Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Центр ННЦЗФН
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

LLM-орієнтована система «SemanticLib»: NL-to-SPARQL та Rhizomer Visualization технології
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІз-21-1
Дар'я Паславська
(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 _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Центр ННЦЗФН

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 Паславській Дар'ї Олександрівні
(прізвище, ім'я, по батькові)

1. Тема роботи LLM-орієнтована система «SemanticLib»: NL-to-SPARQL та Rhizomer Visualization технології

затверджена наказом університету від 7 травня 20 25 р. № 80Стз

2. Термін подання студентом роботи до екзаменаційної комісії 24 червня 20 25 р.

3. Вихідні дані до роботи Науково-технічні публікації, аналіз предметної галузі, набір даних для тренування та тестування системи, опис технічного рішення та запропоновані підходи до вирішення проблеми

4. Перелік питань, що потрібно опрацювати в роботі

1) Аналіз предметної галузі

2) Проектування LLM-інтегрованого Rhizomer

3) Інтеграція LLM з компонентами Rhizomer

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	05.05.2025	виконано
2	Аналіз літератури за темою	07.05.2025	виконано
3	Аналіз предметної галузі	10.05.2025	виконано
4	Постановка задачі	10.05.2025	виконано
5	Опис технічного рішення	13.05.2025	виконано
6	Програмна реалізація експерименту	21.05.2025	виконано
7	Опис програмної реалізації	22.05.2025	виконано
8	Аналіз отриманих результатів	23.05.2025	виконано
9	Оформлення пояснювальної записки	25.05.2025	виконано
10	Перевірка на академічний плагіат	27.05.2025	виконано
11	Нормоконтроль	30.05.2025	виконано
12	Підготовка презентації та доповіді	07.06.2025	виконано
13	Попередній захист	09.06.2025	виконано
14	Рецензування	12.06.2025	виконано
15	Захист перед ЕК	24.06.2025	виконано

Дата видачі завдання 5 травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Олена Гриньова
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 67 с., 10 рис., 1 дод., 15 джерел.

ВЕЛИКА МОВНА МОДЕЛЬ, ДАНІ RDF, НАУКОВІ ПУБЛІКАЦІЇ СТУДЕНТІВ, СЕМАНТИЧНІ ДАНІ, СИСТЕМА RHIZOMER, NL- to- SPARQL.

Об’єкт дослідження – розширення системи Rhizomer функціональністю обробки природномовних запитів до RDF даних наукових публікацій студентів за допомогою LLM.

Предмет дослідження – технології NL-to-SPARQL на основі LLM и Semantic Web, застосовані для розширення функціональності системи Rhizomer в контексті пошуку RDF даних наукових публікацій студентів.

Мета роботи – розробка та інтеграція LLM-орієнтованої підсистеми в систему Rhizomer для забезпечення можливості пошуку RDF даних наукових публікацій студентів природною мовою.

Методи дослідження – аналіз літератури, системний аналіз, проектування та розробка, а також експериментальне тестування для досягнення поставленої мети.

Взаємозв’язок з іншими роботами – робота розвиває дослідження в напрямку NL-to-SPARQL и Semantic Web.

Значимість роботи – спрощення доступу до семантичних даних, підвищення ефективності пошуку, розширення функціональності Rhizomer, демонстрація потенціалу LLM для Semantic Web, можливість застосування в освітній та науковій сферах.

У результаті роботи в систему Rhizomer успішно інтегровано LLM-підсистему, що дозволило реалізувати функціонал пошуку RDF даних про наукові публікації студентів за допомогою запитів природною мовою.

ABSTRACT

Bachelor’s thesis contains: 67 pp., 10 fig., 1 ann., 15 references.

LARGE LANGUAGE MODEL, NL-to-SPARQL, RDF DATA, RHIZOMER SYSTEM, SCIENTIFIC PUBLICATIONS OF STUDENTS.

The object of the research is to expand the Rhizomer system with the functionality of processing natural language queries to RDF data of students' scientific publications using LLM.

The subject of the research is NL-to-SPARQL technologies based on LLM and Semantic Web, applied to expand the functionality of the Rhizomer system in the context of RDF data search of students' scientific publications.

The purpose of the work is to develop and integrate an LLM-oriented subsystem into the Rhizomer system to enable RDF data search of students' scientific publications in natural language.

Research methods – literature review, systems analysis, design and development, as well as experimental testing to achieve the goal.

The significance of the work is simplifying access to semantic data, increasing search efficiency, expanding the functionality of Rhizomer, demonstrating the potential of LLM for Semantic Web, the possibility of application in the educational and scientific fields.

As a result of the work, the LLM subsystem was successfully integrated into the Rhizomer system, which made it possible to implement the functionality of searching RDF data on students' scientific publications using queries in natural language.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Angular – це фреймворк, який надає розробникам зручні інструменти для створення сучасних вебдодатків;

API – Application Programming Interface – інтерфейс програмування застосунків;

Blank Nodes – порожні вузли;

Deepseek – китайська компанія в галузі штучного інтелекту, яка розробляє великі мовні моделі з відкритим кодом;

Global Text Search – глобальний текстовий пошук;

GPT – Generative Pre-trained Transformer – генеративний попередньо тренований трансформер, чат-бот та віртуальний помічник з генеративним штучним інтелектом, розроблений компанією OpenAI;

JavaScript – динамічна, об'єктно-орієнтована прототипна мова програмування;

Jena-Fuseki – трійкове сховище;

JSON-LD – JavaScript Object Notation for Linked Data – об'єктна нотація JavaScript для пов'язаних даних, один із методів передачі зв'язаних даних з використанням текстового формату JSON;

Linked Data – методологія організації та зв'язування даних в інтернеті;

Llama – Large Language Model Meta AI – велика модель мови (LLM) випущена Meta AI;

LLM – Large Language Model – велика мовна модель;

NL – Natural Language;

Python – мова програмування високого рівня із суворою динамічною типізацією;

RDF – Resource Description Framework – структура опису ресурсу;

RDF/XML – стандарт для обміну даними XML;

Rhizomer – платформа для візуалізації та дослідження семантичних даних;

RhizomerAPI – це система Rhizomer, що виступає як серверний компонент-посередник, обробляючи запити від RhizomerEye, до сховища даних, та повертаючи результати назад до клієнтської частини, генеруючи та виконуючи SPARQL запити;

RhizomerEye – це система Rhizomer, що відповідає за інтерфейс користувача та реалізовує візуалізацію семантичних вебданих;

Semantic Web – семантичне павутиння;

Seq2Seq – Sequence-to-Sequence – це підходи машинного навчання, що використовуються для обробки природної мови;

SPARQL – Protocol and RDF Query Language – мова запитів до даних, представлених за моделлю RDF, а також протокол передачі цих запитів і відповіді них;

Turtle – коротка потрійна мова RDF;

TypeScript – мова програмування, позиціонується як засіб розробки вебзастосунків, що розширює можливості JavaScript;

URI – Uniform Resource Identifier – уніфікований ідентифікатор ресурсу;

World Wide Web – найбільше всесвітнє багатомовне сховище інформації в електронному вигляді;

XML – EXtensible Markup Language – розширювана мова розмітки;

Zoom&Filter – масштабування та фільтрація.

ВСТУП

У сучасному світі обсяги наукових даних зростають експоненційно, і ефективний доступ до цих знань є ключовим для прогресу в різних галузях. Зокрема, дані наукових публікацій студентів, що відображають початок їх дослідницького шляху, є цінним ресурсом для аналізу трендів, пошуку потенційних співробітників та підтримки освітнього процесу. Однак, традиційні методи пошуку часто виявляються неефективними для роботи з семантично структурованими даними, такими як RDF (Resource Description Framework – структура опису ресурсу).

Актуальність даної роботи обумовлена зростаючою потребою у спрощенні доступу до семантичних даних для користувачів, які не володіють спеціалізованими мовами запитів, такими як SPARQL (мова запитів до даних, представлених за моделлю RDF, а також протокол передачі цих запитів і відповіді них). Інтеграція великих мовних моделей (LLM) у існуючі системи, такі як Rhizomer, відкриває нові можливості для створення інтуїтивно зрозумілих інтерфейсів пошуку. Це дозволяє користувачам формулювати запити природною мовою та отримувати доступ до глибинної семантичної інформації безпосередньо, що є особливо важливим в освітньому та науково-дослідницькому середовищі.

Метою даної кваліфікаційної роботи є розробка та інтеграція LLM-орієнтованої підсистеми в систему Rhizomer для забезпечення можливості природномовного пошуку RDF даних про наукові публікації студентів. Система Rhizomer, як платформа для візуалізації та дослідження семантичних даних, є ідеальним середовищем для інтеграції NL-to-SPARQL функціональності на основі LLM. Очікується, що результати роботи знайдуть застосування в освітніх установах, університетських бібліотеках, наукових організаціях, а також у дослідницьких групах для покращення доступу до наукової інформації, підтримки дослідницької діяльності, сприяння міждисциплінарним зв'язкам та аналізу наукових трендів у

студентському середовищі. Розроблена система може стати основою для створення більш інтелектуальних інструментів для роботи з науковими даними, що сприятиме прискоренню наукових відкриттів та інновацій.

Варто зазначити, що розвиток технологій Semantic Web ta NL-to-SPARQL бере свій початок с концепції семантичної павутини Тіма Бернерса-Лі, спрямованої на створення більш інтелектуального та взаємопов'язаного вебу даних. Сучасний етап характеризується активним використанням досягнень у галузі штучного інтелекту, зокрема LLM, для подолання бар'єрів у використанні семантичних технологій для широкого кола користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Semantic Web та RDF даних

Semantic Web, або семантичне павутиння, є концепцією розширення World Wide Web, метою якої є надання даним структури та семантики, що дозволяє комп'ютерам краще розуміти та обробляти інформацію [1]. В основі Semantic Web лежать технології, що забезпечують представлення даних у машиночитаному форматі та встановлення семантичних зв'язків між ними.

Експоненційне зростання обсягів даних у сучасному цифровому просторі вимагає нових підходів до їх організації, пошуку та аналізу. Традиційні методи, що добре зарекомендували себе у роботі з реляційними базами даних або неструктурованим текстом, часто виявляються недостатньо ефективними для розкриття глибинного семантичного змісту, що прихований у масивах інформації. Semantic Web, запропонована Тімом Бернерсом-Лі, постає відповіддю на цей виклик, пропонуючи бачення вебу даних, де інформація не лише пов’язана гіперпосиланнями, але й наділена семантикою, зрозумілою для комп’ютерів [2].

В епоху інформаційного перенасичення, коли обсяги цифрових даних зростають у геометричній прогресії, виникає нагальна потреба не лише у зберіганні та передачі інформації, але й у її комп’ютерній систематизації. Саме на вирішення цих викликів спрямована концепція, що постає як амбітна ініціатива розширення можливостей World Wide Web. За задумом її творця, Тіма Бернерса-Лі, Semantic Web має перетворити Інтернет з мережі документів на мережу даних, де інформація є не просто набором символів, а структурованою та семантично розміщеною сукупністю знань, зрозумілою як для людей, так і для машин [3]. На відміну від традиційного вебу, де інформація переважно представлена у вигляді HTML-документів, орієнтованих на візуальне сприйняття людиною, Semantic Web прагне до

створення, де інформація представлена у структурованому форматі, що дозволяє комп’ютерам розуміти дані, а не лише їхній вигляд. Це відкриває шлях до створення інтелектуальних додатків, здатних автоматично знаходити, інтегрувати та аналізувати інформацію з різних джерел, робити логічні висновки та надавати користувачам більш релевантні та персоналізовані результати [4]. Метою семантичного паутиння є створення глобального простору взаємопов’язаних даних, де інформація є машиночитаною інтерпретованою та придатною для автоматизованої обробки. Semantic Web прагне до створення, де інформація представлена у структурованому форматі, що дозволяє комп’ютерам розуміти зміст даних, а не лише їхній вигляд. Це відкриває шлях до створення інтелектуальних додатків, здатних автоматично знаходити, інтегрувати та аналізувати інформацію з різних джерел, робити логічні висновки та надавати користувачам більш релевантні та персоналізовані результати. Концепція семантичної павутини є відповіддю на виклики, що постають перед людством у цифрову еру, прагнучи перетворити хаос розрізнених даних на доступну для ефективного використання організовану систему знань.

Semantic Web дозволяє представити дані у вигляді графу, де ресурси (суб'єкти) пов'язані між собою через властивості (предикати) та значення (об'єкти). RDF-трійки (Resource Description Framework), як фундаментальні будівельні блоки семантичної павутини, забезпечують гнучкий та розширюваний спосіб моделювання знань, що виходить за рамки жорстких схем реляційних баз даних. Буде детально розглянуто структуру RDF, основні компоненти, такі як URI, літерали та порожні вузли, а також різні формати серіалізації RDF даних, як Turtle, XML та JSON-LD [2]. Розуміння RDF є критично важливим для подальшого аналізу та розробки системи, адже саме RDF дані про наукові публікації студентів є об’єктом дослідження в даній роботі. Ці дані, що описують початок дослідницького шляху молодих науковців, є цінним ресурсом, потенціал якого може бути розкритий саме за допомогою семантичних технологій – технологія RDF.

RDF – це стандарт для представлення інформації у вигляді трійок (суб'єкт – предикат – об'єкт), що дозволяє описувати ресурси та їхні властивості. RDF дані організовуються у графи, де вузли представляють ресурси, а ребра – властивості або відношення між ресурсами. RDF забезпечує гнучкий та розширюваний спосіб представлення даних, що робить його придатним для інтеграції різнорідних джерел інформації [2].

RDF забезпечує ряд важливих переваг для представлення даних у семантичній павутині. RDF трійки, як фундаментальні будівельні блоки семантичної павутини, забезпечують гнучкий та розширюваний спосіб моделювання знань, що виходить за рамки жорстких схем реляційних баз даних [2]. Ця модель дозволяє описувати дані будь-якої складності та предметної області, не вимагаючи жорсткої попередньої схеми, також є стандартом, що сприяє інтероперабельності та об'єднанню даних з різних джерел, навіть якщо вони використовують різні схеми та формати. Завдяки цим властивостям, RDF є ідеальним форматом для представлення семантично структурованих даних, таких як дані про наукові публікації студентів, що є об’єктом дослідження в даній роботі.

Технологія RDF є не просто форматом даних, що забезпечує створення, обробку та обмін семантичними даними, важливими компонентами RDF системи, а є цілою системою стандартів та інструментів.

URI (уніфікований ідентифікатор ресурсу) – це фундаментальне поняття в Semantic Web, що відіграє ключову роль у глобальній ідентифікації взаємопов'язаностей ресурсів в семантичній павутині. Простіше кажучи, URI – це унікальна адреса для будь-чого в вебі даних для однозначної ідентифікації ресурсів у глобальному масштабі Semantic Web. Уявіть собі, що ви хочете посилатися на певний об'єкт, будь то реальна річ, концепція, документ або навіть абстрактна ідея, в контексті Semantic Web, для цього вам потрібен URI, який би дозволив комп'ютерам та іншим системам точно розрізняти цей об'єкт від усіх інших. Саме цю функцію виконує URI унікальний та однозначний ідентифікатор, який дозволяє

однозначно посилатися на нього з будь-якої точки вебу. Використання URI є фундаментальним принципом Semantic Web, що забезпечує унікальність, ідентичність та взаємопов'язаність даних. URI забезпечує глобальну ідентичність ресурсів, дозволяючи однозначно посилатися на них з різних RDF документів та систем. URI є ідентифікатором, що дозволяє комп'ютерним системам автоматично обробляти та інтерпретувати RDF дані [5].

Literals (літерали) – об'єкти RDF трійок можуть бути не лише ресурсами, але й такими як рядки, числа, дати тощо. Це один з двох основних типів, які можуть бути присутні в RDF трійках. Основна функція літералів в RDF – це можливість представлення конкретних значень даних у RDF графах, таких як текстові рядки, числа, дати, тощо. Літерали дозволяють приписувати атрибути ресурсам, надаючи їм конкретні значення. Якщо ресурси представляють собою сутності та об'єкти, то літерали служать для надання цим ресурсам атрибутів та властивостей у вигляді безпосередніх даних. Коротко кажучи, літерали в RDF використовуються для представлення даних у вигляді значень, а не у вигляді зв'язків між ресурсами. RDF розрізняє кілька типів літералів, щоб забезпечити гнучкість та точність представлення даних: прості літерали, типізовані, мовні. Таким чином, забезпечуючи можливість представлення літерали є невід'ємною частиною RDF значень даних у RDF графах, з урахуванням їхнього типу мови, що є критично важливим для створення семантично багатих практично корисних даних у Semantic Web [1].

Blank Nodes (порожні вузли) – RDF дозволяє використовувати для представлення ресурсів, які не мають глобального URI ідентифікатора, але існують локально в межах RDF графу. Порожні вузли корисні для представлення анонімних або внутрішніх структур даних, наприклад, списків або складних об'єктів, що не потребують глобальної ідентифікації. На відміну від звичайних ресурсів, які ідентифікуються унікальними URI та існують у глобальному просторі даних, порожні вузли не мають

глобального URI ідентифікатора, в межах конкретного RDF графу, де вони визначені. У світі RDF, більшість ресурсів, які ми хочемо описати, мають чітку ідентичність та можуть бути однозначно визначені в глобальному контексті. Наприклад, наукова публікація, автор, організація – це ресурси, які ми можемо ідентифікувати за допомогою URI, що дозволяє посилатися на них з будь-якого місця у вебі даних [6].

Однак, існують ситуації, коли нам потрібно представити в RDF графі ресурси, які не потребують або не мають глобальної ідентичності.

Анонімні сутності – ресурси, про які ми хочемо сказати щось, але не важливо або неможливо дати їм глобальне ім'я. Наприклад, ми можемо знати, що «публікація має автора», але не мати URI для цього конкретного автора, якщо в контексті важлива лише наявність автора, а не ідентифікація конкретної особи.

Внутрішні структури даних – складні об'єкти, що складаються з кількох частин, або структури даних, як списки, контейнери, проміжні вузли, які використовуються для організації інформації в RDF графі, але самі по собі не є незалежними сутностями, що потребують глобальної ідентифікації.

Екзистенційне квантування – вираження тверджень про існування чогось без необхідності називати це щось конкретно. Наприклад, «існує автор цієї публікації», без необхідності вказувати, хто саме є автором. Саме для представлення таких ситуацій і використовуються порожні вузли, вони дозволяють створювати локальні ідентифікатори в межах RDF графу, які є унікальними лише в контексті цього графу, але не мають глобальної унікальності URI. Коли ми хочемо описати ресурс, не надаючи йому глобального URI, порожні вузли стають необхідними. Наприклад, при описі бібліографічних даних, ми можемо знати, що публікація має автора, але не мати інформації про URI автора або не бажати створювати URI для кожного автора, якщо в даному контексті це не потрібно. Порожні вузли дозволяють виразити факт існування ресурсу та його властивості, навіть якщо його

глобальна ідентичність невідома або нерелевантна. Порожні вузли можна використовувати для вираження, тобто тверджень про існування чогось, не називаючи це щось конкретно. Наприклад, твердження «існує книга», написана автором «Іван Петренко» можна представити за допомогою порожнього вузла, що представляє анонімну книгу, та зв'язати її з автором «Іван Петренко» через властивість «автор». Таким чином, ми стверджуємо існування книги з певним автором, не надаючи книзі унікальний URI.

Порожні вузли є важливим та потужним інструментом в RDF, що розширює можливості моделювання даних. Вони дозволяють представляти анонімні ресурси, внутрішні структури даних та виражати екзистенційні твердження, роблячи RDF модель більш гнучкою та придатною для широкого спектру застосувань. Проте, важливо використовувати порожні вузли, враховуючи їхні переваги та недоліки, та зважаючи на контекст задачі моделювання даних. У багатьох випадках, особливо коли мова йде про представлення даних про наукові публікації, порожні вузли можуть бути корисними для опису складних аспектів даних, таких як списки ключових слів, авторські колективи або структуровані адреси, роблячи RDF дані більш повними та виразними [6].

Формати серіалізації RDF – це способи, якими можна зберігати, передавати мережею, та обробляти програмне забезпечення. Іншими словами, це для запису абстрактної RDF моделі у фізичному форматі, представлення RDF даних у вигляді файлів або потоків байтів в конкретні синтаксичні правила. Уявіть собі RDF як мову для опису даних у вигляді трійок. Щоб записати текст цією мовою на папері або зберегти його у комп'ютерному файлі, нам потрібен – алфавіт, граматика та правила запису форматів серіалізації.

Формати серіалізації RDF потрібні щоб RDF дані могли бути корисними на практиці, їм потрібно представити концепцією у конкретному форматі:

– формати серіалізації дозволяють зберігати RDF-дані у файлах на жорсткому диску, у базі даних, або в іншому сховищі;

– формати серіалізації дозволяють, наприклад, через Інтернет, використовуючи HTTP протокол передавати RDF дані між різними системами та додатками;

– формати серіалізації дають можливість програмам читати та аналізувати RDF дані. Програмне забезпечення використовує формати серіалізації для перетворення текстового або бінарного представлення RDF даних у внутрішню модель даних, з якою воно може працювати, таку як графова структура в пам'яті, що дозволяє здійснювати маніпуляції, запити та аналіз семантичних даних. Також для того, щоб зчитати RDF дані з файлів, передати їх по мережі, та представити їх у вигляді, зрозумілому для програмної обробки, або для того, щоб представити абстрактну RDF модель у конкретному, фізичному форматі, який можна зберігати та обробляти комп'ютером.

Існує кілька форматів серіалізації RDF, кожен з яких пропонує свій баланс між поширених форматів серіалізації RDF, читабельністю для людини, та ефективністю для машини. Вибір конкретного формату часто залежить від цілей використання RDF даних та інструментів, що будуть застосовуватися для їх обробки. Розглянемо їх детальніше в залежності від конкретних потреб проекту [7].

Turtle (коротка потрійна мова RDF). Текстовий формат базується на текстових символах, що робить його придатним для редагування та перегляду у текстових редакторах. Синтаксис Turtle розроблений таким чином, щоб бути максимально зрозумілим та легким для читання, навіть для тих, хто не є експертом у RDF. Turtle має високу виразність, підтримує, включаючи ресурси, літерали (типізовані та мовні), порожні вузли, колекції RDF та інші розширені можливості RDF моделі. Підтримується більшістю RDF бібліотек та трійкових сховищ. Менш поширений у веброзробці, ніж JSON-LD, хоча Turtle є дуже популярним у спільноті Semantic Web [7].

RDF/XML є стандартом для обміну даними XML, використовує XML синтаксис, що забезпечує сумісність з існуючими XML інструментами та інфраструктурою. Був одним з перших форматів серіалізації RDF та відіграв важливу роль у розвитку Semantic Web. Цей формат підтримує всі можливості RDF моделі, легко інтегрується з XML-орієнтованими інструментами та системами. Існує велика кількість інструментів, які можуть бути використані для обробки RDF/XML Але є ряд недоліків: синтаксис RDF/XML є складним та важким для читання та розуміння, особливо для складних RDF графів, великий розмір файлів, що може бути проблематичним для зберігання та передачі великих обсягів даних. В останні роки популярність RDF/XML зменшується на користь більш читабельних та ефективних форматів, таких як Turtle та JSON-LD [8].

JSON-LD (JSON для отримання пов'язаних даних) базується на тому, що він є стандартом для обміну даними в вебі JSON (об'єктна нотація javascript). Розроблений спеціально для легкої інтеграції з вебтехнологіями, вебсервісами та API, підтримує всі можливості RDF моделі, включаючи контексти для визначення префіксів та базових URI. Ідеальний формат для тих, хто працює з семантичними даними вебсервісами, API та вебдодатками. Простота обробки в javascript – легко парситься та генерується в інших веборієнтованих мовах. Підтримується багатьма вебфреймворками, бібліотеками та інструментами [9].

N-Triples та N-Quad орієнтовані на максимальну простоту парсингу та обробки. Кожна RDF-трійка (або четвірка в N-Quads) представлена на дуже простому окремому рядку. N-Quads підтримує іменовані графи. N-Quads є розширенням N-Triples, що дозволяє представляти, додаючи ідентифікатор графу до кожної трійки. Ці формати є програмним забезпеченням, що роблять їх придатними для обробки найшвидшими та найефективнішими для парсингу дуже великих обсягів RDF даних, генерація N-Triples/N-Quads не вимагає складних алгоритмів. Але для людей вони є дуже важкими для читання та розуміння, особливо для складних RDF графів, тому через низьку

читабельність непридатні для ручного створення або редагування RDF даних [10].

Turtle є найбільш збалансованим та оптимальним форматом серіалізації RDF для мого дипломного проекту. Він забезпечує, чудову читабельність, компактність та виразність, легкість використання з Python rdflib, що робить його ідеальним вибором для представлення RDF даних про наукові публікації студентів.

1.2 Підходи до перетворення природної мови в SPARQL (NL to SPARQL)

Якщо RDF надає спосіб представлення семантичних даних, то SPARQL (SPARQL Protocol and RDF Query Language) є мовою, що дозволяє видобувати знання з цих даних. SPARQL, як стандартна мова запитів для RDF, відіграє ключову роль у екосистемі Semantic Web, забезпечуючи можливість пошуку, фільтрації та агрегації інформації, представленої у RDF графах. SPARQL – це мова семантичних запитів для баз даних, здатна отримувати та маніпулювати даними, зберігається у форматі RDF.

Проте, складність SPARQL для користувачів, що не є фахівцями у семантичних технологіях, створює певний бар'єр для широкого використання потенціалу Semantic Web. Саме тут на сцену виходить перетворення запитів, сформульованих природною мовою, у запити SPARQL. В цьому розділі розглянемо як традиційні підходи, що базуються на правилах та шаблонах, так і сучасні методи, що використовують машинне навчання та нейронні мережі. Особливий акцент зробимо на LLM (Large Language Model), з їхнім вражаючим прогресом у розумінні та генерації природної мови, які відкривають нову еру в інтеракції людини з семантичними даними. Далі дослідимо різні підходи до використання LLM для NL-to-SPARQL, включаючи prompt engineering, few-shot learning, fine-tuning та гібридні методи, поєднуючи LLM з символічними підходами.

Проаналізуємо переваги та недоліки кожного з підходів, а також існуючі інструменти та фреймворки, що реалізують функціональність NL-to-SPARQL Важливим аспектом є висвітлення ролі SPARQL у контексті задачі доступу до даних наукових публікацій студентів, демонструючи, як SPARQL дозволяє формулювати складні запити, що виходять за межі можливостей традиційного текстового пошуку.

Перетворення природної мови в SPARQL (NL-to-SPARQL) є ключовою задачею для забезпечення інтуїтивно зрозумілого доступу до семантичних даних. Метою NL-to-SPARQL є створення мосту між світом людської мови та формальною мовою запитів SPARQL, дозволяючи користувачам формулювати запити на природній мові, а системі автоматично перетворювати їх у відповідні SPARQL-запити для отримання даних с RDF графів. Історично, розробка ефективних NL-to-SPARQL систем була значним викликом у галузі Semantic Web, і протягом років було запропоновано різноманітні підходи, кожен с яких має свої переваги та обмеження. Розглянемо основні категорії підходів до NL-to-SPARQL.

1.2.1 Традиційні підходи

Ці підходи базуються на формальних методах обробки мови та чітко визначених правил, хоча вони можуть бути ефективними в обмежених доменах, їхня гнучкість та масштабованість є обмеженою.

Системи на основі правил (Rule-based Systems). Суть підходу полягає в тому, що використовують набір для аналізу природномовного запиту та його перетворення в SPARQL, правила можуть бути розроблені вручну експертами в предметній області та лінгвістиці. Приклад інструментів це Apache Jena. Пропонує інструменти для створення SPARQL-запитів на основі правил, також існують спеціалізовані системи, що використовують формальні граматики для NL-to-SPARQL. Перевагами є точність в обмежених доменах, можуть досягати високої точності для запитів, де

правила можна чітко сформулювати в вузько визначених предметних областях, прозорість та передбачуваність, логіка перетворення запитів є прозорою та зрозумілою, оскільки базується на явно заданих правилах [11];

Інтерфейси, керовані шаблонами (Template-driven Interfaces) надають користувачам контрольовану природну мову (CNL) або набір шаблонів- запитів, які вони можуть заповнювати, щоб сформувати запит. Система потім перетворює заповнений шаблон у SPARQL. В таких платформах як GINSENG і aqualog зменшується неоднозначність, обмежується варіативність виразів через контрольовану мову або шаблони користувача. Інтерфейси можуть бути інтуїтивно зрозумілими для користувачів, які не знайомі з SPARQL, але знають предметну область. Але з іншого боку контрольована мова або шаблони можуть обмежувати виразність запитів, не дозволяючи користувачам формулювати складні або нестандартні запити, обмежуючи їхню свободу вираження.

Підходи на основі ключових слів (Keyword-based Approachs). Цей підхід фокусується на природномовному запиті та його або RDF схемі, потім, на основі співставлених термінів, генерується SPARQL-запит. Підхід є відносно простим у реалізації та не вимагає складних лінгвістичних аналізаторів, обробка запитів може бути швидкою, особливо для простих пошукових запитів. Але він має низьку точність для складних запитів, що вимагають розуміння семантичних відношень, контексту та наміру користувача. Залежить від якості та повноти онтології. Ключові слова можуть бути неоднозначними та мати різні значення в різних контекстах, що може призводити до помилок у перекладі.

1.2.2 Підходи на основі машинного навчання

Сучасні підходи до NL-to-SPARQL все частіше використовують методи машинного навчання, особливо з появою потужних нейронних мереж та великих мовних моделей (LLM). Ці підходи прагнуть навчити

моделі автоматично перетворювати природномовні запити в SPARQL, використовуючи великі обсяги даних та знання, отримані з мовних моделей.

Ранні ML-моделі (моделі Sequence-to-Sequence). Моделі Seq2Seq, рекурентні нейронні мережі з механізмами уваги, для прямого перетворення природномовного запиту (вхідна послідовність) у SPARQL запит (вихідна послідовність). Приклади фреймворків – QUERIN, SPARQL2NL (хоча SPARQL2NL фокусувався на генерації природної мови с SPARQL, аналогічні моделі використовувалися і для NL-to-SPARQL). Моделі навчаються автоматично на даних, не вимагаючи ручного створення правил, можуть потенційно адаптуватися до різних типів запитів та предметних областей за умови наявності достатньої кількості навчальних даних. Ранні Seq2Seq моделі часто мали проблеми з генерацією синтаксично валідних та семантично коректних SPARQL запитів, особливо для складних запитів, що вимагають глибокого розуміння структури RDF даних та SPARQL мови. Ще моделі Seq2Seq часто є «чорними скриньками», що ускладнюють розуміння процесу перетворення запитів та налагодження помилок [12].

Семантичний аналіз з нейронними мережами (Semantic Parsing with Neural Networks) є сучасним та перспективним підходом, що використовує потужність для розуміння перетворення природної мови в SPARQL (NL-to-SPARQL) нейронних мереж семантики природномовного запиту. На відміну від традиційних підходів, що базуються на правилах або ключових словах, семантичний аналіз з нейронними мережами прагне до відображення її на формальну структуру SPARQL глибшого розуміння значення запиту, контексту та намірів користувача, щоб генерувати більш точні та релевантні SPARQL запити. Суть підходу полягає у наступному:

– розбір природномовного запиту на семантичні компоненти;

– відображення семантичних компонентів на RDF схему (онтологію);

– побудова логічного представлення запиту;

– генерація SPARQL запиту з логічного представлення.

Нейронні мережі відіграють ключову роль в автоматичному вивченні складних залежностей між природною мовою та семантичними структурами RDF в цьому підході. Нейронні мережі використовуються для різних підзадач семантичного аналізу, таких як розпізнавання іменованих сутностей NER (Named Entity Recognition), або Transformer-based моделі (наприклад, BERT, RoBERTa) виявлення та класифікації іменованих сутностей (людей, організацій, місць, дат, термінів предметної області) у природномовних запитах. Це дозволяє виділити ключові ресурси, про які йдеться в запиті, та зіставити їх з індивідами або класами в онтології. Також використовуються для розпізнавання відношень, виявлення семантичних відношень між сутностями. Нейронні мережі використовуються для класифікації наміру користувача, типу запиту, цілі запиту (наприклад, знайти публікації автора, знайти ключові слова публікації, порахувати кількість публікацій за рік). Розуміння наміру користувача є важливим для вибору відповідного типу SPARQL запиту (SELECT, CONSTRUCT, ASK, DESCRIBE) та його структури. Трансформаторні моди (BERT, RoBERTa, GPT) особливо ефективні у контекстному розумінні природної мови в реченні, вони дозволяють краще розуміти семантику та розв'язання неоднозначностей. Це дозволяє обробляти складні запити, що містять займенники, неявні посилання та інші лінгвістичні явища [12].

Переваги семантичного аналізу з нейронними мережами:

– нейронні мережі здатні досягати високої точності та виразності у NL-to-SPARQL, обробляючи складні та різноманітні запити користувачів;

– моделі, навчені на великих даних, є адаптивними до різних стилів запитів, формулювань та предметних областей. Тонке налаштування (до-навчання) моделей на даних конкретної предметної галузі дозволяє покращити їхню продуктивність для специфічних задач;

– нейронні мережі з навчальних даних, не вимагають ручного створення правил або шаблонів та автоматично вивчають знання про мову, семантику та RDF структури;

– галузь нейронних мереж та глибокого навчання активно розвивається, постійно з'являються нові архітектури, методи та інструменти, що покращують можливості NL-to-SPARQL систем.

Навчання ефективних нейронних мереж для NL-to-SPARQL вимагає великих обсягів паралельних даних для семантичного аналізу, що може бути складно отримати для деяких предметних областей. Виконання запитів з використанням складних нейронних мереж може вимагати значних обчислювальних ресурсів, особливо для великих моделей та великих обсягів даних. Хоча нейронні мережі досягли значного прогресу обробка дуже складних, неоднозначних або неповних запитів є викликом для сучасних NL-to-SPARQL систем.

Семантичний аналіз з нейронними мережами є потужним та перспективним підходом до NL-to-SPARQL автоматизації перетворення природномовних запитів у SPARQL інтелектуальних та зручних інтерфейсів, що використовує можливості глибокого навчання для постійного прогресу у галузі нейронних мереж, відкриває нові горизонти для створення доступу до семантичних даних, роблячи Semantic Web-технології більш доступними та корисними для широкого кола користувачів.

Підходи на основі великих мовних моделей (LLM-based Approaches). Суть підходу полягає у використанні великих мовних моделей (LLM) безпосередньої генерації SPARQL запитів, таких як GPT, Llama, DeepSeek. Великы мовны моделі є дуже гнучкими та адаптивними до різних стилів запитів, формулювань та предметних областей. LLM використовують величезні обсяги знань, отриманих під час попереднього навчання, що дозволяє їм розуміти мову та генерувати код без необхідності великих навчальних даних для кожної конкретної задачі.

Промпт – це вхідний текст або запит, який надається великій мовній моделі. Якість згенерованих SPARQL запитів сильно залежить від якості промптів, тому є потреба в ефективній інженерії промптів (prompt

engineering). Можливі помилки та неточності, LLM можуть генерувати синтаксично невалідні або семантично некоректні SPARQL запити, використання великих LLM може вимагати значних обчислювальних ресурсів API доступу.

1.2.3 Гібридні підходи

Для поєднання переваг різних підходів та зменшення їхніх недоліків використовують підходи, що комбінують традиційні методи з методами машинного навчання. Наприклад, система може використовувати гібридні підходи правила для попередньої обробки природномовного запиту та інший варіант – використання LLM для генерації SPARQL запиту на основі обробленого входу LLM для семантичного аналізу. Гібридні підходи дозволяють досягти правил для забезпечення синтаксичної валідності SPARQL коду кращого балансу між точністю, гнучкістю та ефективністю NL-to-SPARQL систем.

Вибір підходу до NL-to-SPARQL залежить від конкретних вимог до системи доступних ресурсів, складності предметної області та очікуваної якості результатів. Традиційні підходи можуть бути корисними для обмежених простих запитів, але для складних та різноманітних сценаріїв не є масштабованими. Підходи на основі машинного навчання, особливо LLM-базовані, відкривають здатність обробляти широкий спектр природномовних запитів та забезпечувати інтуїтивний доступ до семантичних даних. Однак, важливо враховувати нові можливості, обмеження та виклики для створення гнучких та потужних NL-to-SPARQL систем. Сучасні підходи, що інтегрують семантичний аналіз та нейронні мережі, досягли значного прогресу, проте ефективна обробка запитів, які вимагають складної логіки, все ще залишається проблематичною. Також, системи часто неявних відношень, часових обмежень, просторових міркувань, або комбінації декількох умов фільтрації недостатньо стійкі до

різноманітності природномовних формулювань, зокрема, синонімії, перифразування, та діалектних відмінностей, що призводить до зниження точності при обробці запитів від широкого кола користувачів.

1.3 Використання великих мовних моделей в NL-to-SPARQL технології

Великі мовні моделі (LLM), такі як GPT, Llama, deepseek та інші, продемонстрували значний прогрес у розумінні та генерації природної мови, їхні можливості у семантичному аналізі, контекстному розуміннні та генерації коду відкривають нові перспективи для NL-to-SPARQL. Великі мовні моделі (LLM) використовуються в багатьох областях обробки природної мови, завдяки своїй вражаючій здатності розуміти, генерувати та маніпулювати природною мовою. LLM використовують свою мову програмування з природномовних промптів, значно спрощуючи процес NL-to-SPARQL та розширюючи можливості для користувачів [13].

У стислому вигляді, процес генерації SPARQL запитів за допомогою LLM можна описати наступними кроками.

Прийом природномовного запиту (Prompt Input). Користувач вводить запит на природній мові, що описує інформацію, яку він хоче отримати з RDF даних.

Обробка запиту LLM (LLM Prompt Processing). Природномовний запит подається на вхід LLM, промпт може включати додатковий контекст, інструкції та приклади, щоб допомогти LLM краще зрозуміти задачу та очікуваний формат виводу (SPARQL).

Семантичне розуміння (Semantic Understanding). LLM використовує свої внутрішні механізми для розуміння семантики запиту, виявляючи наміри користувача, ключові сутності, відношення та обмеження та намагається відобразити ці семантичні компоненти на елементи RDF-схеми або онтології, з якою пов'язані RDF дані (класи, властивості, індивіди).

Генерація SPARQL запиту (SPARQL Query Generation).

Виведення SPARQL запиту. LLM повертає згенерований SPARQL запит як вихід. Цей SPARQL запит потім може бути направлен до трійкового сховища (наприклад, Jena Fuseki) для отримання результатів.

Серед переваг покоління SPARQL з LLM можна виділити:

– доступність, що дозволяє нетехнічним користувачам запитувати семантичні дані без SPARQL досвіду;

– гнучкість, що адаптується до варіацій виразів природної мови та намірів;

– контекстуальне розуміння, що інтерпретує неоднозначні запити на основі контексту має поширені моделі використання;

– використовує попередньо навчені знання для розуміння конкретної предметної області.

Існує кілька технічних підходів до використання LLM для генерації SPARQL, кожен з яких має свої особливості та рівень складності:

– Prompt Engineering. Розробка ефективних підказок, які спрямовують LLM на генерацію дійсних SPARQL синтаксисів;

– Few-Shot Learning. Використання прикладів для демонстрації очікуваного формату запиту та структури;

– постобробка – застосування виправлень на основі правил для забезпечення валідності запиту та покращення продуктивності.

Дослідницький внесок цього проекту усуває прогалини в сучасному стані техніки:

– інтеграція LLM в Rhizomer: розробка функції Zoom&Filter з новим сервером для створення запитів SPARQL. Інтеграція згенерованих запитів до проекту, виконання та візуалізація результатів у функції масштабування;

– оцінка BESDUI: застосування структурованих 12 тестів для кількісної оцінки продуктивності LLM у контекстах семантичного вебу. Розуміння того, які моделі здатні вирішувати задачі, які з них точніші;

– діагностика помилок: виявлення помилок у виконанні LLM, аналіз журналу та повідомлень про помилку, а потім аналіз продуктивності та стратегії вирішення помилок [13].

Додатковою перевагою великих мовних моделей є їхня здатність працювати без глибокого налаштування чи попередньої спеціалізованої підготовки. Завдяки загальним знанням, отриманим під час попереднього навчання, LLM здатні ефективно генерувати SPARQL-запити навіть у загальних сценаріях використання, де відсутні специфічні знання предметної області. Це робить їх особливо корисними у проєктах, де важлива швидка інтеграція та зниження порогу входу для користувачів, які не мають досвіду роботи з онтологіями чи семантичними структурами.

Вирішуючи ці проблеми, робота позиціонує LLM як життєздатні інструменти для генерації запитів SPARQL, зберігаючи зручність використання Rhizomer і розширюючи його доступ до користувачів без досвіду SPARQL.

1.4 Rhizomer

У світі, де обсяги семантичних даних постійно зростають, Rhizomer виступає як інтуїтивно зрозумілий та потужний інструмент. Завдяки йому можна полегшити доступ до Semantic Web даних, зробити їх доступними та зрозумілими для широкого кола користувачів, включаючи дослідників, аналітиків, розробників та кінцевих користувачів, незалежно від їхнього технічного досвіду.

Rhizomer, є вебдодатком, що полегшує публікацію, а також вивчення даних Semantic Web (і особливо Linked Data), веборієнтована платформа публікацій та досліджень даних [4], його інтерфейс показано на рисунку 1.1 та рисунку 1.2.

Рисунок 1.1 – Вигляд інтерфейсу застосунка

Rhizomer доступний через звичайний веббраузер без необхідності встановлення спеціального програмного забезпечення. Rhizomer оптимізован для дослідження Linked Data, тобто даних, що представлені у вигляді RDF графів та пов'язан з іншими даними у вебі за допомогою URI. Він дозволяє користувачам переходити за посиланнями між ресурсами,

досліджувати взаємозв'язки між різними наборами даних, та відкривати нові знання, приховані у мережі пов'язаних даних.

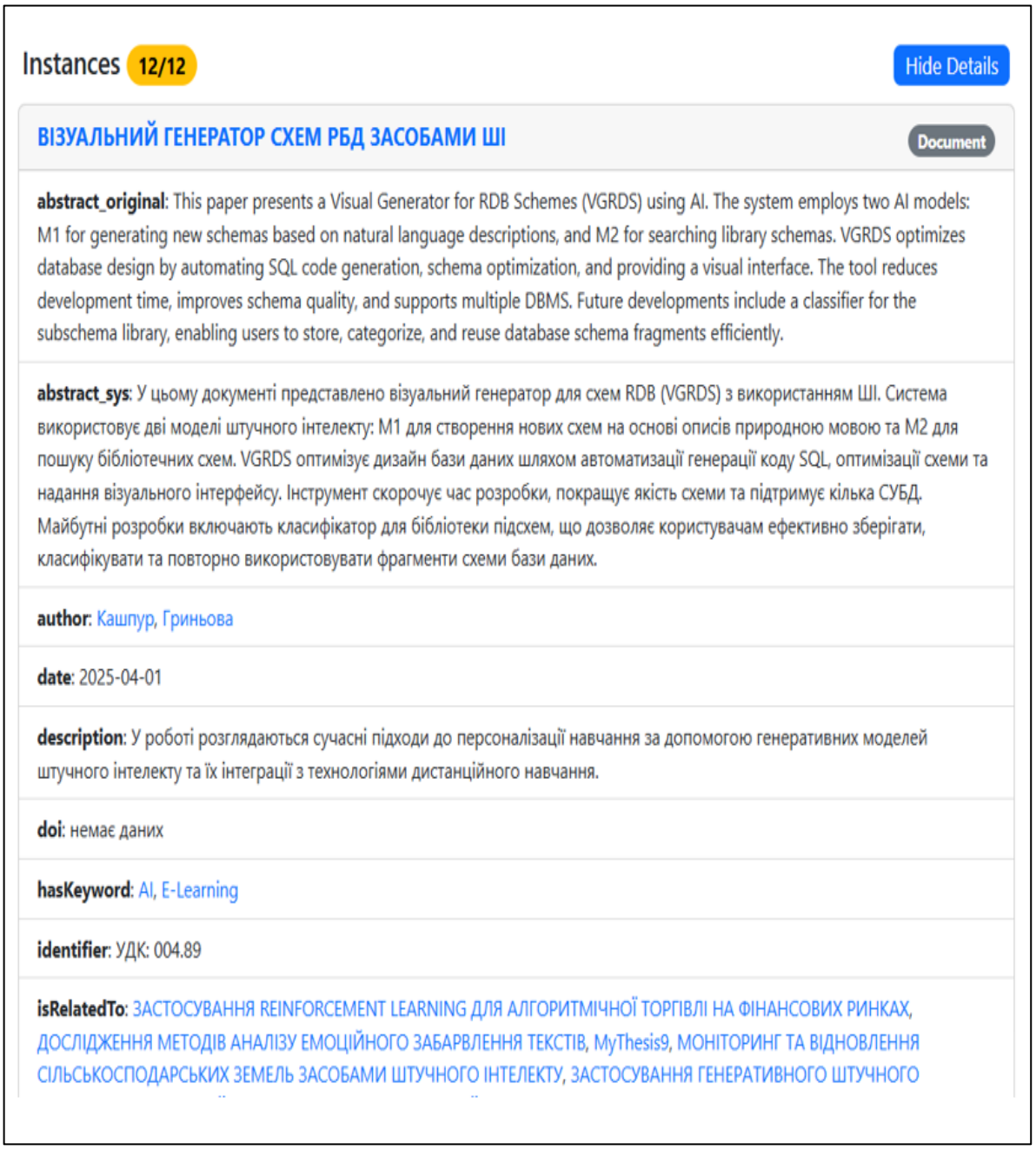

Рисунок 1.2 – Візуальна інтерпретація тезисів

Хоча Rhizomer прагне спростити доступ до семантичних даних для нетехнічних користувачів, однак SPARQL залишається основою для обробки та видобування даних.

Ключовими характеристиками Rhizomer є:

– вебдодаток, який забезпечує платформо-незалежний доступ через стандартний веббраузер, мінімізуючи вимоги до інфраструктури клієнта;

– спеціалізована платформа для навігації та аналізу взаємопов'язаних RDF даних, підтримуючи концепцію Linked Data шляхом забезпечення переходу;

– в основі функціональності Rhizomer лежить використання мови SPARQL для видобування та обробки RDF даних, хоча складність SPARQL є абстрагованою від кінцевого користувача через інтерактивний графічний інтерфейс;

– реалізація трикомпонентної моделі дослідження даних: інтерфейс структуровано відповідно до трьох фундаментальних задач аналізу даних (огляд, масштабування/фільтрація, деталізація), що забезпечує систематичний та послідовний процес дослідження для користувача.

Інтерфейс Rhizomer було запропоновано Шнейдерманом [4] та складається з трьох основних розділів – огляд (overview), збільшення та фільтрація (zoom and filter), деталі на запит (details-on-demand). Ці принципи дозволяють користувачам досліджувати семантичні знанняві графи, отримуючи загальне уявлення про структуру даних, фокусуючись на потрібних частинах і отримуючи детальну інформацію за потреби.

Overview (огляд) надає загальну картину набору даних через візуалізації, такі як хмари слів та мережеві графи, для швидкого розуміння структури.

Компонент Zoom&Filter (масштабування та фільтрація) дозволяє звужувати пошук та фокусуватися на конкретних частинах даних за допомогою різноманітних фільтрів, пошукових полів та діапазонів. Поля автозаповнення дозволяють за мітками полегшувати навігацію по онтології, швидко знаходити класи та властивості. Глобальний текстовий пошук (Global Text Search) дозволяє шукати текст по всьому набору даних, знаходячи ресурси, що містять задані ключові слова. Фасетна фільтрація

надає механізми фільтрації на основі значень властивостей, поступово звужуючи набір даних, дозволяє користувачам вибирати певні значення для різних властивостей. Числові діапазони: дозволяють фільтрувати ресурси за числовими властивостями в заданому діапазоні, наприклад, за роком публікації або чисельним значенням. Текстовий пошук в класі (Class- specific Text Search) дозволяє шукати текст лише в межах певного класу ресурсів, підвищуючи точність пошуку. Саме розділ Zoom&Filter є ключовим фокусом даної роботи природномовного пошуку, де інтегрується LLM-орієнтована підсистема для забезпечення та розширення можливостей фільтрації.

Details-on-demand (деталізація за запитом) відкриває детальну інформацію про окремі ресурси, включаючи метадані та зв'язки з іншими ресурсами, для глибшого вивчення обраних елементів.

Інтерфейс Rhizomer розроблений так, щоб бути доступним для користувачів без спеціальних знань про Semantic Web, пропонуючи візуальні інструменти та інтерактивні елементи для ефективної навігації і дослідження RDF графів. Основне призначення Rhizomer полягає у демократизації доступу до семантичних даних, надаючи інструмент, що дозволяє ефективно досліджувати та використовувати потенціал RDF графів знань без необхідності глибоких технічних знань про RDF, SPARQL або структури даних. Таким чином, Rhizomer сприяє розширенню кола користувачів, включаючи нефахівців у Semantic Web, популяризації технологій Semantic Web, здатних використовувати семантичні дані для аналізу, відкриття знань та прийняття рішень. В рамках даної роботи фокус зроблено на розширенні функціональності розділу Rhizomer шляхом інтеграції LLM для автоматичної генерації SPARQL-запитів з природномовних промптів.

2 ПРОЕКТУВАННЯ LLM-ІНТЕГРОВАНОГО RHIZOMER

Сучасні підходи, що інтегрують семантичний аналіз та нейронні мережі, досягли значного прогресу, проте ефективна обробка запитів, яка вимагає складної логіки все ще залишається проблематичною. Системи неявних відношень, часових обмежень, просторових міркувань, або комбінації декількох умов фільтрації недостатньо стійкі до різноманітності природномовних формулювань, зокрема, синонімії, перифразування, та діалектних відмінностей, що призводить до зниження точності при обробці запитів від широкого кола користувачів.

2.1 Актуальність та постановка задачі

У контексті зростання обсягів семантичних даних, зокрема інформації про наукові публікації студентів, система Rhizomer, як платформа для дослідження Semantic Web, надає потужні інструменти для навігації та аналізу RDF даних, проте її існуючий інтерфейс, особливо розділ Zoom&Filte, не повною мірою відповідає потребам широкого кола користувачів, зокрема тих, хто не володіє спеціалізованими знаннями в галузі семантичних технологій та мови SPARQL, тому актуалізується проблема забезпечення ефективного та інтуїтивно зрозумілого доступу до цих знань.

Необхідність спрощення доступу до даних наукових публікацій студентів зумовлена їхньою значною цінністю для освітнього та науково-дослідницького процесу. Ефективний пошук та аналіз цих даних може сприяти виявленню актуальних дослідницьких напрямків, пошуку потенційних наукових керівників та співробітників, моніторингу наукової активності студентів та підтримці прийняття управлінських рішень в освітній сфері. Однак, складність існуючих механізмів пошуку в Rhizomer обмежує використання цього цінного ресурсу, перешкоджаючи повному

розкриттю його потенціалу. Існуючі механізми пошуку, такі як фасетна фільтрація, вимагають від користувача розуміння структури RDF даних та онтології запитів природною мовою, а відсутність підтримки створює значний бар'єр для широкого кола потенційних користувачів, включаючи студентів, викладачів та адміністративний персонал навчальних закладів.

Існуючий інтерфейс Rhizomer вимагає від користувачів знання SPARQL або використання обмежених фільтрів на основі властивостей RDF даних.

Метою є створення більш інтуїтивного та доступного інтерфейсу, що дозволить ширшому колу користувачів, включаючи студентів та викладачів, ефективно використовувати RDF дані про наукові публікації.

Задача полягає у розширенні функціональності системи Rhizomer, зокрема розділу Zoom&Filter, шляхом інтеграції підсистеми, що дозволяє користувачам здійснювати пошук RDF даних про наукові публікації студентів за допомогою запитів природною мовою.

Система має забезпечити інтуїтивний інтерфейс, який дозволяє користувачам, не знайомим із SPARQL, формулювати запити природною мовою (наприклад, українською чи англійською), такі як «Знайди публікації студентів ХНУРЕ за 2023 рік». Ці запити автоматично транслюються в SPARQL-запити для вилучення даних із семантичної бази RDF. Розширення передбачає використання моделей обробки природної мови (NLP), наприклад, RoBERTa або DeepSeek, для аналізу запитів, витягнення ключових сутностей (автор, рік, установа) і формування відповідних фільтрів. Система також має підтримувати запити на пошук, наприклад, за типом публікації чи ключовими словами. Інтеграція з Rhizomer забезпечить збереження принципів Шнейдермана, дозволяючи користувачам плавно переходити від загального огляду даних до детального аналізу відфільтрованих результатів.

Вирішення цієї задачі дозволить зменшити бар'єр входу для користувачів, спростити процес пошуку інформації, підвищити

ефективність використання RDF даних про наукові публікації студентів, розширити коло користувачів здатних використовувати Rhizomer для аналізу та отримання цінних знань з семантичних даних. Інтеграція технологій обробки природної мови, зокрема великих мовних моделей (LLM), у системі Rhizomer розглядається як перспективний шлях до демократизації доступу до семантичних даних та розкриття їхнього потенціалу для різноманітних застосувань в освітній та науково-дослідницькій сферах.

2.2 Архітектура LLM-інтегрованого Rhizomer

Інтеграція технологій обробки природної мови, зокрема потенціалу великих мовних моделей (LLM) для перетворення природномовних запитів у формалізовані запити (NL-to-SPARQL), в існуючу архітектуру системи Rhizomer, яка ефективно працює з RDF даними, розглядається як надзвичайно перспективний та стратегічно важливий шлях. Цей підхід не лише сприяє радикальній демократизації доступу до складних семантичних даних, що зберігаються у вигляді графів знань [15], долаючи бар'єри, пов'язані з необхідністю володіння мовою запитів SPARQL та розумінням специфічної структури RDF, але й дозволяє повною мірою розкрити їхній прихований аналітичний та інформаційний потенціал, розкрити широкі можливості для різноманітних інноваційних застосувань в освітній та науково-дослідницькій сферах, завдяки інтуїтивно зрозумілим запитам, спрощенням пошуку релевантних наукових публікацій студентів, аналізу тенденцій у дослідницьких напрямках між дослідниками та темами, виявленню взаємозв'язків інтелектуальних систем на основі семантичних знань та роботі з науковою інформацією для всіх учасників освітнього та наукового процесу.

Запропонована архітектура спрямована на розширення функціональних можливостей системи Rhizomer інтеграцією

спеціалізованою LLM-орієнтованою підсистемою, а саме розділу Zoom&Filter, шляхом обробки природномовних запитів та їх перетворення у SPARQL. Ця архітектура є модульною, що дозволяє чітко розмежувати відповідальність компонентів та системи. Вона базується на використанні існуючих компонентів Rhizomer (RhizomerEye ta RhizomerAPI) та трійкового сховища Jena-Fuseki [14], доповнених новим модулем, який сприяє масштабованості та зручності обслуговування LLM-сервер (рисунок 2.1).

Основні компоненти запропонованої архітектури.

Роль RhizomerEye (Frontend) полягає у взаємодії з системою. У контексті даної роботи, RhizomerEye інтерфейс користувача було модифіковано шляхом додавання нового елементу інтерфейсу текстового поля природномовних запитів, зокрема, призначеного для введення користувачем (наприклад, англійською мовою). Відповідає за безпосередню візуалізацію семантичних вебданих з кінцевим користувачем, функціонує у стандартному веббраузері та зазвичай імплементується з використанням сучасних frontend-технологій та фреймворків, таких як JavaScript, TypeScript и Angular.

Оригінальна архітектура Rhizomer. Компонент RhizomerEye.

Компонент RhizomerEye відповідає за візуалізацію даних, графічне представлення RDF даних, отриманих від серверної частини (RhizomerAPI). Це включає відображення графів між класами (у розділі Overview), списків ресурсів, результатів фасетної фільтрації (у розділі Zoom&Filter), а також детальних описів окремих ресурсів (у розділі Details-on-demand). RhizomerEye обробляє дії користувача, такі як кліки по елементам візуалізації. Цей компонент відображає RDF-дані у вигляді інтерактивних графів або таблиць, дозволяючи користувачам бачити загальну структуру знань, наприклад, зв’язки між авторами, публікаціями та установами.

RhizomerEye формує запити до Backend.

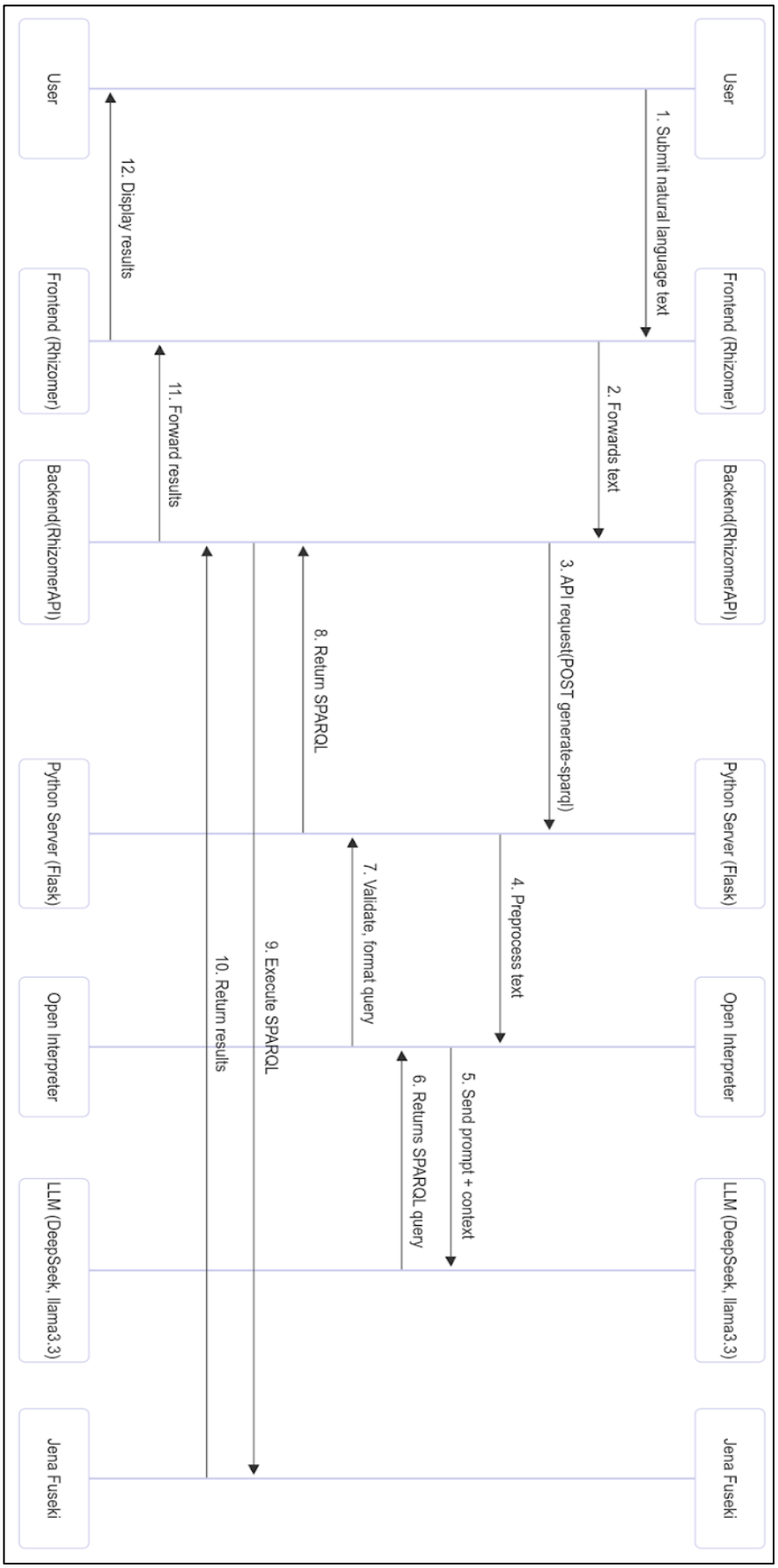

Рисунок 2.1 – Діаграма послідовності

RhizomerEye формує запити до Backend на основі дій користувача, які можуть ініціювати генерацію SPARQL-запитів на стороні RhizomerEye, відправляє їх до RhizomerAPI для обробки даних та отримання оновленої інформації.

Після отримання оброблених даних від RhizomerAPI, RhizomerEye відображає та оновлює візуалізацію.

2.3 Інтеграція LLM-орієнтованої системи NL-to-SPARQL з Open Interpreter

В роботі була запропонована інтеграція LLM-орієнтованої системи NL-to-SPARQL з Open Interpreter, та розширена підтримкою природномовного пошуку.

2.3.1 Модифікація RhizomerEye

Ключові зміни та можливості RhizomerEye в архітектурі з інтеграцією LLM:

– додавання інтерфейсу для природномовного запиту. В рамках розділу «Масштабування та фільтрація (Zoom&Filter)» в RhizomerEye був модифікован новий елемент інтерфейсу користувача, спеціально призначений для текстового поля (поле ввода) введення природномовних запитів користувачем (наприклад, англійською мовою);

– збір та передача природномовного запиту: RhizomerEye буде відповідати за збір тексту природномовного запиту, введеного користувачем у нове текстове поле при ініціації пошуку (наприклад, натисканням кнопки «Пошук» або клавіші «Enter»), сформує відповідний HTTP-запит (наприклад, POST-запит) та передасть природномовний текст запиту нового API ендпоінта на стороні RhizomerAPI, який призначений для прийому та обробки NL-to-SPARQL запитів;

– отримання та відображення результатів: після того, як RhizomerAPI обробить природномовний запит (залучивши LLM-Server та Jena-Fuseki) та отримає результати виконання згенерованого SPARQL запиту (наприклад, список відповідних наукових публікацій студентів), RhizomerEye від RhizomerAPI, далі Rhizomer Eye буде відповідати за візуалізацію цих результатів в інтерфейсі користувача, адаптуючи їх для представлення у зрозумілому форматі , який відповідає загальному стилю Rhizomer (наприклад, відображення списку знайдених ресурсів у основній області Zoom&Filter або представлення детальної інформації у розділі Details-on-demand при виборі конкретного результату). Таким чином, у запропонованій архітектурі RhizomerEye виступає як точка входу для природномовних запитів користувачів, відіграє ключову роль у зборі запиту та його ініціації до серверної частини, а також відповідає за фінальну візуалізацію результатів пошуку необхідних для реалізації природномовного інтерфейсу та досягнення мети даної кваліфікаційної роботи щодо спрощення доступу до семантичних даних, отриманих через складний процес NL-to-SPARQL перетворення та виконання SPARQL запиту до RDF даних.

2.3.2 Модифікація RhizomerAPI (Backend)

Існуючий бэкенд Rhizomer було модифіковано для обробки природномовних запитів та інтеграції с LLM-системою. RhizomerAPI (Бэкенд) в архітектурі системи Rhizomer виступає як серверний компонент - посередник та координатор між клієнтською частиною (RhizomerEye) та сховищем даних (Jena-Fuseki).

В оригінальній архітектурі Rhizomer, RhizomerAPI виконує наступні ключові завдання:

– прийом запитів від RhizomerEye;

– генерація та виконання SPARQL запитів;

– обробка результатів запитів;

– повернення даних до RhizomerEye.

Ключові зміни та функціональність RhizomerAPI в LLM-інтегрованій архітектурі:

– додавання нового API ендпоінта для природномовних запитів, він потрібен для прийому запитів та слугуватиме точкою входу для NL-to-SPARQL функціональності;

– передача природномовних запитів до LLM-Server: отримавши природномовний запит на новому ендпоінті, RhizomerAPI не буде самостійно генерувати SPARQL запит, а буде пересилати (форвардувати) цей природномовний запит до окремого LLM сервера, який відповідає за основну логіку перетворення NL-to-SPARQL за допомогою великої мовної моделі;

– прийом згенерованого SPARQL запиту від LLM-Server: після того, як LLM-Server обробить природномовний запит та згенерує відповідний SPARQL запит, він поверне цей SPARQL запит назад до RhizomerAPI;

– виконання згенерованого SPARQL запиту: отримавши SPARQL запит від LLM-Server, RhizomerAPI буде відповідати за нього. Цей процес включає виконання до Jena-Fuseki Server формування запиту до SPARQL ендпоінта Jena-Fuseki с використанням стандартних протоколів в взаємодії зі SPARQL серверами [14];

– обробка результатів виконання SPARQL: результати, повернуті Jena-Fuseki після виконання згенерованого SPARQL запиту, будуть прийняті та оброблені RhizomerAPI та структуровані у вигляд, зручний для візуалізації в RhizomerEye;

– повернення результатів до RhizomerEye: оброблені результати виконання SPARQL запиту передаються назад до RhizomerEye;

– обробка помилок та валідація (опційно): можуть виникати помилки в процесі взаємодії з LLM-Server (наприклад, помилки API LLM, rate limits, помилки генерації невалідного SPARQL) або з Jena-Fuseki (наприклад,

помилки виконання SPARQL запиту). Також може бути реалізована базова обробка помилок валідації на рівні RhizomerAPI перед відправкою запитів до LLM-Server або Jena-Fuseki.

У запропонованій LLM-інтегрованій архітектурі, RhizomerAPI відіграє центральну роль координатора потока обробки природномовних запитів та виконання згенерованих SPARQL запитів між існуючою frontend частиною Rhizomer та новою LLM-орієнтованою підсистемою NL-to-SPARQL, а також взаємодію з базовим сховищем RDF даних. Розширення функціональності RhizomerAPI для прийому природномовних запитів та координації їхньої обробки через зовнішній LLM-Server є критично важливим досягненням для реалізації механізму природномовного пошуку.

2.3.3 LLM-Server

Підсистема NL-to-SPARQL грає роль LLM-Server. Цей компонент в рамках даної кваліфікаційної роботи є новою розробкою, яка реалізує перетворення природної мови в SPARQL на основі великої мовної моделі. Новий компонент, розроблений на Python з використанням Flask, відповідає за:

– отримання природномовних запитів від RhizomerAPI;

– перетворення запитів у SPARQL за допомогою LLM;

– валідацію SPARQL запитів;

– виконання SPARQL запитів до Jena-Fuseki;

– повернення результатів до RhizomerAPI.

Функціональність LLM-Server в інтегрованій системі охоплює наступні етапи:

– прийом природномовного запиту: LLM-Server, що надходить від RhizomerAPI через визначений API ендпоінт (наприклад, HTTP POST запит) приймає природномовний запит користувача. Цей запит є текстом, сформульованим користувачем природною мовою (наприклад,

англійською), що виражає його інформаційну потребу щодо RDF даних наукових публікацій студентів;

– LLM Interaction. Отримавши природномовний запит, LLM-Server ініціює взаємодію з зовнішнім API LLM та формує спеціальний промпт (prompt) для LLM. Промпт включає сам природномовний запит користувача та, що критично важливо, може містити додатковий контекст;

– генерація SPARQL-запиту (SPARQL Query Generation). LLM, отримавши промпт по API, обробляє його та генерує потенційний SPARQL запит як текстовий вивід. LLM використовує свої попередньо навчені знання про мову, логіку, програмування та структуру даних, а також наданий контекст, для виконання задачі перетворення природної мови в формалізований запит до RDF даних. Результат цього етапу – це рядок, що містить згенерований LLM текст, який за задумом, є SPARQL запитом;

– валідація згенерованого SPARQL. LLM-Server приймає код, згенерований LLM, та здійснює його перевірку синтаксично коректним SPARQL запитом. Наприклад, ця валідація виконується за допомогою функціоналу синтаксичного парсингу Apache Jena ARQ або RDFlib. Якщо згенерований текст не є синтаксично валідним SPARQL, LLM-Server може викликати спеціалізовані бібліотеки або методи RDFlib, спробувати його скоригувати, або згенерувати повідомлення про помилку та повернути його до RhizomerAPI;

– виконання згенерованого SPARQL запиту. Якщо згенерований SPARQL запит є синтаксично валідним, LLM-Server ініціює його виконання до Jena-Fuseki Server, формування відповідного запиту до SPARQL ендпоінта Jena-Fuseki;

– прийом та обробка результатів виконання запиту. Ці результати зазвичай надходять у стандартних форматах, таких як JSON, але LLM-Server може здійснювати попередню обробку або форматування цих результатів, якщо це необхідно для подальшої передачі до RhizomerAPI, наприклад, для стандартизації формату або вилучення ключових полів;

– на завершальному етапі LLM-Server, що містить результати виконання SPARQL запиту, формує відповідь та повертає її назад до RhizomerAPI.

Таким чином, у запропонованій архітектурі, LLM-Server є центральним інтелектуальним вузлом, який здійснює комплексний процес перетворення природномовного запиту в результати, доступні для візуалізації в RhizomerEye. Його функціональність охоплює підготовку результатів, взаємодію з LLM для генерації коду, валідацію цього коду, виконання запиту до сховища RDF даних, реалізуючи тим самим основну ідею кваліфікаційної роботи щодо природномовного доступу до семантичних даних.

2.3.4 Сервер Jena-Fuseki (RDF Data Storage)

Існуючий сервер Jena-Fuseki виступає як основне сховище RDF даних про наукові публікації студентів.

Розмежування функціоналу між компонентами (frontend, backend, LLM обробка, сховище даних) забезпечує гнучкість, масштабованість та можливість незалежного розвитку кожного модуля, дає можливість ефективно реалізувати функціональність природномовного доступу до RDF даних. Запропонована модульна архітектура, що інтегрує нову LLM-орієнтовану підсистему в існуючу структуру Rhizomer, дозволяє використовувати переваги існуючої та перевіреної платформи Rhizomer, додаючи інноваційну функціональність обробки природної мови, що є ключовим для досягнення мети даної кваліфікаційної роботи.

2.4 Деталі реалізації підсистеми NL-to-SPARQL

Підсистема NL-to-SPARQL є ключовим компонентом розробленого рішення. Цей модуль відповідає за перетворення природномовних запитів

користувачів, що надходять від RhizomerAPI, у синтаксично валідні та семантично коректні SPARQL запити, які потім виконуються до Jena-Fuseki для видобування релевантних даних про наукові публікації студентів. Деталі реалізації охоплюють вибір та інтеграцію великої мовної моделі, стратегію формування промптів, а також механізми валідації та виконання згенерованих SPARQL запитів.

2.4.1 Вибір та інтеграція великої мовної моделі (LLM Selection and Integration)

На основі аналізу сучасних тенденцій та можливостей, а також враховуючи доступ до LLM через її програмний інтерфейс (API), для реалізації підсистеми було обрано практичні аспекти доступності, вартості та продуктивності Groq API, Llama 3 (через API провайдера), DeepSeek API. Обґрунтуванням вибору слугує відкритість моделі та можливість локального розгортання, також висока якість генерації коду.

Враховуючи специфіку задачі NL-to-SPARQL, яка вимагає точність, продуктивність та загальну ефективність глибокого розуміння природної мови, процес вибору LLM базувався на комплексному аналізі ряду критеріїв:

– ключовим критерієм є здатність LLM генерувати синтаксично валідні та семантично коректні SPARQL запити;

– ефективність роботи з контекстом (Prompt Engineering). Оскільки в рамках даної роботи, важливою є здатність LLM ефективно використовувати інформацію надану в промпті, включаючи інструкції, контекст предметної області (опис онтології) та приклади (Few-Shot Learning), для покращення якості генерації SPARQL, основним підходом є оперативный инжиниринг;

– продуктивність (швидкість, латентність). Для інтерактивної системи, такої як Rhizomer, є критично важливою для забезпечення

належного користувацького досвіду. Висока латентність може призвести до затримок у відображенні результатів пошуку, що негативно вплине на зручність використання системи;

– доступність та стабільність API. Використання LLM через API вимагає надійності та стабільності сервісу, що надається провайдером, важливими факторами є доступність API, наявність чіткої документації, та мінімізації ризиків, пов'язаних з обмеженнями (лимитами швидкості) або перебоями в роботі сервісу;

– бюджетні обмеження проекту можуть впливати на вибір LLM, оскільки різні провайдери мають різну політику ціноутворення за використання своїх API;

– англомовні промпти. Оскільки природномовні запити користувачів у даній системі формулюються англійською мовою (згідно з уточненими вимогами), LLM повинна мати високу якість обробки запитів саме англійською мовою.

На основі аналізу доступних LLM, їхніх характеристик та відповідностей вищезазначених критеріїв, для реалізації підсистеми NL-to-SPARQL були обрані Groq API за високу швидкість інференсу, та Llama 3 за потужну відкриту модель.

Обґрунтований вибір, здійснений на основі критеріїв якості генерації SPARQL, є ключовим етапом реалізації підсистеми NL-to-SPARQL. LiteLLM – це бібліотека Python та проксі-сервер (LLM Gateway) для уніфікованого доступу до 100+ API великих мовних моделей (OpenAI, Anthropic, Azure, Grok тощо) у форматі OpenAI. Дозволяє викликати моделі через єдиний інтерфейс, підтримуючи потокову передачу, балансування навантаження, відстеження витрат і логування. Для інтеграції з RhizomerEye, LiteLLM може обробляти запити природною мовою, транслюючи їх у SPARQL через LLM, і повертати лише вибрані JSON-поля (наприклад, автор, назва), оптимізуючи дані для візуалізації. Використання бібліотеки Лителльм для API LLM забезпечує необхідну

гнучкість та уніфікацію взаємодії, що є важливим для подальшого тестування, оцінки та потенційного масштабування системи. Цей підхід дозволяє ефективно використовувати потужність сучасних LLM для вирішення задачі перетворення природномовних запитів у SPARQL в контексті системи Rhizomer.

2.4.2 Стратегія Prompt Engineering для генерації SPARQL

Оскільки в рамках даної роботи основним підходом до використання LLM є оперативний інжиніринг, ключовим етапом реалізації буде розробка та оптимізація промптів (запитів-підказок), які подаються на вхід LLM. Мета промпт-інжинірингу – максимально чітко направити генерацію таким чином, щоб отримати синтаксично валідний та семантично коректний SPARQL запит для RDF даних про наукові публікації студентів, та інструктувати LLM щодо задачі надати їй необхідний контекст.

Промпт, що подається на вхід LLM, конструюється з наступних основних компонентів:

– чітка інструкція для LLM генерувати SPARQL запити, задача полягає у перетворенні природномовного запиту користувача у SPARQL запит, наприклад, «Перетворіть наступне питання природною мовою на запит SPARQL», «Згенеруйте запит SPARQL на основі запиту користувача»;

– надання LLM контексту предметної області (наукові публікації студентів) та структури RDF даних (префікси, класи, властивості з онтології). LLM необхідно надати інформацію про структуру RDF даних, з якими вона працює, щоб вона могла правильно зіставити терміни природномовного запиту з елементами RDF схеми (онтології). До промпту включається явне перелічення використовуваних URI префіксів (PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX thesis: <http://example.org/thesis/> PREFIX rdfs: <http://www.w3.org/2000/01/rdf-

schema#>), короткий опис основних класів онтології з поясненням їхнього значення в контексті наукових публікацій студентів, а також перелічення та короткий опис основних властивостей, що пов'язують класи із зазначенням їхніх доменів та діапазонів (якщо відомо). Це допомагає LLM зрозуміти як ресурси пов'язані між собою та які властивості можна використовувати;

– природномовний запит, введений користувачем у RhizomerEye, підлягає перетворенню у SPARQL;

– надання LLM прикладів природномовних запитів та відповідних SPARQL запитів для навчання. Ці приклади слугують для демонстрації LLM бажаного стилю генерації SPARQL, використання специфічних термінів онтології, та структури запитів.

Оптимізація промптів є ітеративним процесом, що включає тестування різних варіантів промптів, аналіз згенерованих SPARQL запитів та їх коригування для покращення результатів.

2.4.3 Валідація та виконання згенерованого SPARQL

LLM можуть генерувати синтаксично невалідний код, тому обов'язковим етапом є синтаксична валідація згенерованого LLM тексту перед його виконанням. Це запобігає помилкам на стороні SPARQL ендпоінта Jena-Fuseki та забезпечує надійність системи. Валідація здійснюється за домогою функціоналу парсингу SPARQLRDFlib [], що надається стандартними RDF бібліотеками, або завдяки взаємодії з Jena здійснюється на рівні API, що дозволяє валідацію. Якщо згенерований текст не проходить синтаксичну валідацію, система функціонал парсингу Apache Jena ARQ обробляє цю помилку, наприклад, повертає відповідне повідомлення користувачеві через RhizomerAPI та не намагається виконати невалідний запит.

Може виконуватися семантична валідація (не обов'язкова, залежить від складності), перевіряється семантична коректність запиту. Наприклад,

шляхом порівняння результатів з очікуваними результатами для набору тестових запитів.

Після успішної синтаксичної валідації, LLM-Server ініціює виконання згенерованого SPARQL запиту. Запит надсилається до Jena-Fuseki, і LLM-Server функціонал SPARQL Querying бібліотеки RDFlib очікує відповіді, що містить результати виконання, наприклад, у форматі JSON. Далі відбувається обробка результатів виконання запиту. Отримавши результати виконання SPARQL запиту від Jena-Fuseki, LLM-Server обробляє ці дані, та виконує мінімальне форматування результатів, якщо це необхідно для уніфікації відповіді до RhizomerAPI, перш ніж повернути їх як фінальну відповідь на початковий запит від RhizomerAPI. Зазвичай результати надходять у стандартизованому форматі, наприклад, SPARQL Query Results JSON Format, який легко парсити.

3 ІНТЕГРАЦІЯ LLM З КОМПОНЕНТАМИ RHIZOMER

Процес інтеграції LLM-підсистеми передбачає модифікацію ключових компонентів Rhizomer. Нижче наведено опис основних частин системи та їх реалізацію.

3.1 Search-Input Component

Frontend (RhizomerEye). Новий компонент Search-Input Component. Компонент пошуку (search-input.component.ts). який відповідає за введення природномовних запитів користувача. Цей компонент є ключовим елементом інтерфейсу та забезпечує зручну інтеракцію з системою. Програмний код HTML-шаблону компонента наведений у лістингу 3.1, який демонструє структуру форми введення та кнопку запуску пошуку.

Лістинг 3.1 – HTML-Код нового компоненту

```
<div class="search-container">
    <div class="input-group">
      <input id="search"
        name="search"
        type="text"
        class="pl-3 form-control form-control-sm"
        [(ngModel)]="searchQuery"
        (keyup.enter)="onSearch()"
        placeholder="Search all..."  />
        <div class="input-group-append">
          <button class="btn btn-sm
 btn-secondary" type="button"
(click)="onSearch()">
            <span class="fa fa-search"></span>
        </button>
    </div>
</div>
```

Дозволяє користувачам вводити запити типу «Знайти дисертації Гриньової». Обробка подій: При натисканні Enter або кнопки пошуку викликається метод onSearch(), який передає запит до PromptHandlerService. Логіка обробки цих подій реалізована у методі onSearch(), код якого наведено в лістингу 3.2.

Лістинг 3.2 – Додавання текстового поля

```
onSearch(): void {
  this.promptHandler.processPrompt(trimmedQuery,
this.datasetId);
}
```

Представлено фрагмент TypeScript-коду компонента, в якому реалізована передача введеного запиту до відповідного сервісу (PromptHandlerService). Цей сервіс відповідає за подальше опрацювання запиту та ініціацію генерації SPARQL-запиту через LLM. На рисунку 3.1 представлен новий компонент для зв’язку з моделями.

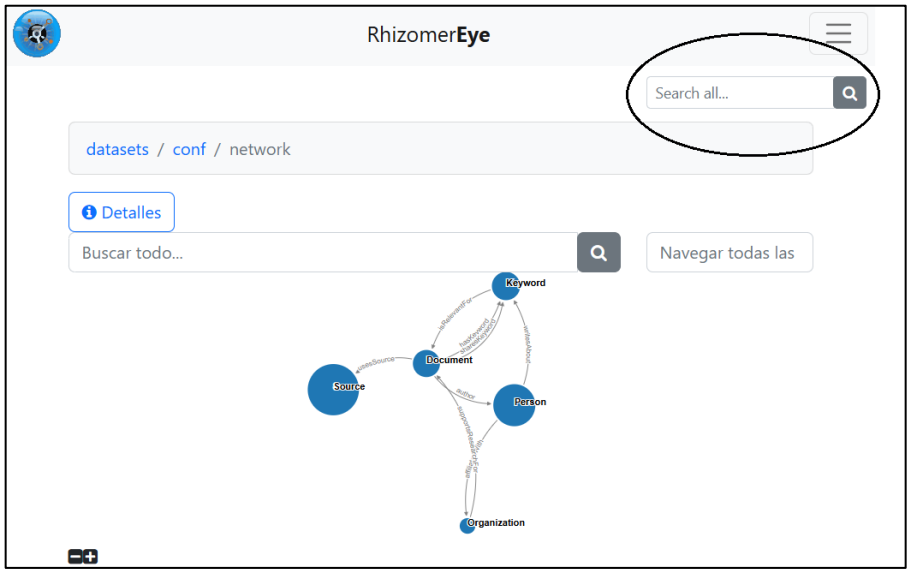

Рисунок 3.1 – Вигляд інтерфейсу застосунка з новим компонентом

Таким чином, Search-Input Component виступає початковою точкою взаємодії користувача з системою, забезпечуючи зручний механізм введення запитів у природній мові.

3.2 REST ендпоінт

Backend (RhizomerAPI). На серверній стороні реалізовано новий REST ендпоінт у класі ClassController.java, який відповідає за прийом запитів з фронтенду, взаємодію з LLM-сервером (Flask) та виконання згенерованих запитів через Jena-Fuseki. У лістингу 3.3 наведено оголошення нового ендпоінту, що обробляє запити за параметрами datasetId, classCurie та prompt.

Лістинг 3.3 – Новий REST ендпоінт у ClassController

```
@GetMapping("/datasets/{datasetId}/classes/{classCurie}/instances"
public ResponseEntity<StreamingResponseBody> retrieveClassFacetedInstances2(...)
```

Обробляє параметр prompt, декодує його та передає до LLM-Server (Flask). Для передачі запиту до LLM-Server використовується RestTemplate, приклад використання якого подано в лістингу 3.4.

Лістинг 3.4 – Взаємодія з LLM-Server

```
RestTemplate restTemplate = new RestTemplate();
ResponseEntity<String> flaskResponse = restTemplate.postForEntity("http://localhost:5050/api/askGroq", entity, String.class);
```

Java-сервер надсилає HTTP POST-запит до Flask-сервера для генерації SPARQL-запиту. Отриманий результат у форматі тексту містить SPARQL, який буде виконано через Jena-Fuseki:

```
StreamingResponseBody stream = analizeDataset.retrieveClassInstancesStringFromLLM(...);
```

Для більшого розуміння потоку данних у застосунку Rhizomer представлено рисунок 3.2

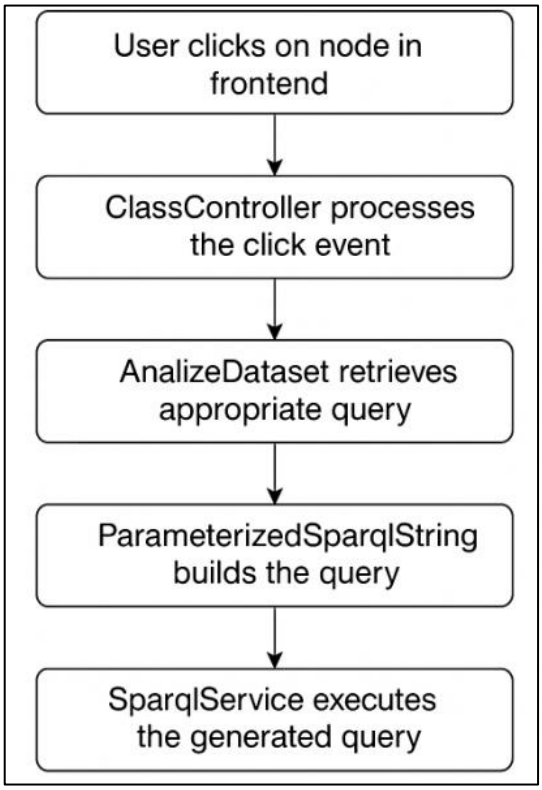

Рисунок 3.2 – Схема потоку даних у Rhizomer

Реалізація REST ендпоінту у RhizomerAPI забезпечує критичну зв'язувальну ланку між інтерфейсом користувача та сервером генерації SPARQL-запитів. Цей компонент не лише приймає природномовні запити, але й координує взаємодію з LLM-сервером та системою виконання запитів, забезпечуючи повний цикл обробки – від введення запиту до отримання структурованих результатів.

3.3 LLM-Server (Flask): генерація SPARQL

На стороні Flask реалізовано сервер LLM-Server, який приймає POST-запити, обробляє їх за допомогою великої мовної моделі (наприклад, DeepSeek або Llama-3.3) і повертає SPARQL-запит. Спочатку на рисунку 3.3 представлена робота open interpreter з терміналу, яка гарантує, що модель працює та оброблює запити.

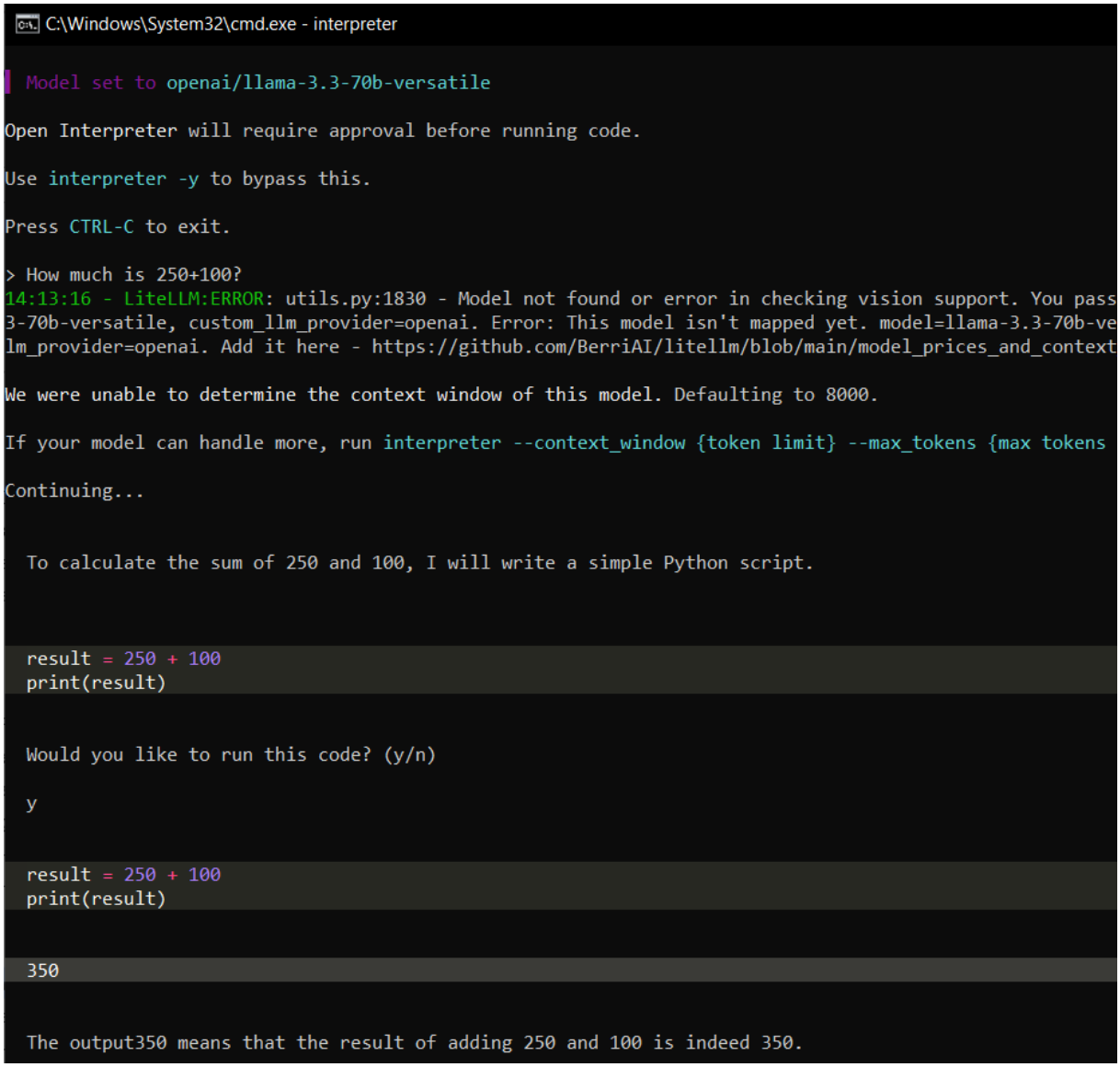

Рисунок 3.3 – Вигляд робочої моделі llama-3.3-70b-versatile

У лістингу 3.5 представлено Python-метод обробки POST-запиту.

Лістинг 3.5 – Обробка POST запитів

```
@app.route('/api/library', methods=['POST'])
def receive_post3i():
    data = request.json
    prompt = prompt_template.replace("{{TERM}}",
data["prompt"])
```

Сервер отримує JSON-запит, витягує з нього параметр prompt, і підставляє його в шаблон для подальшої генерації.

Подальша генерація виконується за допомогою Open Interpreter, як наведено у лістингу 3.6.

Лістинг 3.6 – Використання Open Interpreter

```
response = interpreter.chat(prompt)
sparql_query = extract_sparql_from_response(response)
```

Мовна модель генерує відповідь, з якої за допомогою регулярних виразів видобувається SPARQL-запит.

Шаблон запиту (general_library.txt) містить інструкції для LLM щодо генерації SPARQL, шаблон для генерації представлено в лістингу 3.7.

Лістинг 3.7 – Шаблон запиту

```
You are an AI assistant tasked with helping a user search
for products in an RDF file using SPARQL queries.
The user wants to find products whose name contains a
specific term and map these names to their respective URIs in
the RDF data.
Then, you need to fetch the products using these URIs with
rdflib and SPARQL to obtain the labels of the products that meet
the conditions.
Do not execute it, just give SPARQL query.
In result you must give in json format with a 'sparql' key
and only query as a value.
```

LLM-сервер на Flask виконує ключову роль у системі, перетворюючи природномовні запити на формалізовані SPARQL-запити за допомогою великих мовних моделей. Завдяки використанню шаблонів і Open Interpreter, сервер забезпечує гнучке та масштабоване рішення для генерації запитів без потреби у жорсткому доменному налаштуванні.

3.4 Виконання SPARQL (Jena-Fuseki)

Серверна частина системи після отримання SPARQL-запиту виконує його через Jena-Fuseki. Клас SPARQLService.java відповідає за формування і виконання запиту, що показано на лістингу 3.8.

Лістинг 3.8 – Побудова запиту

```
QueryExecutionHTTPBuilder qBuilder = QueryExecutionHTTPBuilder.create();
qBuilder.query(query).endpoint(endpoint.getQueryEndPoint().toString());
```

Результати запиту обробляються у формат JSON-LD, як показано в лістингу 3.9.

Лістинг 3.9 – Обробка результатів виконання запиту

```
ResultSet resultSet = (ResultSet) result;
String jsonLd = transformer.transformToJsonLd(query, resultSet);
out.write(jsonLd.getBytes());
```

Видно, як результат SPARQL-запиту трансформується для подальшої візуалізації у RhizomerEye.

Виконання SPARQL-запитів за допомогою Jena-Fuseki забезпечує ефективну обробку згенерованих запитів та трансформацію результатів у

формат JSON-LD. Це дозволяє інтегрувати відповіді у фронтенд-систему RhizomerEye, забезпечуючи зручну візуалізацію та подальший аналіз отриманих даних.

3.5 Валідація та обробка помилок

Синтаксична валідація SPARQL: у AnalyzeDataset.java перевіряється коректність запиту перед виконанням, наприклад, `Query query = QueryFactory.create(escapedQueryString);`

У ClassController.java реалізовано перехоплення винятків (наприклад, 429 Rate Limit), приклад чого наведено у лістингу 3.10.

Лістинг 3.10 – Обробка помилок при взаємодії з LLM

```
catch (Exception e) {
    logger.error("Error: {}", e.getMessage());
    return
ResponseEntity.status(500).body("Помилка обробки запиту");
}
```

Механізми валідації та обробки помилок підвищують надійність системи, запобігаючи виконанню некоректних SPARQL-запитів і забезпечуючи стабільну роботу навіть у випадку збоїв або неправильного вводу користувача.

3.6 Тестування та результати

В цьому розділі описано методологію оцінки, використані набори даних та бенчмарки, отримані результати та їх аналіз.

3.6.1 Набір даних для тестування

Система надійно обробляє основні запити природною мовою, перетворюючи їх на виконувані запити SPARQL та відображаючи результати в інтерфейсі Rhizomer.

Приклад 1.

На природній мові дано системі запит: «Give all Thesis».

Результат. Генерує SELECT ?thesis WHERE { ?thesis a thesis:Document }, отримуючи всі екземпляри тези.

Користувацький досвід: Результати відображаються в розділі «Details», як і очікувалося (рисунок 3.4).

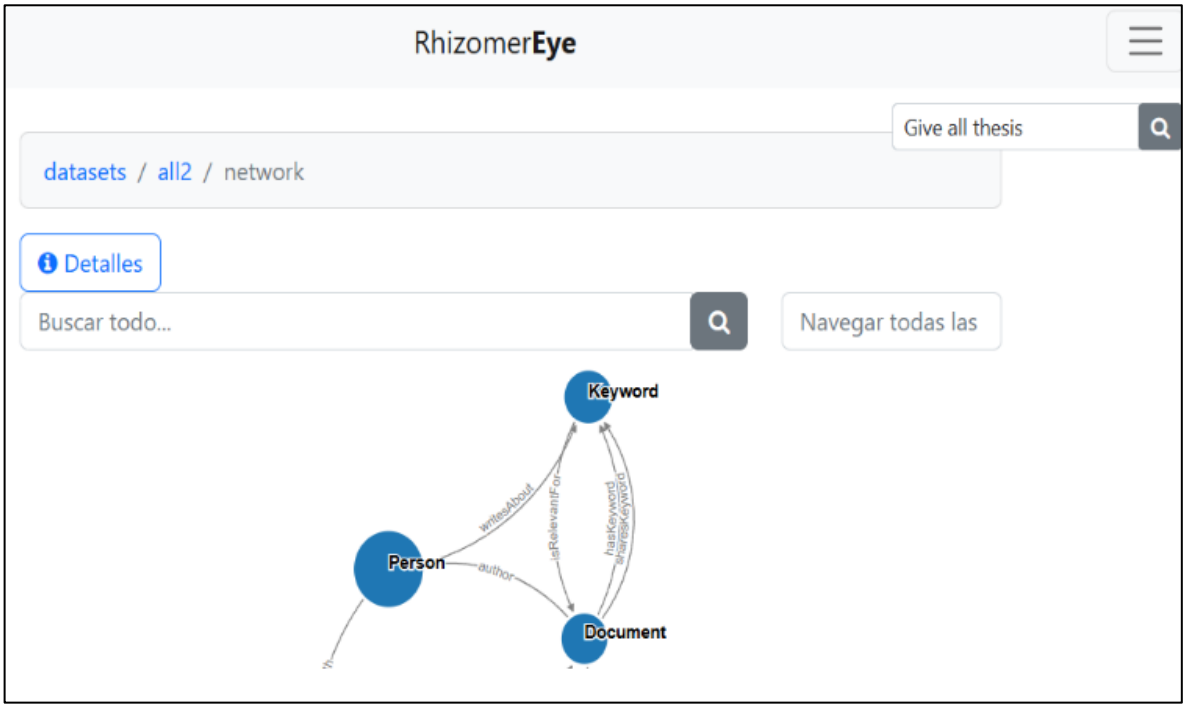

Рисунок 3.4 – Вигляд інтерфейсу застосунка з запитом для моделі

Представлений граф ілюструє онтологічну модель, де вузли типів Person, Document, Source, Keyword та Organization пов’язані між собою через чітко визначені семантичні властивості, які формують основу зв’язної структури RDF-даних:

– thesis:author пов’язує об'єкти класу Document з одним або кількома екземплярами класу Person, вказуючи, хто є автором дисертації;

– thesis:hasKeyword – пов’язує документ з ключовими поняттями (Keyword), що дозволяє автоматизувати тематичну класифікацію;

– thesis:affiliatedWith описує належність автора до певної Organization, наприклад університету;

– thesis:hasKeyword – пов’язує документ з ключовими поняттями (Keyword), що дозволяє автоматизувати тематичну класифікацію;

– thesis:usesSource – вказує на використані в дисертації джерела (Source), що є важливою складовою наукової достовірності;

– thesis:relatedTo – пов’язує Source із відповідними Keyword, уточнюючи контекст використаного матеріалу.

Далі можемо побачити результат, який дав LLM (рисунок 3.5).

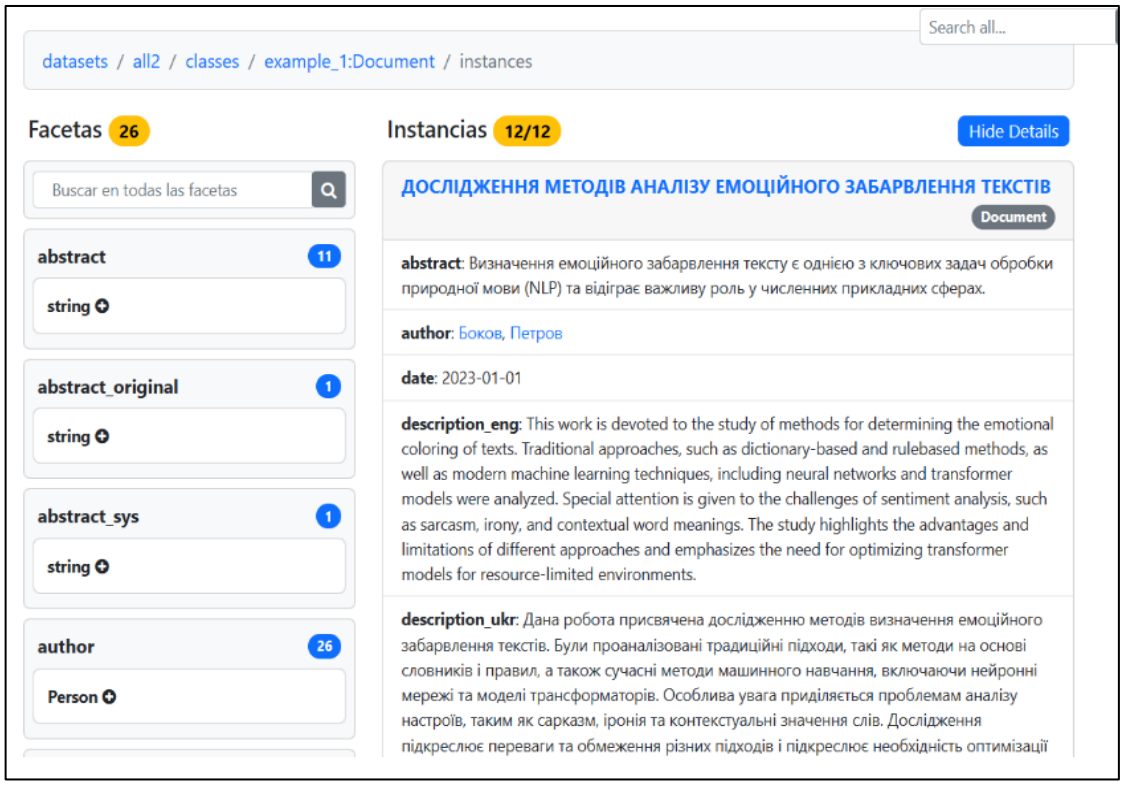

Рисунок 3.5 – Вигляд результату

Приклад 2.

«Give all thesis of Author Гриньова»

Результат. Генерує SELECT ?thesis ?title WHERE { ?thesis rdf:type thesis:Document . ?thesis thesis:author <http://example.org/thesis/Гриньова> . }

Користувацький досвід: Всі роботи автора Гриньова відображаються в таблиці з можливістю прокручування з гіперпосиланнями для подальшого дослідження (рисунок 3.6).

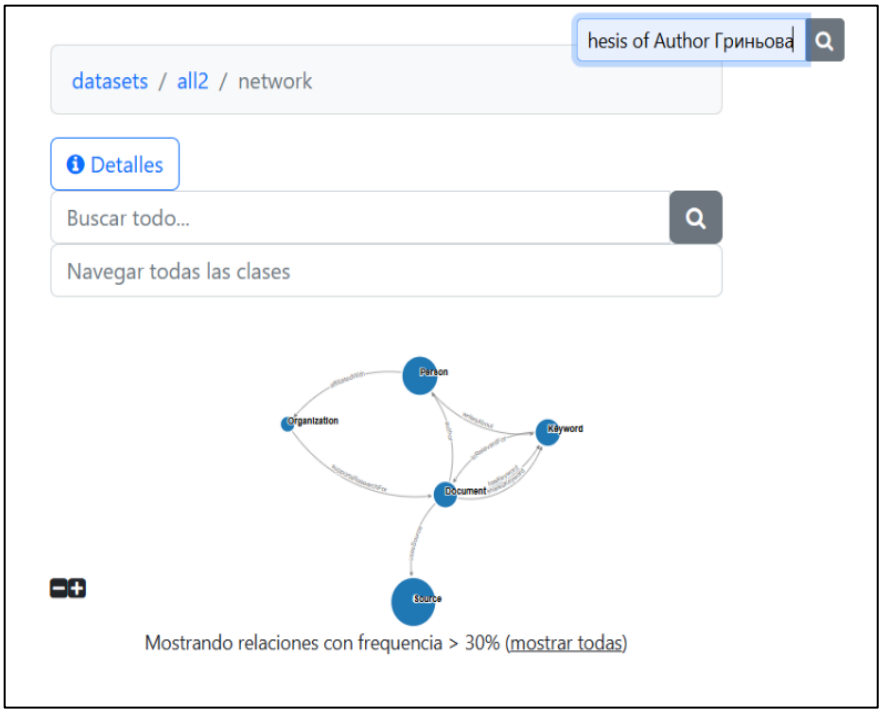

Рисунок 3.6 – Вигляд інтерфейсу застосунка з запитом для моделі

Далі можемо побачити результат який дав LLM на рисунку 3.7.

У цих випадках LLM успішно зіставляє природну мову з синтаксисом SPARQL, використовуючи префікси онтологій (наприклад, thesis:Document) та базову фільтрацію. Користувачі можуть інтуїтивно досліджувати набори даних без досвіду роботи зі SPARQL.

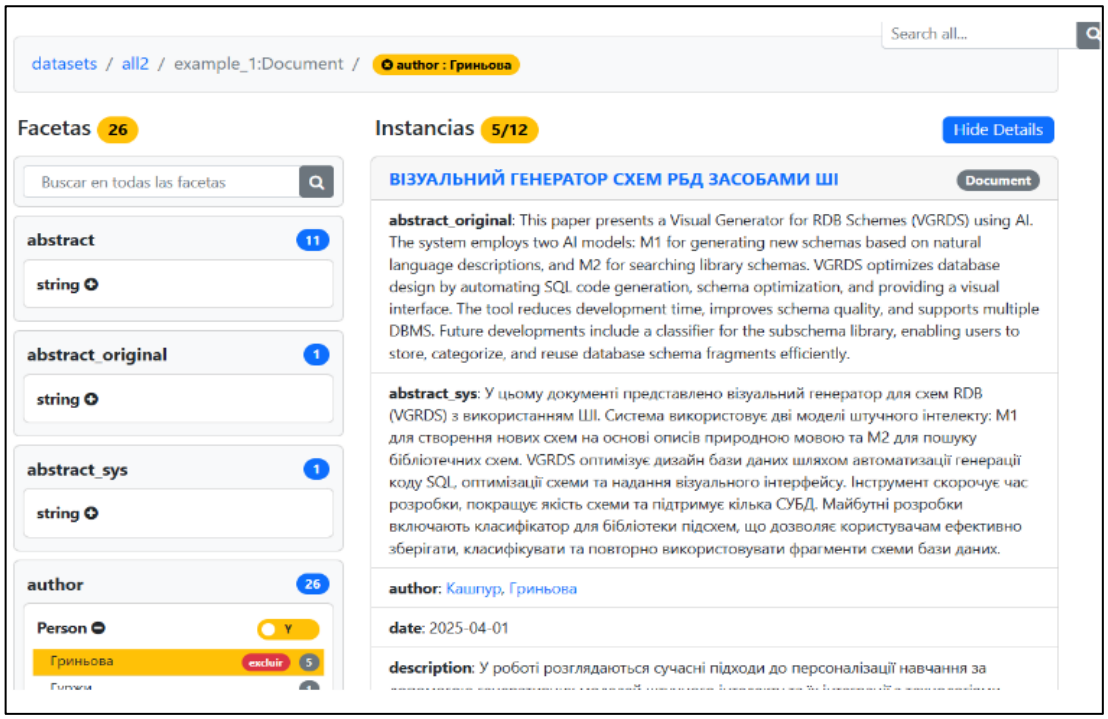

Рисунок 3.7 – Вигляд інтерфейсу застосунка з запитом для моделі

Результати тестування демонструють, що система ефективно трансформує природномовні запити у коректні SPARQL-конструкції, використовуючи релевантні онтологічні префікси та властивості. Інтерфейс Rhizomer забезпечує зручне візуальне представлення результатів, що дозволяє користувачам без знань SPARQL інтуїтивно взаємодіяти з RDF-даними. Це підтверджує життєздатність інтеграції LLM у сценарії пошуку та фільтрації семантичної інформації.

3.6.2 Проблеми зі складними запитами

Для запитів з кількома умовами або великими зв'язками система має труднощі зі створенням точного SPARQL.

Приклади проблемних запитів.

Приклад 3. «Give all thesis of authors from Харківський національний університет радіоелектроніки who wrote after 01.01.2020».

Проблема: LLM часто не враховує вкладений зв'язок між авторами та організаціями. Не вдається поєднати речення FILTER для дат та афіліацій та використовує неправильні предикати (наприклад, thesis:affiliation замість thesis:affiliatedWith).

Приклад 4. «Підрахувати авторів, які опублікували більше 5 дисертацій».

Проблема: LLM рідко правильно генерує запити агрегації (COUNT, GROUP BY), пропускаючи групування або використовуючи недійсний синтаксис.

У таких випадках згенерований SPARQL може бути синтаксично коректним, але семантично невирівняним, що призводить до порожніх або неправильних результатів. Користувачі не отримують результатів або отримують нерелевантні дані, що вимагає ручного уточнення запиту.

3.6.3 Обмеження токенів та практичні обмеження

Залежність системи від сторонніх API LLM створює вузькі місця.

Щоденні обмеження токенів, наприклад, сервіс LLM встановлює щоденне обмеження в 100 000 токенів. Вичерпання цього ліміту блокує подальші запити до скидання.

Під час оцінювання складні запити швидко споживали токени, обмежуючи обсяг експериментів.

Базові запити (наприклад, «Показати всі дисертації») використовують менше токенів, що робить їх стійкими в рамках обмежень швидкості.

3.7 Перспективи розвитку системи

Хоча система ефективно працює для простого семантичного дослідження, для її масштабування та продуктивного використання у реальних сценаріях необхідні суттєві покращення.

В першу чергу, доцільним є удосконалення підказок (prompt engineering) для забезпечення більш точного та керованого використання онтологічних термінів мовною моделлю. Наприклад, можна прямо зазначати: «Використовуйте thesis:affiliatedWith для позначення організацій, з якими пов’язані автори». Такі інструкції знижують рівень неоднозначності та покращують відповідність згенерованих запитів наявним RDF-схемам.

Також перспективним є поєднання LLM з механізмами пост-обробки на основі фіксованих правил. Це дозволяє автоматично виправляти типові помилки, які часто повторюються: наприклад, додавання відсутніх префіксів (thesis:), виправлення неправильних структур WHERE-блоків або коректне замикання фігурних дужок у SPARQL-запитах.

Крім того, важливо передбачити можливість інтеграції механізмів зворотного зв’язку: користувачі могли б коригувати згенеровані запити або результати, що дозволить створити цикл самонавчання для покращення точності у наступних ітераціях.

У перспективі система може бути доповнена інтерфейсом для ручного редагування SPARQL, підсвічуванням помилок, логом виконаних запитів та статистикою продуктивності моделей. Це відкриє шлях до її використання не лише студентами, а й дослідниками, аналітиками та розробниками у сфері семантичного вебу.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було успішно завершено розробку та здійснено інтеграцію спеціалізованої LLM-орієнтованої підсистеми, призначеної для ефективного перетворення природномовних запитів користувачів у формалізовані запити мовою SPARQL. Ця підсистема була безшовно інтегрована у функціональний простір існуючої системи Rhizomer, що дозволило реалізувати нову можливість здійснювати пошук RDF даних, які описують наукові публікації студентів, безпосередньо за допомогою запитів, сформульованих природною мовою. Таке досягнення суттєво спростило механізми доступу користувачів до семантичної інформації, що зберігається у вигляді графу знань, усунувши необхідність володіння мовою запитів SPARQL та покращивши загальний користувацький досвід взаємодії з даними наукових публікацій.

Розроблена система успішно демонструє принципову можливість та реалізовує функціонал природномовного доступу до RDF даних наукових публікацій студентів в системі Rhizomer. Інтеграція LLM спростила механізм пошуку порівняно з необхідністю володінням SPARQL або обмеженою фасетною фільтрацією. Підтвердженням досягнення основної мети роботи є отримані результати, які мають високу практичну значимість для потенційних користувачів (студентів, викладачів), роблячи семантичні дані більш доступними та корисними.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) Gupta, R., & Malik, S. K. (2022). A classification using RDFLIB and SPARQL on RDF dataset. *Journal of Information and Optimization Sciences*, *43*(1), 143–154. https://doi.org/10.1080/02522667.2022.2039461 (дата звернення: 29.04.2025).

2) Cyganiak, R.; Wood, D.; Lanthaler, M. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation, 25 February 2014. URL: https://www.w3.org/TR/rdf11-concepts/ (дата звернення: 29.04.2025).

3) Cardoso, J., & Pinto, A. M. (2015). The Web Ontology Language (OWL) and Its Applications. En *Encyclopedia of Information Science and Technology, Third Edition* (pp. 7662–7673). IGI Global. https://doi.org/10.4018/978-1-4666-5888-2.ch755 (дата звернення: 30.04.2025).

4) García R., López-Gil J.-M., Gil R. Rhizomer: Interactive semantic knowledge graphs exploration. *SoftwareX*. 2022. T. 20. C. 101235. URL: https://doi.org/10.1016/j.softx.2022.101235 (дата звернення: 29.04.2025).

5) Nottingham, M. (2019). Well-Known Uniform Resource Identifiers (URIs). RFC Editor. https://doi.org/10.17487/rfc8615 (дата звернення: 02.05.2025).

6) Tzitzikas, Y., Lantzaki, C., & Zeginis, D. (2012). Blank Node Matching and RDF/S Comparison Functions. En *The Semantic Web – ISWC 2012* (pp. 591–607). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-35176-1_37 (дата звернення: 05.05.2025).

7) Kim, K., Moon, B., & Kim, H.-J. (2013). R3F: RDF triple filtering method for efficient SPARQL query processing. *World Wide Web*, *18*(2), 317–357. https://doi.org/10.1007/s11280-013-0253-1 (дата звернення: 05.05.2025).

8) Klein, M. (2001). XML, RDF, and relatives. *IEEE Intelligent Systems*, *16*(2), 26–28. https://doi.org/10.1109/5254.920596 (дата звернення: 05.05.2025).

9) Transitive Credit and JSON-LD. (2015). *Journal of Open Research Software*, *3*. https://doi.org/10.5334/jors.by (дата звернення: 17.05.2025).

10) Gerding, D. N. (1987). Triples. *Journal of Infectious Diseases*, *155*(5), 1085. https://doi.org/10.1093/infdis/155.5.1085 (дата звернення: 17.05.2025).

11) Hayes-Roth, F. (1985). Rule-based systems. *Communications of the ACM*, *28*(9), 921–932. https://doi.org/10.1145/4284.4286 (дата звернення: 17.05.2025).

12) Yin, X., Gromann, D., & Rudolph, S. (2021). Neural machine translating from natural language to SPARQL. *Future Generation Computer Systems*, *117*, 510–519. https://doi.org/10.1016/j.future.2020.12.013 (дата звернення: 17.05.2025).

13) Yurchak, I. Y., Kychuk, O. O., Oksentyuk, V. M., & Khich, A. O. (2024). CAPABILITIES AND LIMITATIONS OF LARGE LANGUAGE MODELS. *Computer systems and network*, *6*(2), 267–280. https://doi.org/10.23939/csn2024.02.267 (дата звернення: 17.05.2025).

14) *Apache Jena Documentation*. Apache Software Foundation. URL: https://jena.apache.org/documentation/ (дата звернення: 20.05.2025).

15) В.І. Жеребкін, С.Г. Удовенко, Л.Е. Чала, О.Є. Гриньова. Аналіз складності та побудова концептуальних графів масових відкритих онлайн-курсів// Біоніка інтелекту. – 2023. – Вип. 1 (99). С.26-37. https://doi.org/10.30837/bi.2023.1(99).04 (дата звернення: 24.05.2025).