові знання поза програмою. Розподіл за рівнями важливості показано на зведеній панелі (рисунок 4.2), а повний граф передумов цього плану – на рисунку 4.4. У графі видно, що цільова дисципліна спирається на кілька проміжних, а ті, своєю чергою, – на базові критичні дисципліни, від яких залежить найбільше курсів.

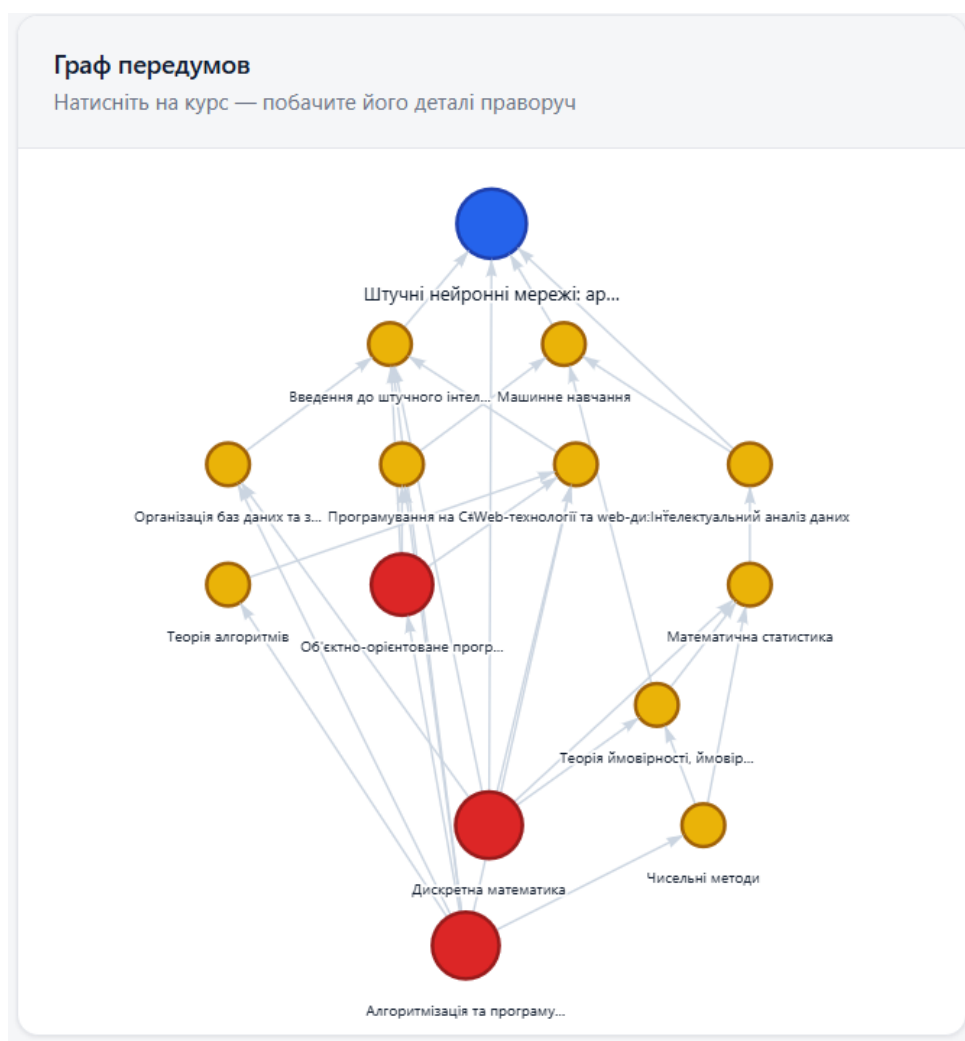


Рисунок 4.4 – Граф передумов плану для запиту про нейронні мережі

На третьому кроці для кожної дисципліни формується пояснення. Наприклад, дисципліну «Дискретна математика» позначено критичною, бо на ній ґрунтуються 6 інших дисциплін плану; її пояснення має вигляд: «Критичний базовий курс – на ньому ґрунтуються 6 наступних курсів вашого плану. Без нього подальше навчання буде неможливим». Поряд із причиною пояснення містить перелік дисциплін, які ця дисципліна відкриває, і компетенції, які вона розвиває.

Цей сценарій демонструє повний цикл роботи системи: від запиту природною мовою до впорядкованого плану з перевірними поясненнями.

4.2.2 Запит із відомою ціллю

Іноді дисципліна, що відповідає запиту, користувачеві вже відома, і він хоче дізнатися, що вивчати далі в цьому напрямі. Для такого випадку в системі передбачено окремий стан – з відомою ціллю (підрозділ 2.4).

Розглянемо приклад. Користувач вводить запит «машинне навчання» і в блоці вже опанованих дисциплін зазначає, що цю дисципліну вже вивчив.

Модуль обробки запиту, як і раніше, визначає за змістом цільову дисципліну – «Машинне навчання». Проте система виявляє її серед відомих користувачеві, тому замість плану для її досягнення переходить у стан з відомою ціллю. У цьому стані система рекомендує дисципліни, що спираються на вже відому, – тобто ті, для яких вона є передумовою, наприклад «Штучні нейронні мережі». Відповідний екран наведено на рисунку 4.5. Такий підхід дозволяє реалізувати механізм прямого розширення знань на основі онтологічного графу, пропонуючи користувачеві вектор для подальшого професійного розвитку. Окрім візуалізації карток суміжних курсів, система генерує текстові пояснення (ХАІ), які обґрунтовують доцільність вивчення кожної наступної дисципліни через семантичний зв'язок «є передумовою для».

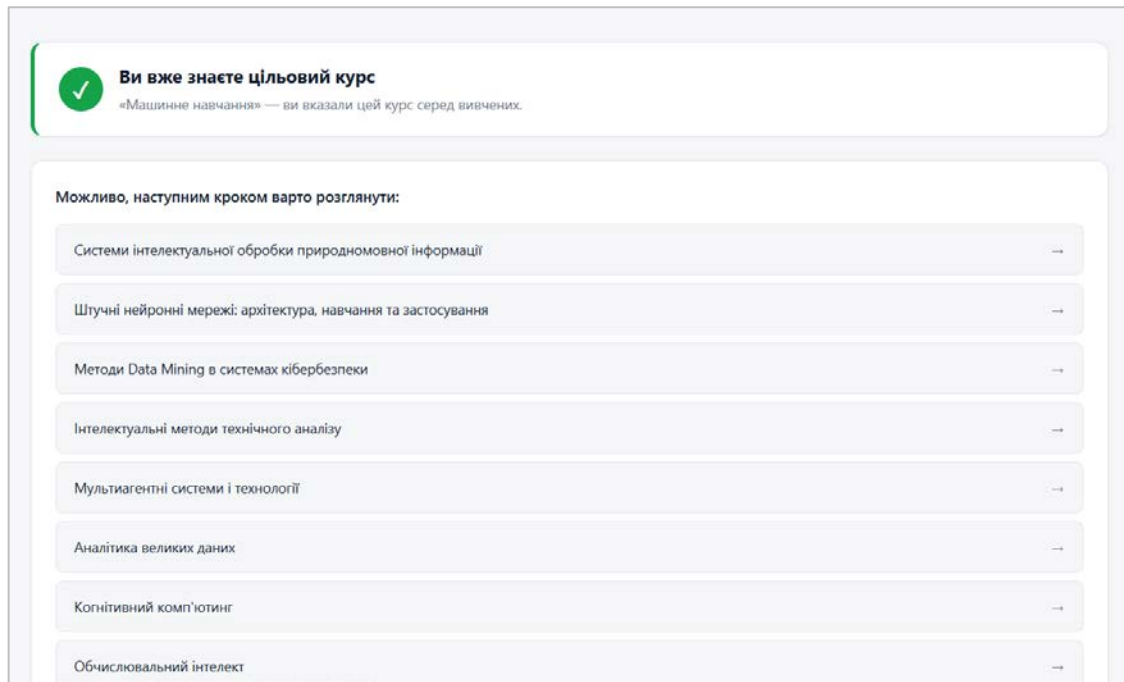


Рисунок 4.5 – Рекомендація наступних дисциплін, коли ціль уже відома

Цей сценарій показує, що система корисна не лише початківцям. Користувачеві з попередньою підготовкою вона підказує наступний крок, а не повторює вже відомий матеріал.

4.2.3 Обробка некоректного запиту

Важливою властивістю системи є коректна поведінка на запитах, що не мають відповіді в предметній галузі. Розглянемо запит, не пов'язаний з ІТ-освітою, – «Як зварити борщ».

Модуль обробки запиту кодує цей запит і порівнює його з усіма дисциплінами. Оскільки запит не має смислової близькості до жодної ІТ-дисципліни, подібність навіть найближчої з них виявляється нижчою за поріг; доменних термінів у запиті теж не виявлено, тож знижений поріг не застосовується. Через це система не повертає випадкову дисципліну, а повідомляє, що відповідної дисципліни не знайдено. Відповідний екран наведено на рисунку 4.6.

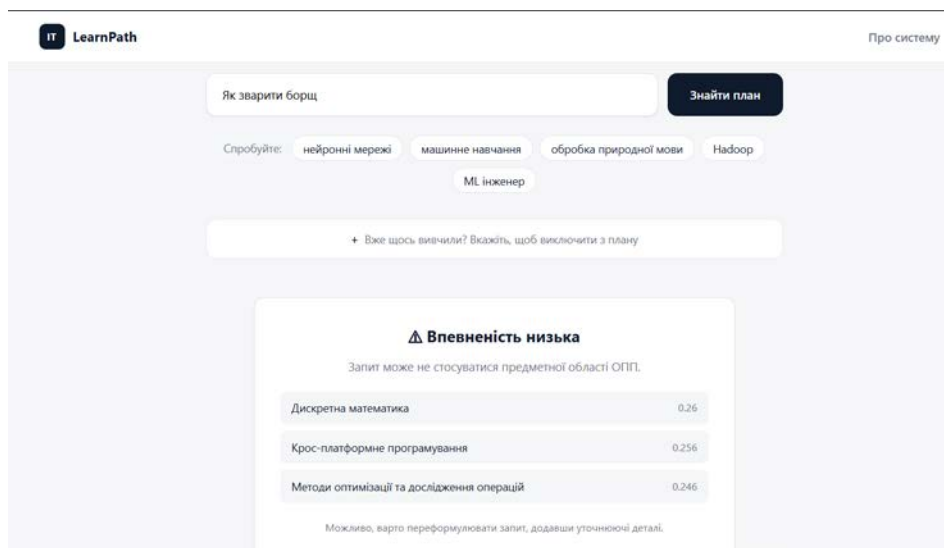


Рисунок 4.6 – Відмова системи на запит поза межами предметної галузі

Саме тут поріг впевненості діє як запобіжник. Цей сценарій також показує відмінність системи від мовної моделі загального призначення: на такий самий запит модель найімовірніше згенерувала б правдоподібний, але вигаданий план, тоді як ця система чесно відмовляє, коли запит виходить за межі предметної галузі. Така надійність є наслідком того, що відповідь спирається на онтологію, а не генерується (підрозділ 1.2).

4.3 Тестування та оцінювання

Тестування охопило три аспекти роботи системи: правильність семантичного зіставлення, коректність побудованого плану та швидкодію.

Правильність семантичного зіставлення перевірено на наборі типових запитів, серед яких «нейронні мережі», «машинне навчання» та «обробка природної мови». Для кожного релевантного запиту система визначила відповідну йому дисципліну: наприклад, для запиту про нейронні мережі цільовою стала дисципліна «Штучні нейронні мережі: архітектура, навчання та застосування» з подібністю 0,632. Натомість запити поза предметною галуззю, як-от «Як зварити борщ» (підрозділ 4.2.3), система

відхиляє, бо їхня подібність не досягає порога. Це підтверджує, що зіставлення відрізняє релевантні запити від нерелевантних.

Коректність плану означає, що він не порушує жодної передумови: кожна дисципліна стоїть після всіх своїх передумов, а в плані немає циклів. Цю властивість забезпечують два механізми. По-перше, резонатор HermiT підтвердив, що граф передумов онтології є ациклічним (підрозділ 3.2.3), тож коректне впорядкування завжди існує. По-друге, топологічне сортування під час побудови плану перевіряє, чи вдалося впорядкувати всі дисципліни, і позначило б результат некоректним інакше. Для всіх перевірених запитів усі дисципліни плану було впорядковано успішно, тобто отримані плани топологічно коректні.

Швидкодію виміряно на запиті про нейронні мережі; результати наведено в таблиці 4.1. Найбільші витрати часу припадають на одноразове завантаження моделі обробки мови, що виконується лише раз під час запуску системи. Саме ж обслуговування запиту відбувається швидко: семантичне зіставлення триває близько 0,19 с, побудова плану – менше за 0,01 с, а формування пояснень – близько 0,06 с. Таким чином, відповідь на запит система формує менш ніж за пів секунди; мала тривалість побудови плану узгоджується з лінійною складністю алгоритмів (підрозділ 2.4).

Таблиця 4.1 – Час виконання етапів обробки запиту

Етап	Час, с
Завантаження моделі (одноразово, під час запуску)	15–19
Семантичне зіставлення запиту	0,19
Побудова плану	< 0,01
Формування пояснень	0,06

Проведене тестування підтвердило, що система визначає релевантні дисципліни, відхиляє запити поза предметною галуззю, будує плани без порушення передумов і відповідає на запит за частки секунди.

ВИСНОВКИ

У роботі розроблено та програмно реалізовано інтелектуальну систему синтезу персональних освітніх планів, яка на основі запиту користувача природною мовою будує впорядковану послідовність дисциплін із поясненням кожного кроку. Поставлену мету досягнуто, а всі визначені у вступі завдання виконано.

Аналіз предметної галузі та наявних рішень показав, що жодне з них не поєднує одночасно природномовного інтерфейсу, перевірності рекомендацій, строгого врахування передумов та адаптації під користувача. Платформи онлайн-навчання не вибудовують індивідуального порядку, мовні моделі загального призначення не гарантують правильності, а академічні онтологічні системи не приймають запит звичайною мовою. На основі цього висновку обрано спеціалізований гібридний підхід, що поєднує сильні сторони цих груп.

Відповідно до обраного підходу спроектовано та реалізовано систему, що складається з онтологічної бази знань і трьох модулів. Онтологію освітньо-професійної програми побудовано з 53 дисциплін зі 139 зв'язками передумови; її передумови розв'язано нечітким зіставленням, а несуперечливість і відсутність циклів підтверджено резонатором HermiT. Модуль обробки запиту визначає цільову дисципліну за допомогою трансформерної моделі та семантичного зіставлення. Модуль побудови плану збирає передумови обходом графа, упорядковує їх топологічним сортуванням і позначає важливість дисциплін за кількістю залежних від них курсів. Модуль пояснення складає для кожної дисципліни обґрунтування виключно з фактів онтології.

Роботу системи продемонстровано на трьох сценаріях – звичайному запиті, запиті з відомою ціллю та некоректному запиті – і перевірено за трьома показниками. Тестування підтвердило, що система визначає

релевантні дисципліни, відхиляє запити поза предметною галуззю, будує плани без порушення передумов і формує відповідь менш ніж за півсекунди.

Основним результатом роботи є поєднання відомих методів – онтологічного інжинірингу, трансформерних моделей, графових алгоритмів і пояснюваного штучного інтелекту – у єдину систему, спеціально побудовану під синтез освітнього плану з прив'язкою до конкретної освітньо-професійної програми. Принциповою її властивістю є перевірність пояснень: кожне твердження складене лише з наявного факту онтології, тому система не може вигадати неправдиве обґрунтування – на відміну від генеративних моделей.

Робота має й обмеження. Базові знання поза програмою дедуплікуються за точним збігом тексту, через що серед них трапляються дублікати й артефакти парсингу; пояснення, своєю чергою, успадковує якість онтології – хибний зв'язок, отриманий нечітким зіставленням, буде відображено в поясненні. Ці обмеження визначають напрями подальшого розвитку: застосування нечіткого зіставлення й до базових знань, розширення онтології на кілька освітніх програм та інтеграцію відстеження проходження курсів користувачем.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Kosten C., Cudré-Mauroux P., Stockinger K. Spider4SPARQL: a complex benchmark for evaluating knowledge graph question answering systems. *2023 IEEE international conference on big data (bigdata)*, Sorrento, Italy, 15–18 December 2023. 2023. URL: <https://doi.org/10.1109/bigdata59044.2023.10386182> (date of access: 12.06.2026).
2. Chi Y.-L. Ontology-based curriculum content sequencing system with semantic rules. *Expert systems with applications*. 2009. Vol. 36, no. 4. P. 7838–7847. URL: <https://doi.org/10.1016/j.eswa.2008.11.048> (date of access: 14.06.2026).
3. Modern baselines for SPARQL semantic parsing / D. Banerjee et al. *SIGIR '22: the 45th international ACM SIGIR conference on research and development in information retrieval*, Madrid Spain. New York, NY, USA, 2022. URL: <https://doi.org/10.1145/3477495.3531841> (date of access: 18.06.2026).
4. Markchom T., Liang H., Ferryman J. Review of explainable graph-based recommender systems. *ACM computing surveys*. 2025. URL: <https://doi.org/10.1145/3772273> (date of access: 12.06.2026).
5. Complexity analysis and construction of conceptual graphs of mass open online courses / V. I. Zharebkin et al. *Bionics of intelligence*. 2023. Vol. 1, no. 99. P. 26–37. URL: [https://doi.org/10.30837/bi.2023.1\(99\).04](https://doi.org/10.30837/bi.2023.1(99).04) (date of access: 18.06.2026).
6. Володін В. А., Гриньова О. Є. Онтологічна система персоналізації навчальних планів засобами NLP. *Радіотехніка та молодь у XXI столітті* : матеріали 30-го Міжнар. молодіж. форуму, 22-24 квітня 2026 р. Харків : ХНУРЕ, 2026. Т. 6. С. 28–30.
7. Володін В. А., Гриньова О. Є. Онтологічна система персоналізації навчальних планів засобами NLP. *Сучасні інформаційні системи та*

технології в цифровому суспільстві : матеріали Міжнар. наук.-практ. конф., 16–17 квіт. 2026 р. Харків : ХНЕУ ім. Семе́на Кузне́ця, 2026. С. 92.

8. Coursera reports fourth quarter and full year 2025 financial results. *Coursera, Inc.* URL: <https://investor.coursera.com/overview/default.aspx> (date of access: 20.06.2026).

9. Analysis of the consistency of prerequisites and learning outcomes of educational programme courses by using the ontological approach / A. Nazyrova et al. *Applied sciences*. 2023. Vol. 13, no. 4. P. 2661. URL: <https://doi.org/10.3390/app13042661> (date of access: 20.06.2026).

10. A courses ontology system for computer science education / Y. Wang et al. *2019 IEEE international conference on computer science and educational informatization (CSEI)*, Kunming, China, 16–19 August 2019. 2019. URL: <https://doi.org/10.1109/csei47661.2019.8938930> (date of access: 13.06.2026).

11. Cheng Y., Nunes B. P. The use of semantic technologies in computer science curriculum: a systematic review. *arXiv preprint arXiv:2205.00462*. 2022. URL: <https://doi.org/10.48550/arXiv.2205.00462> (date of access: 10.05.2026).

12. Enhancing SPARQL query generation for knowledge base question answering systems by learning to correct triplets / J. Qi et al. *Applied sciences*. 2024. Vol. 14, no. 4. P. 1521. URL: <https://doi.org/10.3390/app14041521> (date of access: 18.06.2026).

13. Liu Y., Zhang Y., Zhang G. Learning path recommendation based on Transformer reordering. *2020 5th international conference on information science, computer technology and transportation (ISCTT)*, Shenyang, China, 13–15 November 2020. 2020. URL: <https://doi.org/10.1109/isctt51595.2020.00025> (date of access: 18.06.2026).

14. Li A., Li Y., Gao X. Personalized learning path recommendation based on knowledge graphs: a survey. *Electronics*. 2026. Vol. 15, no. 1. P. 238. URL: <https://doi.org/10.3390/electronics15010238> (date of access: 14.06.2026).

15. Moon J. Symmetric graph-based visual question answering using neuro-symbolic approach. *Symmetry*. 2023. Vol. 15, no. 9. P. 1713. URL: <https://doi.org/10.3390/sym15091713> (date of access: 15.06.2026).
16. OWL 2 web ontology language document overview (second edition). W3C. URL: <https://www.w3.org/TR/owl2-overview/> (date of access: 15.06.2026).
17. HermiT: an OWL 2 reasoner / B. Glimm et al. *Journal of automated reasoning*. 2014. Vol. 53, no. 3. P. 245–269. URL: <https://doi.org/10.1007/s10817-014-9305-1> (date of access: 15.06.2026).
18. Jurafsky D., Martin J. H. Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition. 3rd ed., draft. Stanford : Stanford University, 2024. 636 p. URL: <https://web.stanford.edu/~jurafsky/slp3/> (date of access: 06.03.2026).
19. Attention is all you need / A. Vaswani et al. *arXiv*. 2023. URL: <https://doi.org/10.48550/arXiv.1706.03762> (date of access: 15.06.2026).
20. Reimers N., Gurevych I. Sentence-BERT: sentence embeddings using siamese bert-networks. *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, Hong Kong, China. Stroudsburg, PA, USA, 2019. URL: <https://doi.org/10.18653/v1/d19-1410> (date of access: 11.06.2026).
21. Cormen T. H., Stein C., Rivest R. L. Introduction to algorithms, fourth edition. MIT Press, 2022. 1332 p.
22. Lamy J.-B. Owlready: ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial intelligence in medicine*. 2017. Vol. 80. P. 11–28. URL: <https://doi.org/10.1016/j.artmed.2017.07.002> (date of access: 18.06.2026).