МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 50.00.00.000 ПЗ

Група ІПм-23-2

Мацевич Олег

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Мацевич Олег Ярославович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі, методи та алгоритми створення графа знань

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Мацевич О.Я.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Михайлюк Ірина Романівна, к.п.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Мацевичу Олегу Ярославовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Моделі, методи та алгоритми створення графа знань”

керівник проекту (роботи) Михайлюк Ірина Романівна, к.п.н., доцент

затверджені наказом закладу вищої освіти від “ 22 ” листопада 2024 р. № 781/7

2. Строк подання студентом проекту (роботи) 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови та функціонування інформаційних та програмних технологій побудови графів знань

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Дослідження предметної області вилучення даних для побудови графів знань
2. Проектування дизайну онтологій, створення архітектури, методології. Підготовка даних
3. Вивчення методології вилучення інформації. Побудова моделі
4. Реалізація моделей та методів для побудови графа знань

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Таксономія областей застосування графів знань (рис. 1.1)
2. Покращений процес пошуку (рис. 1.2)
3. Архітектура системи підтримки документації програмного забезпечення (рис. 1.3)
4. Приклад графа знань із двома представленнями (рис. 1.4)
5. Фрагмент семантичної мережі знань здорового глузду ConceptNet (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі магістерської роботи	15.09.2024	виконано
2	Аналіз концепцій та алгоритмів предметної області	29.09.2024	виконано
3	Дослідження предметної області вилучення даних для побудови графів знань	15.10.2024	виконано
4	Проектування дизайну онтологій, створення архітектури, методології. Підготовка даних	08.11.2024	виконано
5	Вивчення методології вилучення інформації. Побудова моделі	20.11.2024	виконано
6	Реалізація моделей та методів для побудови графа знань	01.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 78 с., 27 рис., 4 табл., 52 джерела.

Тема: Моделі, методи та алгоритми створення графа знань

Об’єкт дослідження: процес вилучення даних та побудови графів знань із використанням онтологій і графових баз даних.

Мета роботи: розробка моделі, методів та засобів побудови графу знань для ефективного вилучення, структурування та зберігання знань із неструктурованих текстових даних.

Предмет дослідження: моделі, методи та інструменти для вилучення, структурування та зберігання знань із текстових даних у вигляді графів знань.

Результати дослідження

В представленій роботі розроблені методи і підходи дозволили створити якісний граф знань, що відповідає вимогам сучасних інформаційних систем.

Висновок

Досліджено предметну область вилучення даних для побудови графів знань, розроблено методологію побудови онтологій, створено дизайн архітектури графу знань, реалізовано моделі та методи вилучення інформації.

ГРАФ ЗНАНЬ, ОНТОЛОГІЯ, ВИЛУЧЕННЯ ДАНИХ, ОБРОБКА ПРИРОДНОЇ МОВИ, ГРАФОВА БАЗА ДАНИХ, СТРУКТУРА ЗНАНЬ, ЗВ’ЯЗУВАННЯ СУТНОСТЕЙ, ВИЛУЧЕННЯ ВІДНОШЕНЬ

ABSTRACT

Master Thesis: 78 pp., 27 fig., 4 tab., 52 sources.

Thesis Subject: Models, methods and algorithms for creating a knowledge graph

Object of research: the process of data extraction and building knowledge graphs using ontology and graphical databases.

Purpose of work: development of models, methods and tools for building a knowledge graph for effective extraction, structuring and storage of knowledge from unstructured text data.

Subject of research: models, methods and tools for extracting, structuring and storing knowledge from text data in the form of knowledge graphs.

Research results

In the presented work, the methods and approaches developed allowed to create a high-quality knowledge graph that meets the requirements of modern information systems.

Conclusion

The subject area of data extraction for building knowledge graphs was studied, a methodology for building an ontology was developed, a knowledge graph architecture design was created, and models and methods for information extraction were implemented.

KNOWLEDGE GRAPH, ONTOLOGY, DATA EXTRACTION, NATURAL LANGUAGE PROCESSING, GRAPH DATABASE, KNOWLEDGE STRUCTURE, ENTITY ASSIGNMENT, RELATION EXTRACTION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

MDE - Model-Driven Engineering

DSL - Domain-Specific Language

KG - Knowledge Graph

RDF - Resource Description Framework

OWL - Web Ontology Language

LDA - Latent Dirichlet Allocation

HITS - Hyperlink-Induced Topic Search

DBMS - Database Management System

QA - Question Answering

GNN - Graph Neural Network

KGQA - Knowledge Graph Question Answering

RGCN - Relational Graph Convolutional Network

SPM - Semantic Parsing Model

IE - Information Extraction

LOD - Linked Open Data

NDCG - Normalized Discounted Cumulative Gain

TF-IDF - Term Frequency-Inverse Document Frequency

ВСТУП

Актуальність теми.

Сучасне суспільство стикається з безпрецедентним зростанням обсягів неструктурованої інформації, яка генерується в різних сферах: науці, медицині, освіті, бізнесі, соціальних мережах та інших галузях. Вилучення, структуризація та використання цих даних для прийняття рішень стають критично важливими завданнями в умовах цифрової трансформації. У цьому контексті графи знань є одним із найефективніших інструментів для представлення складних взаємозв'язків між сутностями та забезпечення глибокого аналізу даних.

Актуальність теми обумовлена необхідністю розробки сучасних методів вилучення знань із неструктурованого тексту, що дозволить зменшити час і витрати на обробку великих обсягів інформації. Побудова графів знань на основі онтологій та графових баз даних забезпечує семантичну інтерпретацію даних, що сприяє підвищенню точності та релевантності результатів. Це особливо важливо для реалізації проєктів у сфері штучного інтелекту, де графи знань використовуються для розробки систем підтримки прийняття рішень, інтелектуального пошуку, автоматизованого перекладу та чат-ботів.

Додатково, використання графів знань актуальне для вирішення проблем дублювання сутностей, некоректної категоризації та забезпечення повноти даних, які часто виникають у традиційних системах обробки інформації. Впровадження таких систем сприяє ефективному управлінню знаннями в організаціях, забезпечує прозорість процесів аналізу даних та створює умови для швидкого доступу до релевантної інформації.

Тема також набуває значущості через розвиток платформ для роботи з графами знань, таких як Neo4j, які забезпечують масштабованість і продуктивність систем, що працюють із великими обсягами даних. У поєднанні з алгоритмами обробки природної мови та машинного навчання ці

технології відкривають нові можливості для аналізу текстової інформації та інтеграції даних із різних джерел.

Таким чином, розробка моделей і методів для побудови графів знань є актуальною задачею, яка відповідає сучасним потребам у створенні інформаційних систем нового покоління. Вона має значний потенціал для впровадження в різних галузях, сприяючи підвищенню ефективності обробки інформації, зменшенню впливу людського фактора та забезпеченню якісного аналізу даних.

Мета дослідження - розробка моделі, методів та засобів побудови графу знань для ефективного вилучення, структурування та зберігання знань із неструктурованих текстових даних.

Об'єкт дослідження - процес вилучення даних та побудови графів знань із використанням онтологій і графових баз даних.

Предмет дослідження - моделі, методи та інструменти для вилучення, структурування та зберігання знань із текстових даних у вигляді графів знань.

Задачі дослідження:

- Провести аналіз предметної області вилучення даних для побудови графів знань.

- Дослідити концепції побудови графів знань, процеси вилучення інформації та особливості створення онтологій.

- Розробити архітектуру, зв’язки та методологію для побудови графу знань.

- Реалізувати моделі та методи вилучення даних для графів знань із використанням графових баз даних.

- Провести оцінювання якості побудованого графу знань за критеріями точності, повноти та ефективності.

Методи дослідження

В роботі використано аналіз і синтез для вивчення предметної області та визначення вимог до моделей і методів побудови графів знань; машинне

навчання для вилучення знань із тексту та побудови моделей; методи обробки природної мови (NLP) для зв’язування сутностей і вилучення відношень; методи оцінювання точності та повноти для перевірки якості графу знань.

Наукова новизна отриманих результатів

Запропоновано методологію побудови графу знань, що включає інтеграцію онтологій, графових баз даних та методів обробки природної мови і розроблено підходи до забезпечення унікальності трійок знань у графах та мінімізації помилок категоризації сутностей.

Практичне значення результатів

Розроблені моделі та методи можуть бути використані для автоматизації процесів аналізу текстових даних, створення інформаційних систем, інтелектуального пошуку та рекомендацій. Запропоновані підходи можуть знайти застосування у бізнес-аналітиці, освіті, наукових дослідженнях та інших галузях, де необхідно працювати з великими обсягами неструктурованих даних.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 78 сторінок, і містить 27 рисунків, 4 таблиці, список використаних джерел із 52 найменувань.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ВИЛУЧЕННЯ ДАНИХ ДЛЯ ПОБУДОВИ ГРАФІВ ЗНАНЬ

1.1. Особливості та опис галузі дослідження

Граф знань - це сукупність карт, що зображують розширення знань та їх структурні зв'язки. Він ілюструє знання та їх взаємозв'язки за допомогою методів візуалізації та слугує цінним і корисним ресурсом для наукових досліджень. Знання архітектури окреслює структуру системи, її функції та іншу відповідну інформацію, дозволяючи користувачам та розробникам легко зрозуміти систему. Якщо існує програмне забезпечення, яке дозволяє людям зрозуміти архітектуру системи без вивчення документації, час, необхідний для ознайомлення з системою, буде значно скорочено. Метою цього проекту є розробка системи, здатної витягувати знання архітектури з документації програмної системи та генерувати граф знань, використовуючи цю витягнуту інформацію.

Системна документація містить вичерпний огляд архітектурного дизайну системи, включаючи її різні параметри та організаційну структуру інженерів, залучених до її розробки. Це допомагає новим користувачам швидко навчитися користуватися системою. Крім того, це дозволяє інженерам ефективно підтримувати та вдосконалювати систему без будь-яких збоїв.

Тим не менш, слід зазначити, що не вся системна документація відрізняється лаконічністю і легкістю сприйняття. Деякі розгалужені програмні системи мають обширну та відповідальну системну документацію, яка всебічно охоплює всі аспекти системи, що потребує значних витрат часу на розуміння. Особи, залучені до створення або використання системи, які прагнуть отримати всебічне розуміння цих програмних систем, вимагають значної кількості часу, присвяченого перевірці системної документації. Водночас інженерам-початківцям у сфері програмних систем зазвичай не

потрібно всебічного розуміння системи в цілому, а лише ті компоненти, які стосуються їхніх конкретних завдань. Виникла потреба в розробці допоміжного програмного забезпечення, здатного ефективно витягувати з системи відповідну інформацію, яку потребують інженери.

Технологія графів знань привернула багато уваги дослідників в останні роки після того, як була запропонована Google. Дослідження графів знань можна розділити на дві категорії: дослідження методів побудови графів знань та застосування графів знань. Дослідження методів побудови зосереджені на вилученні, представленні, об'єднанні та логічному висновку знань у графах [7], таких як правильне зв'язування сутностей та відношень з графом знань після їх вилучення з неструктурованого тексту та логічний висновок нових фактів з такого графу знань. У той час як дослідження застосування наголошують на застосуванні графів знань до практичних систем та конкретних областей. Ця стаття дає систематичний огляд застосувань графів знань. Таксономія областей застосування графів знань наведена на рисунку 1.1.

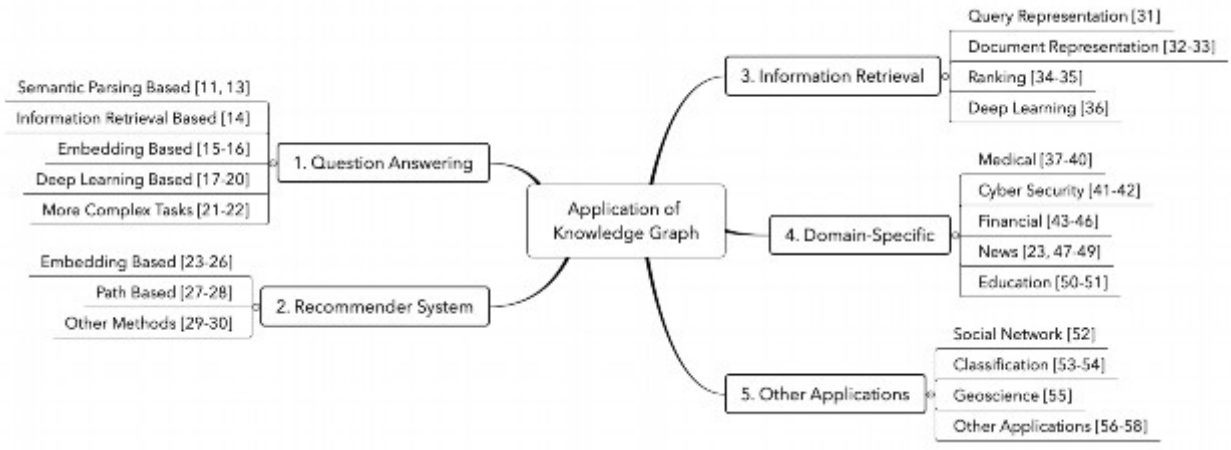

Рис. 1.1. Таксономія областей застосування графів знань

Попередні дослідження показали, що документація системи програмного забезпечення містить знання, що стосуються системи, які можуть бути структуровані в рамках знань про архітектуру системи [3]. Якщо додаткове програмне забезпечення має можливість видобувати ці

знання, відповідним чином класифікувати їх і зберігати, то шляхом пошуку в збережених знаннях можна запропонувати швидкий огляд вимог користувача.

А граф знань [13] відноситься до компіляції взаємопов'язаних даних, які представляють зв'язки між об'єктами в реальному світі. Метою його використання є систематичне впорядкування та структуроване відображення знань, що, отже, покращує доступність і розуміння наявної інформації. Управління знаннями відіграє важливу функцію в цьому секторі. Граф знань є цінним інструментом для організації та керування значними обсягами інформації, що полегшує пошук і використання відповідної інформації користувачами. Обґрунтування використання графа знань як інструменту, який допомагає користувачам ефективно розуміти необхідну інформацію в документації програмної системи, випливає з його внутрішньої властивості.

Таким чином, цей проект був задуманий як результат визнаної необхідності та життєздатної теоретичної основи. Метою цього проекту є визначення відповідної методології для вилучення знань про архітектуру із системних файлів і подальшої побудови відомого графа на основі зібраної інформації. Завданням цього проекту є створення основи для розробки допоміжного програмного забезпечення, яке полегшує користувачам ефективне отримання необхідної інформації для програмної системи.

Метою цього проекту є розробка підходу, який уможливить ефективне вилучення архітектурних знань із системної документації та подальше перетворення їх у граф знань. Цей проект буде оцінено шляхом проведення ретельного дослідження ефективності вилучення, складності впровадження та кінцевої корисності графа знань.

Процес побудови графа знань можна розділити на три окремі етапи: проектування онтології, витяг інформації та зберігання графу знань. Після завершення трьох вищезазначених процедур буде необхідний додатковий крок для оцінки графа знань. Проект охоплює додатковий набір із чотирьох конкретних цілей як пряму реакцію на вищезгадані чотири основні етапи.

Цей проект спеціально зосереджений на розробці узгодженого графа знань, ідентифікації ефективних методів вилучення інформації, пов’язаної з архітектурою, з документів програмної системи, ефективному зберіганні графа знань і подальшій оцінці графа знань за допомогою відповідних методологій. Певні дослідницькі питання розробляються безпосередньо у відповідь на вищезазначені цілі.

Онтологія служить основою для графа знань і процесу. Побудова онтології є початковою фазою у створенні графа знань. Розробка надійної онтології є надзвичайно важливою для успішного виконання проекту.

Тіло графа знань складатиметься з інформації, отриманої з системної документації програмного забезпечення. Цей етап є ключовим при оцінці доцільності побудови графу знань і оцінки якості отриманого графа знань.

Оцінка є важливим компонентом, який добре розроблений проект не може дозволити собі недооцінювати. Отже, необхідно передбачити раціональний критерій оцінки якості графа знань. Для цього проекту важливо оцінити, якою мірою граф знань точно та вичерпно представляє знання про архітектуру, що містяться в системній документації.

1.2. Аналіз схожих проектів вилучення знань з неструктурованого тексту

Численні вчені проводили дослідження з вилучення знань архітектури з неструктурованого тексту. У цьому контексті я представляю три наукові статті, які заглиблюються у вилучення інформації про архітектуру програмного забезпечення з різних текстових джерел. Ці публікації служать ілюстративними прикладами поточних дослідницьких зусиль у цій конкретній галузі.

У дослідженні [19] автори провели аналіз проектів Apache Java з метою виявлення потенційних проблем архітектури. Метою цього дослідження було

визначення переважаючих ідей, що використовуються в знаннях архітектури, шляхом застосування як якісного, так і кількісного аналізу.

Автори в [20] провели комплексне дослідження на платформі StackOverflow для збору даних від спільнот розробників, що стосуються архітектури. Шляхом емпіричного дослідження дослідники змогли виявити онлайн-публікації, які містили технологічні знання, пов'язані з архітектурою. Шляхом класифікації публікацій на категорії, пов'язані з архітектурою та програмуванням, дослідники змогли провести аналіз публікацій, пов'язаних з архітектурою, щоб виявити потенційні зв'язки з концепціями знань архітектури.

У дослідженні про знання архітектури в пошукових системах [21] автори провели дослідження, щоб з'ясувати ефективність пошукових систем в індексації концепцій та джерел знань архітектури. Дослідники оцінили ефективність пошукових систем у пошуку архітектурної інформації з текстових джерел. Дослідники провели експертизу, щоб оцінити здатність пошукових систем ефективно отримувати відповідну архітектурну інформацію з текстових джерел.

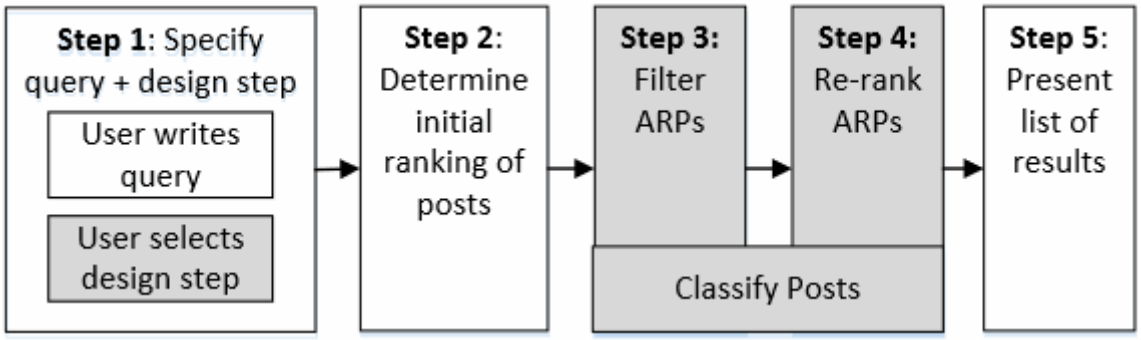

Рис. 1.2. Покращений процес пошуку (сірі елементи є розширеннями пошуку за ключовими словами) [20]

На рисунку 1.2 зображено алгоритм, який, використовується для пошуку та ранжування інформації, пов'язаної з архітектурою програмного забезпечення, у відповідь на запит користувача.

Алгоритм складається з п'яти кроків:

1. Вказати запит + крок проектування: Користувач вводить запит та вибирає крок проектування, який, ймовірно, визначає контекст пошуку (наприклад, "проектування бази даних", "проектування інтерфейсу" тощо).

2. Визначення початкового ранжування постів: На основі запиту та кроку проектування система визначає початковий список постів (ймовірно, з якоїсь бази даних або форуму) та ранжує їх за релевантністю.

3. Фільтрування ARPS: Система фільтрує пости, відбираючи лише ті, які містять ARPS (скоріше за все, це абревіатура, пов'язана з архітектурою, наприклад, "архітектурно релевантні пости").

4. Повторне ранжування ARPS: Відфільтровані пости повторно ранжуються, можливо, з урахуванням додаткових критеріїв або контексту.

5. Представлення списку результатів: Система представляє користувачеві список відранжованих постів, які, ймовірно, містять відповіді на його запит.

Вищезазначені експерименти свідчать про те, що можна витягувати знання архітектури з текстів, пов'язаних з програмним забезпеченням. Проте важливо визнати, що ці дослідження часто вимагають значних зусиль та часу, особливо якщо їх проводить один дослідник.

Іншим пов'язаним проектом є дослідження [26], яке також досліджувало вилучення знань архітектури з документації системи. Запропонований тут підхід передбачав методичну класифікацію кожного окремого речення в документації, а потім групування цих речень в окремі кластери на основі різних категорій знань архітектури. Після цього було створено програмне забезпечення з метою використання класифікованих та згрупованих текстів як відповідей на запити, представлені користувачами. Методологія передбачала сприйняття кожного речення як потенційної сутності, відходячи від типових методів вилучення інформації, які покладаються на розпізнавання іменованих сутностей для вилучення сутностей з речень. Ця особа розробила навчальні модулі, які містять

різноманітні концепції архітектури та їх відповідні значення. На основі емпіричних результатів можна зробити висновок, що ці дослідницькі зусилля демонструють ефективність у вилученні інформації з документації системи. Незважаючи на різкий контраст у наших відповідних підходах.

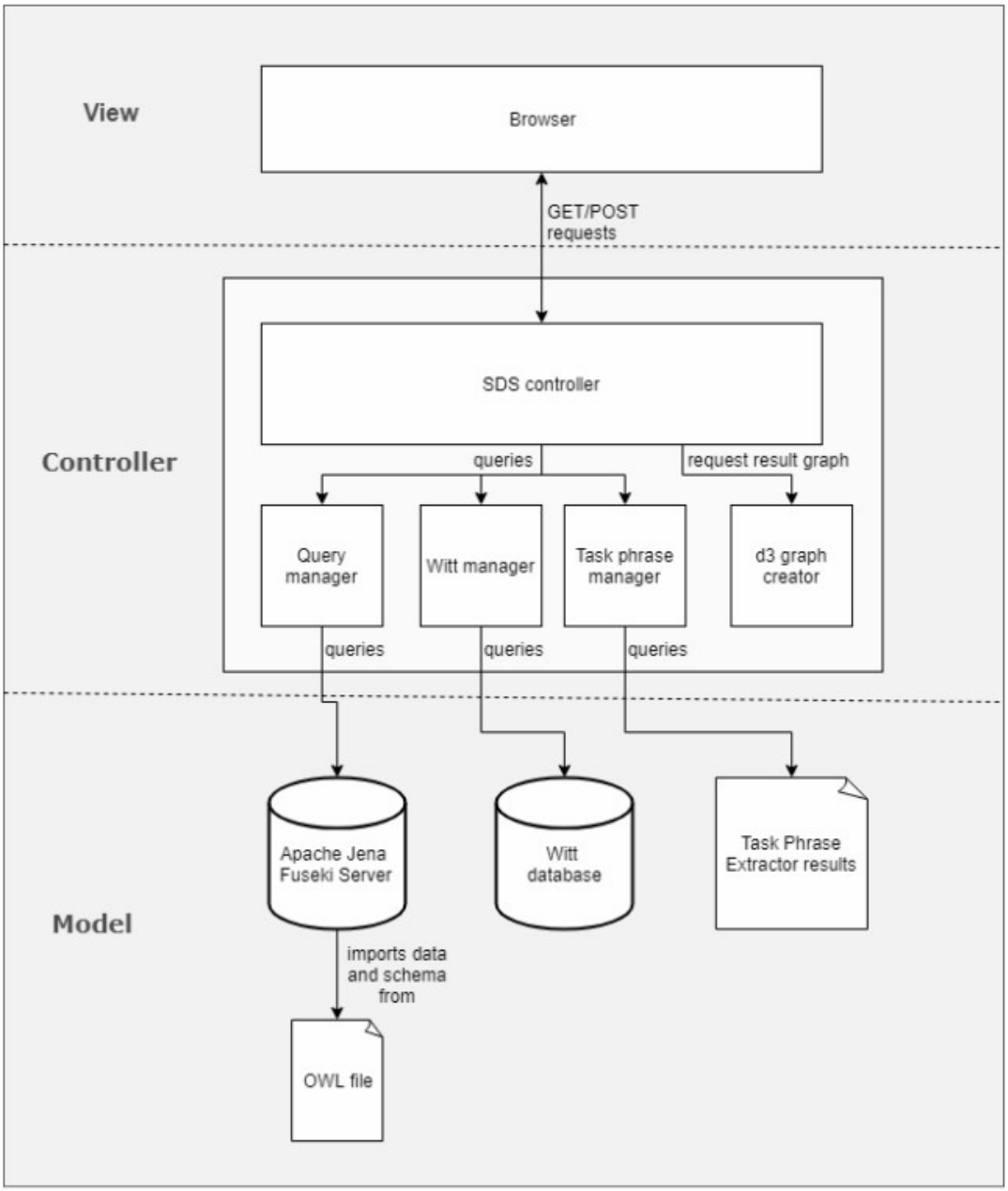

Рис. 1.3. Архітектура системи підтримки документації програмного забезпечення

Проект, розроблений в [26] має на меті створити систему питань-відповідей, яка ефективно витягує знання архітектури з заданої документації

системи та надає користувачам доступ до цих знань у відповідь на їхні запитання. Незважаючи на обмеження щодо можливості користувача ставити певні запити, очевидно, що система питань-відповідей здатна виконувати завдання вилучення знань архітектури з документації системи.

Щоб переконатися в перевазі цього проекту порівняно з аналогічними проектами, можна розробити систему питань-відповідей, щоб полегшити порівняльний аналіз їх відповідної продуктивності.

1.3. Дослідження концепції побудови графу знань

Згідно з визначенням [3], архітектура програмного забезпечення системи — це набір структур, необхідних для міркування про систему. Структура складається з набору елементів, укріплених зв'язками. Важливо зауважити, що, незважаючи на важливість цих структур для системи програмного забезпечення, жодну з них не слід розглядати як цілісну архітектуру. Структура вважається архітектурною лише тоді, коли вона сприяє оцінці характеристик системи або зацікавлених сторін. Іншими словами, архітектура складається зі структур, які сприяють міркуванню щодо характеристик і зацікавлених сторін системи.

Архітектура системи є важливим інструментом для перетворення нематеріальних ідей у конкретні та матеріальні програмні продукти. Знання про архітектуру складаються з проектування архітектури, а також проектних рішень, припущень, контексту та інших факторів, які разом визначають, чому певна система така, якою вона є.

Концепція Knowledge Graph була вперше оголошена компанією Google у 2012 році. Її основною метою було створення повного представлення концепцій, речей, подій та їх взаємозв'язку у фізичному світі. Це послужило фундаментальним будівельним блоком для розвитку складних технологій пошукових систем. Граф знань можна описати як складну мережу, що включає взаємопов'язані взаємодії, які пов'язують різні категорії інформації.

Графи знань дозволяють досліджувати проблеми з реляційної точки зору. Надане цитування посилається на примітивний приклад графа знань із двома видами, як показано на рисунку 1.4 [12]. У перспективі онтології звичайні метазв'язки та ієрархічні метазв'язки представлені пунктирними лініями помаранчевого та чорного кольорів відповідно.

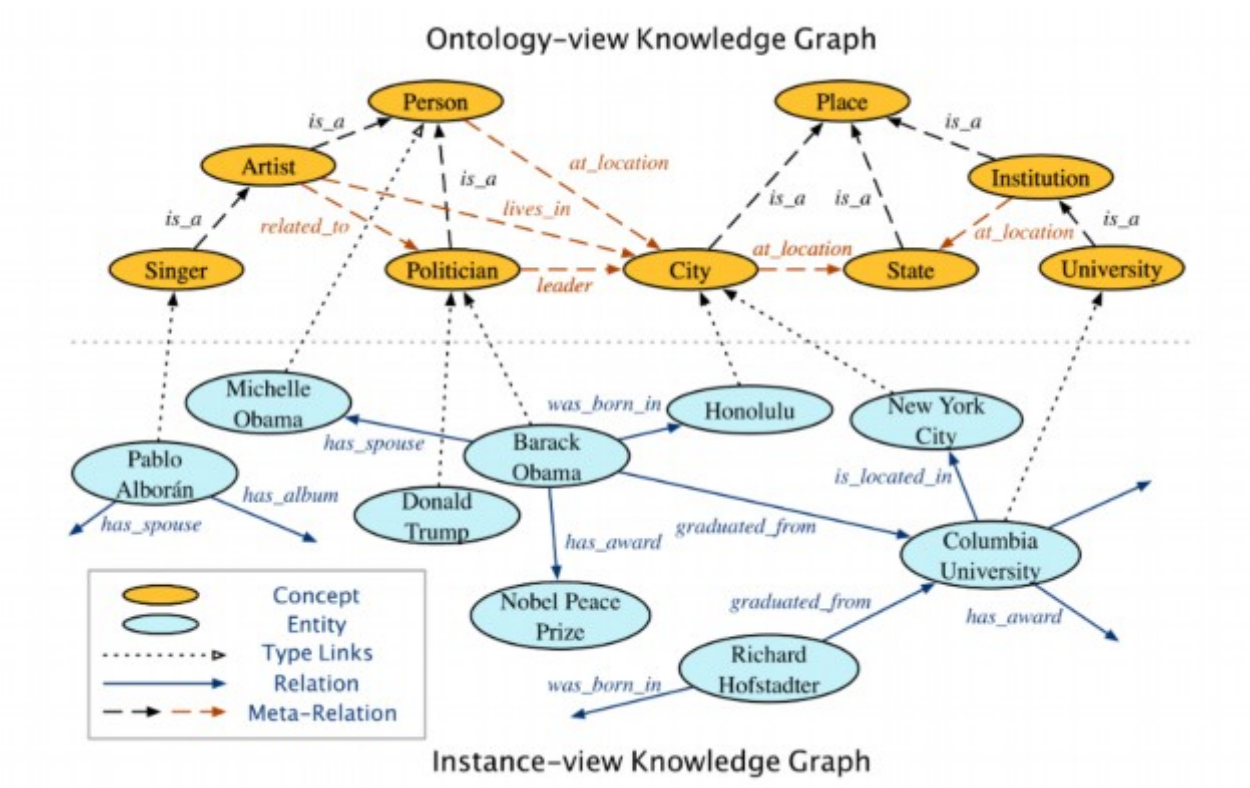

Рис. 1.4. Приклад графа знань із двома представленнями

Граф знань — це комплексна формальна структура, яка представляє семантичну інформацію. У цій структурі вузли символізують семантичні символи, а краї символізують семантичні відносини між цими символами. Він слугує механізмом для точної характеристики складного, неоднорідного та обширного корпусу інформації. Граф знань стосується фундаментального дослідження найефективніших засобів представлення та структурування інформації для полегшення ефективного пошуку, інтеграції та аналізу знань, отриманих із різних джерел. Використання підходу, заснованого на графах,

дозволяє графу знань ефективно відображати та встановлювати зв’язки між величезною кількістю даних, таким чином дозволяючи отримати більш складне та складне розуміння взаємозв’язків і взаємозалежностей у даних.

Поточний ландшафт високоякісних, великомасштабних, відкритих графів знань включає добре відомі приклади, такі як DBpedia [2], Wikidata [33], ConceptNet [19] та інші, які охоплюють різноманітні багатомовні домени.

ConceptNet - це семантична мережа знань здорового глузду, яка на даний момент містить 1,6 мільйона ребер, що з'єднують понад 300 000 вузлів. Вузли - це напівструктуровані фрагменти англійської мови, пов'язані між собою онтологією з двадцяти семантичних відношень. Частковий знімок фактичних знань у ConceptNet наведено на рис. 1.5.

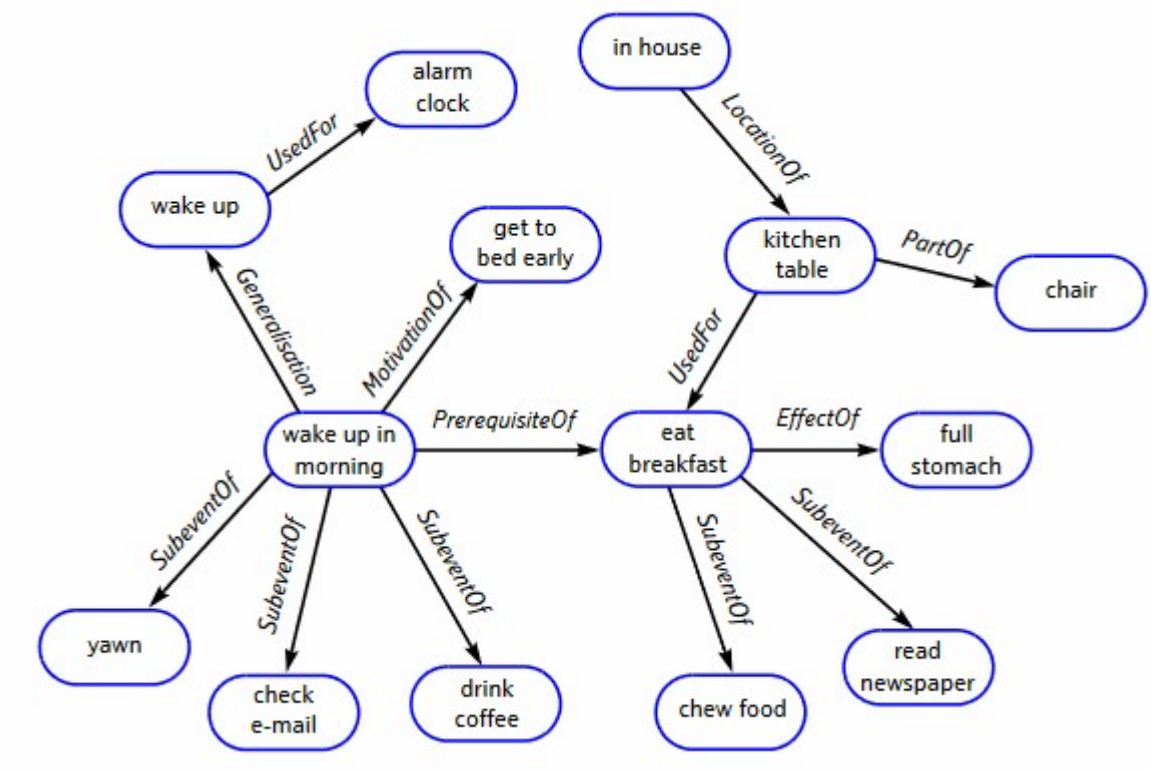

Рис. 1.5. Фрагмент семантичної мережі знань здорового глузду ConceptNet

На рисунку 1.5 складні (на відміну від простих) концепції представлені напівструктурованою англійською мовою шляхом поєднання дієслова

(наприклад, «пити») з іменниковою фразою («кава») або прийменниковою фразою («вранці»).

Розглядаючи розміри ConceptNet, WordNet і Cyc, ми хотіли б застерегти, що цифри в кращому випадку дають слабке уявлення для порівняння. Оскільки ConceptNet, WordNet і Cyc використовують різні представлення знань, числові порівняння між представленнями можуть бути не особливо значущими.

Технологія графів знань включає три основні дослідницькі області: представлення знань, побудова графів знань і застосування графів знань. В академічних колах не існує загальновизнаного визначення знання, але загалом його можна розуміти як розуміння, обізнаність або обізнаність, якими люди володіють щодо певного предмета чи області [16]. Це зазвичай передбачає отримання та засвоєння інформації, фактів, концепцій, принципів і теорій через освіту , навчання, досвід або дослідження. А знання вважається обґрунтованою істинною вірою, підкріпленою доказами та міркуваннями, і воно дозволяє людям розуміти, інтерпретувати та ефективно застосовувати інформацію. Існують різні форми знання, включаючи знання предметної області, енциклопедичні знання, знання сценаріїв, лінгвістичні знання та здоровий глузд.

Побудова графа знань є складною та важливою областю досліджень штучного інтелекту в даний час. Особливо це стосується автоматизованої побудови графів знань. Побудова системи знань, отримання знань, злиття знань, зберігання знань і висновок знань є ключовими етапами розробки графа знань.

Побудова системи знань зазвичай передбачає визначення того, як представляти знання, з побудовою онтології [11] як її центрального компонента. Для цього проекту метою побудови системи знань є розробка онтології для характеристики цільових знань. Виключення інших процесів призводить до спрощення завдання створення системи знань, яке можна розглядати як завдання проектування онтології.

Отримання знань [17] — це формальний процес, що передбачає отримання структурованих знань із об’ємних текстових даних. Складність і стратегії отримання знань можуть відрізнятися залежно від типу джерела даних, що використовується. Якщо більшість даних знаходиться в організованому або напівструктурованому форматі, процес отримання знань з них може бути досить простим. Тим не менш, процес створення графів знань іноді включає роботу з неструктурованим матеріалом. Управління неструктурованими даними вимагає використання різноманітних стратегій для вилучення інформації, щоб виявити важливі зв’язки та сутності, вбудовані в текстовий вміст. Враховуючи те, що метою цього проекту є вилучення архітектурних знань із документації програмної системи та побудова графа знань, процес отримання знань у цьому проекті дорівнює завданню вилучення інформації.

Інтеграція систем знань і прикладів становить фундаментальні елементи об’єднання знань [18]. У ситуаціях, коли дані надходять з кількох джерел, необхідним стає процес синтезу знань. Однак цей проект не включатиме злиття знань, оскільки він має на меті зосередитися виключно на одному графі знань.

Існуючі графи знань систематично організовуються та керуються під час процесу зберігання знань. Після злиття знань новоотримана інформація з оцінкою якості (можливо, за участю людини) повинна бути включена в базу знань. Зараз графові бази даних, серед яких neo4j [22] є однією з найпопулярніших, переважно використовуються для зберігання графів знань.

За наявності недостатніх даних результуючі графіки знань зазвичай містять прогалини або недоліки. Дефіцит даних може ускладнити збагачення графіка лише за допомогою вилучення та злиття. За таких обставин необхідний висновок за знаннями [24], щоб завершити відсутні зв’язки та сутності графа. Однак слід зазначити, що цей проект не включає міркування знань, оскільки його основна мета не передбачає проведення досліджень щодо самозбагачення графів знань.

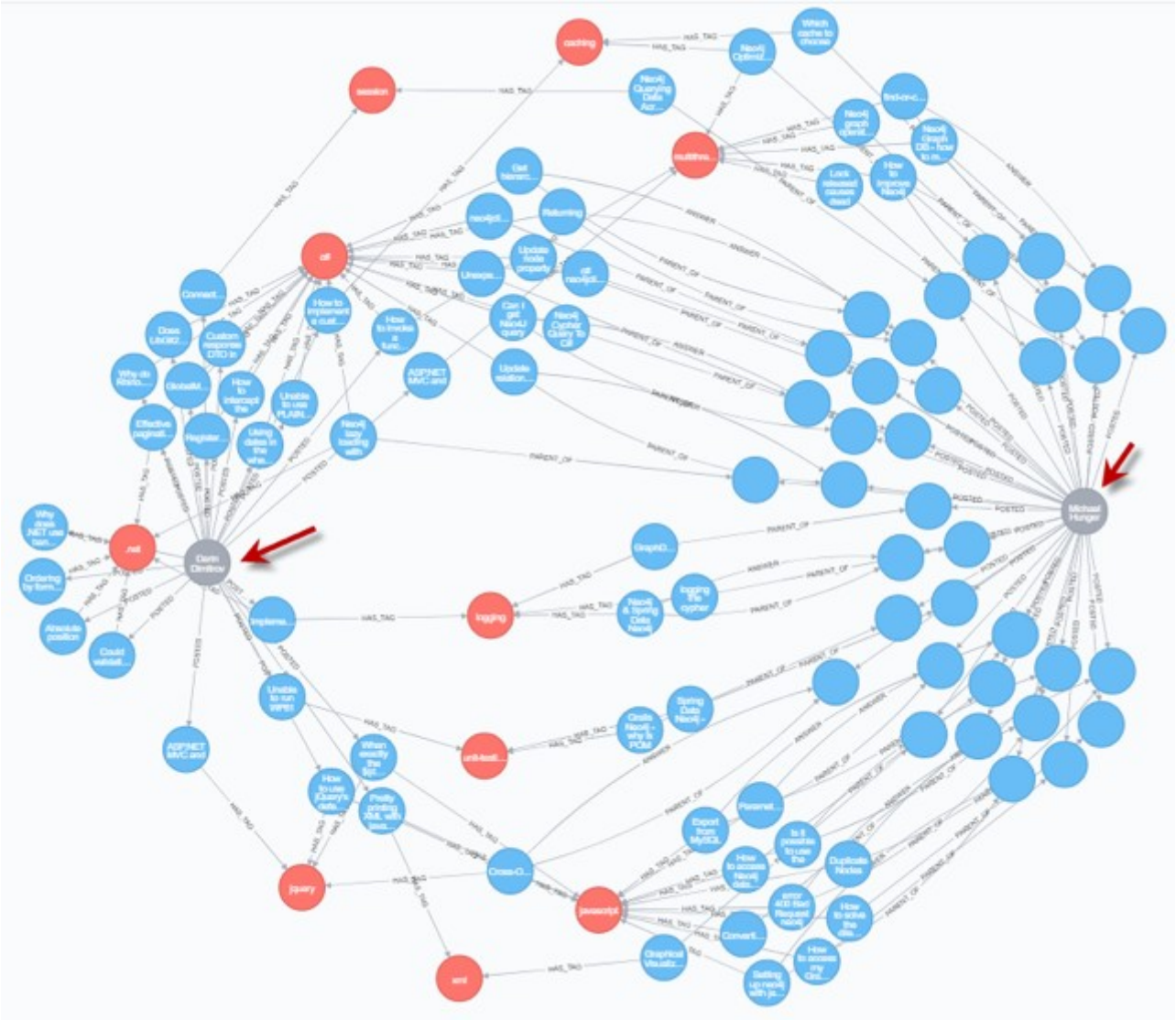

Рис. 1.6. Приклад графової бази даних neo4j

1.4. Особливості побудови онтологій

Побудова системи знань зазвичай передбачає визначення того, як представляти знання, з побудовою онтології [11] як її центрального компонента. Онтологія пропонує структурований і організований метод для представлення знань, що забезпечує ефективне вилучення знань, інтеграцію та міркування в графі знань. Для цього проекту метою побудови системи знань є розробка онтології для характеристики цільових знань. Виключення інших процесів призводить до спрощення завдання створення системи знань, яке можна розглядати як завдання проектування онтології.

Онтологія — це формальна, явна специфікація спільної концептуалізації в межах певної області. Він визначає категорію знань, поняття та сутності, які належать до кожної категорії, атрибути, якими володіє конкретна категорія понять та сутностей , а також семантичні зв'язки між поняттями та сутностями. Це методологія моделювання знань, щоб комп'ютери могли осягнути людські знання.

Онтологія визначається як кортеж

$$o =< C, I, R, T, V, \leq, \perp, \in, => $$

такий, що:

C – набір класів;

I — набір індивідів;

R – множина відношень;

T — набір типів даних;

V — набір значень (C, I, R, T є попарно непересічними);

≤ - це відношення на (C x C) u (R x R) u (T x T), яке називається спеціалізацією;

I – це відношення на (C x C) u (R x R) u (T x T), яке називається виключенням;

є – відношення над (l x C) u (V x T), яке називається інстанціюванням;

= є відношенням над (l x P) x (I u V) називається присвоєння;

Три найвидатніші характеристики онтології полягають у тому, що вона є явною, формальною та спільною. По-перше, це зрозуміло, оскільки поняття визначені та сформульовані точно. Він також є формальним, оскільки його можна зчитувати машиною та обробляти комп'ютерами. Нарешті, він є спільним, тому що він приймається та визнається групою, а не обмежується приватним використанням однієї особи.

Щоб розробити онтологію, потрібно здійснити кілька ключових кроків, які можуть включати:

- Вказати область і домен: визначити конкретну область або предметну область онтології. І визначити, які функції домену слід включити або виключити з онтології.

- Визначити поняття: визначити ключові поняття або сутності в домені.

- Вказати зв’язки: визначити зв'язки між поняттями. Перевірити, чи мають поняття будь-які ієрархічні, асоціативні чи інші типи зв’язків. І вказати характеристики або атрибути, які характеризують поняття та їхні зв’язки.

- Встановити структуру онтології: визначити організацію онтології. І визначити класи, підкласи та екземпляри більш детально.

- Вибрати мову онтології: вибрати відповідну мову онтології або структуру для представлення онтології. RDF (Resource Description Framework) [27], OWL (Web Ontology Language) [21] і RDFS (RDF Schema) [4] є поширеними мовами онтології.

- Перевірити онтологію: перевірити онтологію, перевіривши її на послідовність, точність і повноту.Після ефективного проектування онтології її можна реалізувати на графі знань. Ця реалізація дозволяє організовувати, робити запити та міркувати про предметно-спеціальні знання.

1.5. Процес вилучення інформації для графів

Основна мета вилучення інформації для графів знань полягає в тому, щоб витягти трійки, що складаються з сутностей та їхніх відповідних зв’язків. Отже, цей процес, як правило, складається з двох різних етапів: зв’язування об’єктів і виділення зв’язків. У наступних розділах буде коротко представлено два етапи вилучення інформації.

1.5.1. Зв'язування сутностей

Пов’язування сутностей можна далі розділити на два основні процеси:

- розпізнавання іменованих сутностей (NER) [15]: Початковий крок цього процесу включає ідентифікацію та вилучення іменованих сутностей із

неструктурованих текстових даних. Ці сутності можуть включати імена окремих осіб, назви організацій і назви географічних місць.

- Усунення неоднозначності сутності [6]: Процес усунення неоднозначності сутності передбачає розв’язання неоднозначних згадок сутностей у базі знань, гарантуючи, що вони правильно пов’язані з відповідними сутностями. Завдання може передбачати диференціацію сутностей, які мають схожі назви, або диференціацію кількох сутностей, які пов’язані з тією самою згадкою.

Вищезазначені процеси разом сприяють створенню зв’язків між об’єктами, згаданими в тексті, та пов’язаними з ними об’єктами в базі знань. Розпізнавання іменованих сутностей забезпечує більш точне та структуроване представлення інформації.

1.5.2. Вилучення відношень

Вилучення відношень — це завдання передбачення атрибутів і відношень для сутностей у реченні [14]. Вилучення зв’язків можна класифікувати на дві основні категорії: підходи на основі правил і підходи на основі машинного навчання. Вилучення зв’язків на основі машинного навчання можна класифікувати на кілька підтипів, включаючи контрольовані, слабко контрольовані, неточно контрольовані та неконтрольовані методи вилучення зв’язків. Підходи вилучення зв’язків на основі машинного навчання широко використовуються вченими в сучасному науковому середовищі. Використовуючи ці конкретні стратегії, дуже важливо ретельно вибрати відповідну модель.

Традиційно зв'язування сутностей і вилучення зв'язків розглядаються як різні види діяльності. Однак останні досягнення в глибокому навчанні дозволили об’єднати ці два завдання в єдину операцію. У сфері вилучення інформації зростає тенденція до прийняття підходів кооперативного вилучення. Використання спільної моделі полегшує одночасну інтеграцію підмоделей зв’язування сутностей і вилучення зв’язків. Цей підхід дозволяє

об’єднати спільну інформацію між двома завданнями та служить для пом’якшення потенційних недоліків, пов’язаних із поширенням помилок під час процесу вилучення. Ці методи уможливлюють одночасне прогнозування зв’язків і сутностей, отже, оптимізуючи процес вилучення знань. Застосовуючи цей конкретний підхід, стає можливим ввести речення, а потім витягти трійки сутностей разом із пов’язаними зв’язками, використовуючи спільну модель для розпізнавання сутностей і виділення зв’язків.

1.6. Платформи та інструменти для зберігання знань

Графи знань зазвичай використовуються для моделювання та представлення знань у графовій структурі, де інформація зберігається як дані графа. Під час експериментальної фази невеликі графи знань зазвичай зберігають свої дані у файлах. Коли ви маєте справу з обширними графами знань, які вимагають запитів, змін і навичок міркування, доцільно використовувати систему керування базами даних (СУБД) для ефективного зберігання знань. Системи управління реляційними базами даних (СУБД), графові СУБД і сховища RDF (Resource Description Framework) – це СУБД, які зазвичай використовуються для зберігання граф знань. Графові СУБД і RDF-сховища добре підходять для зберігання та пошуку даних на основі графів завдяки використанню в них графової моделі даних. Це дозволяє їм ефективно та ефективно зберігати графи знань безпосередньо.

Neo4j, JanusGraph, ArangoDB, TigerGraph та інші порівняльні бази даних [23] часто використовуються для зберігання графів знань як баз даних графів.

Отже, на основі вищезазначеного огляду літератури можна отримати фундаментальне розуміння реалізації проекту. Початковий етап цього проекту передбачає розробку онтології. Враховуючи, що область онтології раніше була визначена як знання про архітектуру в документації програмного забезпечення, цей етап включає ідентифікацію концепцій, специфікацію

взаємозв’язків, встановлення структури онтології, вибір відповідної мови онтології та перевірку онтології. Наступним етапом цього проекту є витяг інформації. Враховуючи те, що вилучення сутності, що зв’язує й відношення, може виконуватися одночасно за допомогою моделей спільного вилучення, використання таких методів може спростити процес вилучення інформації. Наступний етап цього проекту передбачає зберігання графа знань, що вимагає ідентифікації відповідної бази даних для цієї мети. Завершальні етапи включатимуть оцінку графу знань із порівняльним аналізом усього проекту з іншими проектами, які мають подібні цілі.

Висновки до розділу

В даному розділі проведено проведено дослідження предметної області вилучення даних для побудови графів знань. Вилучення знань для побудови графів знань є актуальною галуззю, що спрямована на інтеграцію, структурування та організацію інформації з різноманітних джерел. Графи знань надають можливість ефективного представлення даних у вигляді пов’язаних сутностей і відношень, сприяючи поліпшенню інформаційного пошуку, автоматизації прийняття рішень і семантичного аналізу.

Дослідження показало, що більшість сучасних проектів базуються на методах обробки природної мови (NLP), таких як розпізнавання сутностей, вилучення відношень і класифікація. Успішні проекти (наприклад, DBpedia, YAGO та Wikidata) використовують комбінацію ручної роботи та автоматизованих алгоритмів для створення високоякісних графів знань.

Розглянуто концепцію побудови графа знань та виявлено особливості побудови онтологій. Розробка онтологій є важливою складовою побудови графа знань, оскільки вони визначають концептуальні моделі предметної області, терміни та зв'язки між ними. Використання онтологій сприяє семантичній сумісності даних і забезпечує їхню інтероперабельність.

Проаналізовано процес вилучення інформації для графів знань. Зв'язування сутностей включає ідентифікацію та зіставлення сутностей з різних джерел для уникнення дублювання. вилучення відношень фокусується на виявленні зв'язків між сутностями на основі текстових і структурованих джерел. Розглянуто платформи та інструменти для зберігання знань.

Таким чином, проведений аналіз створює основу для подальшого вивчення методів і моделей побудови графів знань, а також розробки інноваційних підходів до вилучення даних із неструктурованих джерел.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ДИЗАЙНУ ОНТОЛОГІЙ, СТВОРЕННЯ АРХІТЕКТУРИ, ЗВ'ЯЗКІВ, МЕТОДОЛОГІЇ. ПІДГОТОВКА ДАНИХ

Цей розділ представляє процеси розробки онтології. Спочатку будуть розроблені концепції архітектурного знання і визначені зв'язки між концепціями. Потім встановлюється структура онтології, вибирається мова для представлення онтології та мето методологія перевірки онтології. Робочий процес цього етапу зображено на рисунку 2.1.

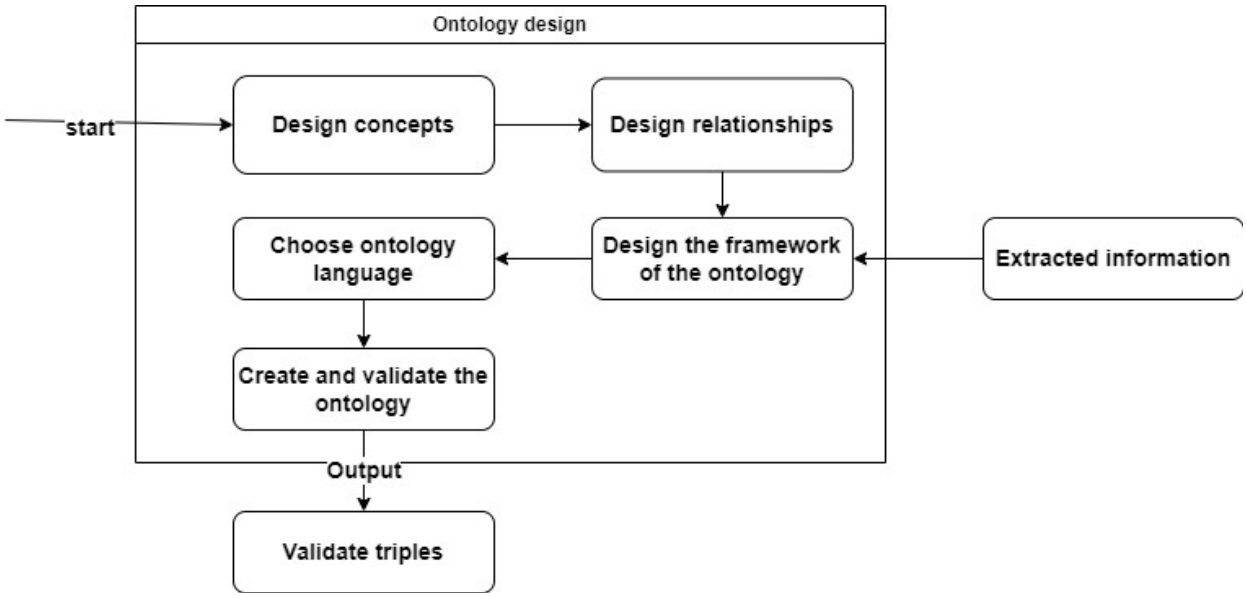

Рис. 2.1. Робочий процес етапу проектування онтології

2.1. Концепції знань про архітектуру

Розділ 2.1 представляє знання про архітектуру. Це може допомогти в розробці концепцій в області архітектурних знань.

Вкрай важливо, щоб розроблені концепції всебічно охоплювали всю архітектурну інформацію, що міститься в документації програмної системи. Однак генерувати нові концепції без будь-якої попередньої основи, очевидно, неможливо. Щоб забезпечити валідність концепцій, необхідно спиратися на усталені рамки, які пройшли ретельну перевірку. Отже,

розсудливим підходом є синтез внесків попередніх поколінь та включення поширених концепцій архітектурних знань. На основі книги "Архітектура програмного забезпечення: Основи, теорія та практика" [3] взято десять концепцій. Ці десять концепцій охоплюють три фундаментальні аспекти системи, а саме метаінформацію системи, структури системи/програми та інтерфейс користувача. Здобувши розуміння цих концепцій, можна розвинути базове розуміння структури, функціональності та взаємодії користувача з системою.

Нижче наведено список десяти концепцій архітектурних знань та їх визначення:

1. Виконавча платформа:

Ця концепція стосується технології або інфраструктури, на якій працює програмне забезпечення. Вона також включає технологію розгортання.

2. Імплементація:

Стосується численних аспектів реалізації системи, таких як проблеми програмування/вихідного коду, відображення алгоритмів у вихідний код та реалізація API.

3. Дизайн:

Ця концепція описує створення плану або креслення програмної системи або компонента. Вона передбачає визначення структури, поведінки та взаємодії різних програмних елементів для задоволення конкретних вимог. Як правило, дизайн передбачає визначення та представлення основних характеристик програмної системи, нехтуючи нерелевантною інформацією.

4. Архітектура:

Архітектура надає огляд процедур та структур системи. Цю концепцію можна витягти з речень, що описують загальну архітектуру системи.

5. Вимоги:

Ця концепція стосується письмового опису можливості, функції або умови, якою повинна володіти програмна система для досягнення певної мети або задоволення потреб зацікавленої сторони. Як правило, академічне

визначення вимоги в програмній інженерії охоплює такі елементи, як намір, специфікація, повнота, узгодженість, простежуваність, перевірка, еволюція та залучення зацікавлених сторін.

6. Знання предметної області:

Ця концепція стосується знань та досвіду щодо галузі чи дисципліни, в якій розробляється або використовується програмна система. Вона містить академічне визначення знань, пов'язаних з цією конкретною областю, разом із включенням фонової інформації про систему в сценарії практичного застосування.

7. Структура:

Ця концепція описує певну структуру, таку як шар або компонент. Вона може з'являтися в будь-якому реченні, що описує дезінтеграцію, фрагменти, компоненти або шари.

8. Поведінка:

Ця концепція стосується процесів, створення/видалення та взаємодії між компонентами, послідовностями, паралелізмом, потоками, подіями та діями. І на відміну від процесу розробки концепції, основна увага, як правило, приділяється поведінці системи.

9. Модель даних:

Ця концепція стосується структур даних, включаючи такі структури, як дерева, таблиці.

10. Інтерфейс користувача:

Ця концепція описує засоби, за допомогою яких користувачі взаємодіють із програмним додатком або системою. Вона включає зовнішній вигляд, використання вікон, значків, кольорів, форм та навігації, серед іншого. Вона підкреслює застосування принципів дизайну, орієнтованого на користувача, та міркувань щодо зручності використання для створення інтерфейсів, які задовольняють вимоги користувачів, підвищують продуктивність та забезпечують приємний користувацький досвід.

Концепції, що представляють архітектурні знання, слугуватимуть кресленнями, отриманими з документації системи. Вони є ключовими концепціями онтології, розробленої в цьому проекті.

2.2. Представлення зв’язків в архітектурі і фреймворк онтології

У книзі [3] представлені зв’язки між багатьма концепціями знань про архітектуру. Враховуючи, що зв’язки, показані в книзі, були раніше підтверджені як раціональні, цей проект безпосередньо використовуватиме зв’язки в книзі.

Загалом у цьому проекті представлено дев’ять унікальних зв’язків: «втілювати», «направляти», «представляти», «описувати», «зустрічати», «використовувати», «впроваджувати», «виконувати» та «підтримувати». Визначення кожного відношення збігається з його буквальним тлумаченням у словнику. Рисунок 2.2 відображає всі зв’язки між поняттями. Десять понять пов'язані між собою різними відношеннями.

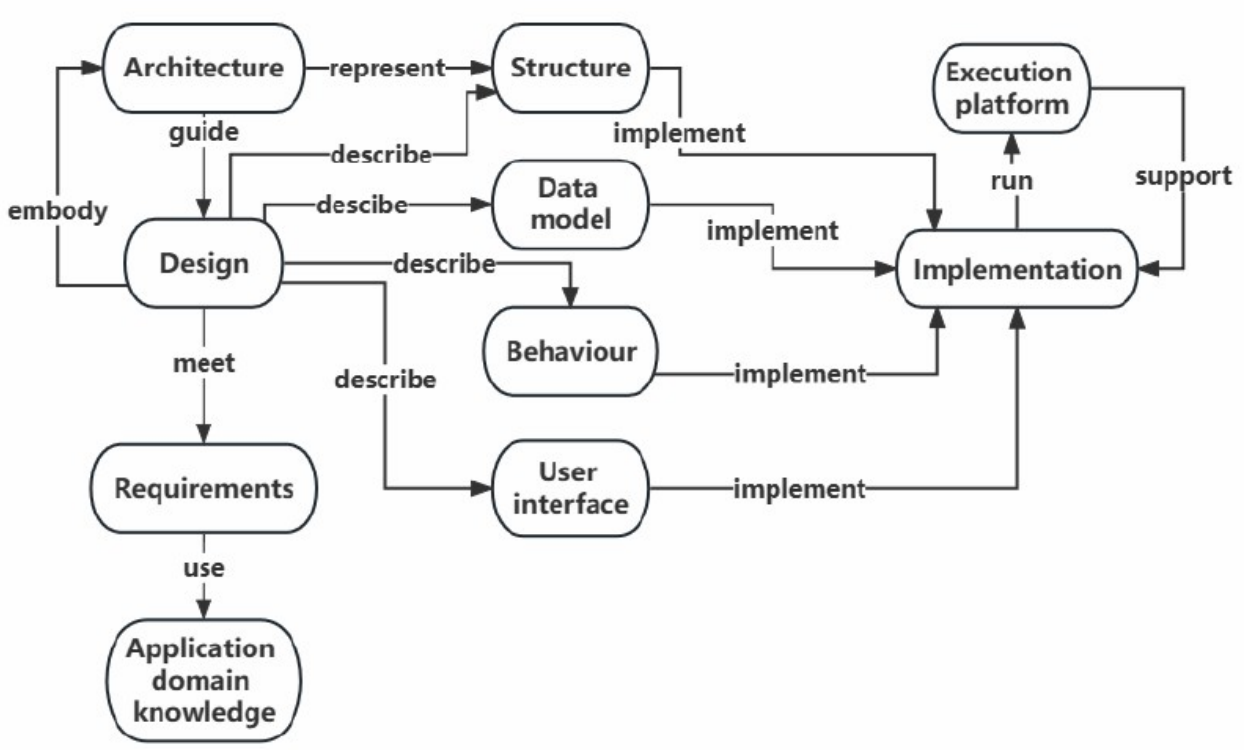

Рис. 2.2. Представлення зв’язків між концепціями

Для цього проекту головне, що слід враховувати при проектуванні основи онтології, — це класи та ієрархія між класами. Раніше розроблені концепції можуть безпосередньо використовуватися як класи онтології. Щоб встановити добре обґрунтовану ієрархію між класами, доцільно прийняти методологію, яка передбачає перегляд і дотримання існуючих досліджень і наукових внесків. У книзі [3] також представлено ієрархію класів.

Класи діляться на три групи. Базуючись на зосередженості на знаннях і контексті, клас «Знання домену програми» включено до групи «Контекст системи». Група «Вид» містить класи «Структура», «Поведінка», «Модель даних» і «Інтерфейс користувача», оскільки всі вони можуть стосуватися видимих процесів, структур або компонентів системи програмного забезпечення. Група «системні абстрактні класи» містить усі інші класи, які складаються з «Реалізації», «Платформи виконання», «Архітектури», «Дизайну» та «Вимог». Серед яких « Платформа виконання» та «Дизайн» є підкласами «Реалізація» та «Архітектура» відповідно.

Після завдання вилучення інформації, представленого в третьому розділі, витягнуту інформацію використовуватимуть екземпляри онтології.

2.3. Мова онтології та її перевірка

Protégé [32] - це комплексне середовище редагування онтологій, яке забезпечує повну підтримку мови веб-онтологій OWL 2 [21] та підключення до логічних резонаторів в оперативній пам'яті. Завдяки високому рівню зручності цей редактор буде використовуватися протягом усього проекту. Отже, для представлення онтології в цьому проекті було обрано мову OWL. Мова OWL - це мова опису онтологій, розроблена W3C, і її можна розділити на три рівні виразності: OWL-Lite (найнижча виразність), OWL-DL (середня виразність) та OWL-Full (найвища виразність).

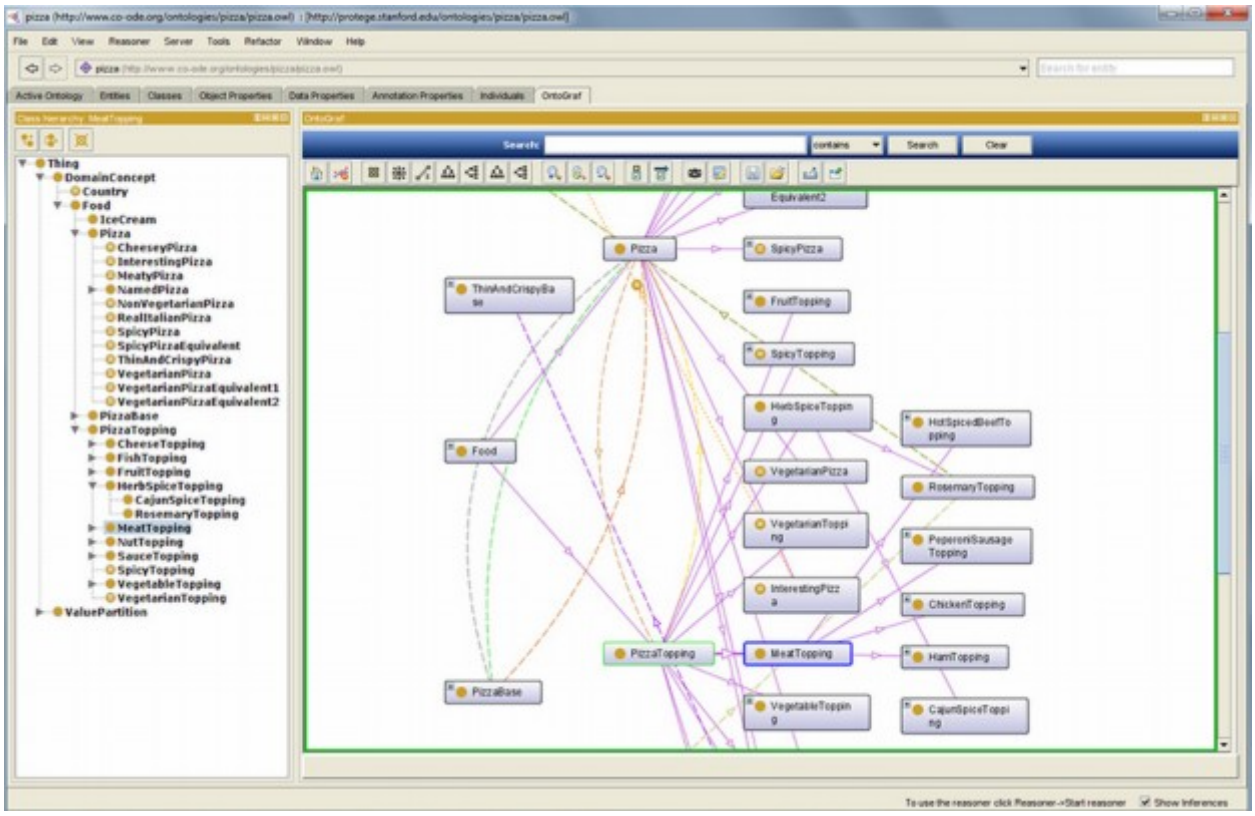

Рис. 2.3. Платформ редагування онтологій Protégé

У цьому проекті буде використана мова OWL-DL, яка також є мовою за замовчуванням для редактора. Використовуючи цей редактор для розробки онтології, він має можливість автоматично генерувати файл owl. У Protégé також є утиліта візуалізації. На рис. 2.4 показано візуалізацію ієрархії класів.

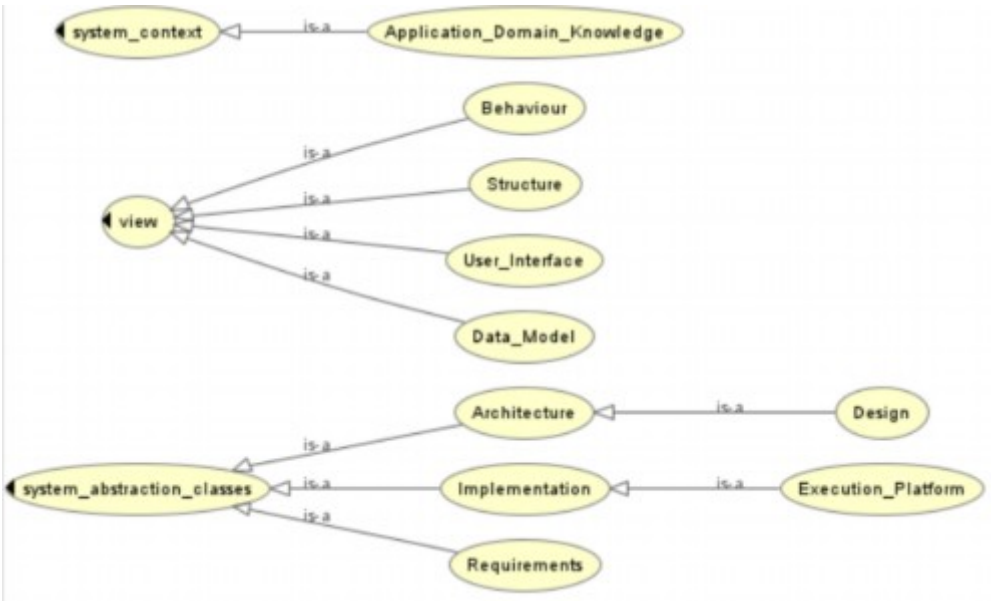

Рис. 2.4. Візуалізація ієрархії класів

Protégé включає резонатор Hermit [10]. HermiT - це резонатор, призначений для побудови онтологій з використанням мови веб-онтологій (OWL). Коли йому надається файл OWL, HermiT має можливість ідентифікувати різні типи, характеристики помилок ієрархії, а також відповідну область атрибутів, помилки обмеження діапазону та інші пов'язані проблеми.

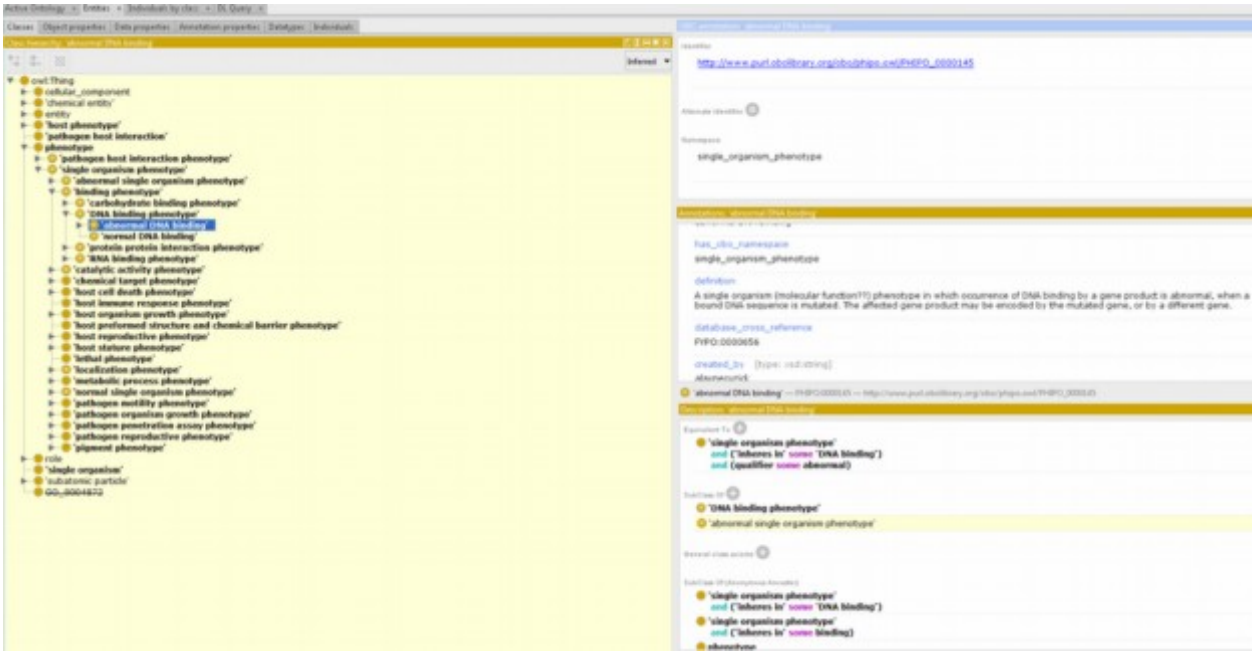

Рис. 2.5. Резонатор HermiT для побудови онтологій

Ключові особливості HermiT:

- Підтримка OWL 2: HermiT підтримує всі профілі OWL 2, включаючи OWL 2 DL.

- Ефективність: HermiT є одним з найефективніших резонаторів для OWL.

- Відкритий вихідний код: HermiT є програмним забезпеченням з відкритим вихідним кодом, що дозволяє його вільно використовувати та модифікувати.

У цьому проекті використання функції виведення HermiT не буде необхідним, оскільки запити та зберігання графа знань будуть здійснюватися за допомогою альтернативного програмного забезпечення.

Основна мета полягає в тому, щоб запитувати вхідні сутності для виявлення будь-яких дублікатів та з'ясувати, чи належать вони принаймні до одного з десяти класів. Трійки, що містять неправильні сутності, будуть видалені. Після виправлення всіх помилок було перевірено 588 різних трійок.

2.4. Процес підготовки наборів даних

У цьому проекті використовуватиметься модель спільного вилучення. Таким чином, необхідно отримати набори даних для цілей навчання, перевірки, тестування та оцінки. У цьому розділі представлено підготовку наборів даних, обрані джерела даних, створюється базова правда, набори даних для навчання, перевірки та тестування. Також показано процес дедуплікації трійок. Робочий процес цього етапу зображено на рисунку 2.6.

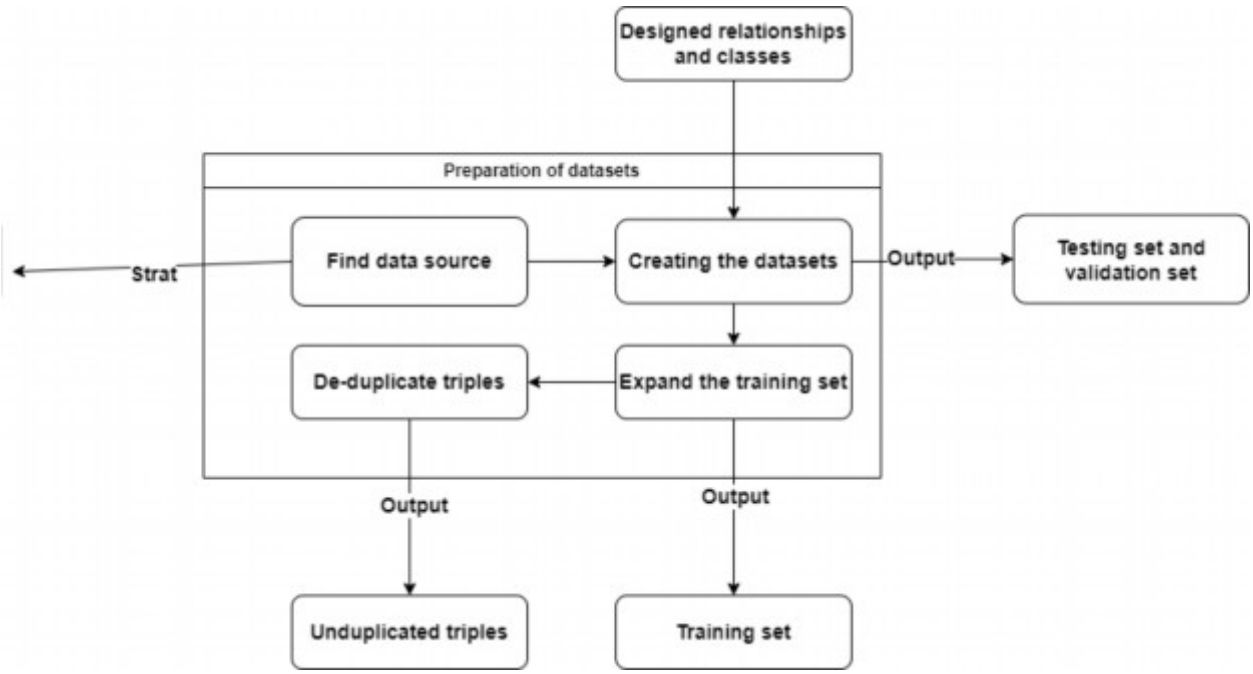

Рис. 2.6. Робочий процес етапу підготовки наборів даних

2.4.1. Джерело даних

Необхідно ретельно вибрати відповідну системну документацію, яка слугуватиме джерелом даних для набору даних, які буде використовуватися

для оцінки. Тому було вибрано документацію програмного забезпечення Eclipse. Ця документація має чітко визначену структуру та містить значну кількість архітектурних знань, які також описані в [5]. Довжина та широта архітектурних тем у документі роблять його ідеальним предметом для дослідження. Ця системна документація поділена на 78 параграфів. Абзаци є джерелом базового набору даних.

Підручники з вивчення архітектури програмного забезпечення зазвичай містять загальну інформацію про знання архітектури програмного забезпечення без згадування будь-яких конкретних програм. Як наслідок, вони є відповідними джерелами. Для цього проекту вибрано перші два розділи четвертого видання архітектури програмного забезпечення на практиці [3]. Дві глави книги поділені на 135 параграфів, разом із 78 параграфами документації програмного забезпечення Eclipse, вони є джерелом необробленого набору даних.

2.4.2. Створення наборів даних

Початковий етап передбачає процес маркування вихідних даних. Мітки будуть однією або кількома з десяти концепцій знань про архітектуру та зв’язки будуть одними з дев’яти зв’язків, розроблених у розділі 2.2.

Щоб оптимізувати процес анотації та мінімізувати трудомісткі зусилля, цей проект спочатку використовуватиме ChatGPT-3 для позначення сутностей і зв’язків між ними. Згодом анотований вміст пройде процес перевірки вручну. Перегляд вручну буде з використанням платформи doccano [25].

Doccano - це платформа з відкритим вихідним кодом для розмітки даних, призначена для фахівців з машинного навчання. Вона надає інструменти для виконання різних типів завдань з розмітки з різними форматами даних.

Основні можливості Doccano:

- Розмітка тексту: Doccano дозволяє розмічати текст для таких завдань, як класифікація тексту, виділення ключових слів, розпізнавання іменованих сутностей та аналіз тональності.

- Розмітка послідовностей: Doccano підтримує розмітку послідовностей для завдань, таких як тегування частин мови та розпізнавання іменованих сутностей.

- Розмітка зображень: Doccano дозволяє розмічати зображення для завдань, таких як класифікація зображень, виявлення об'єктів та сегментація зображень.

Експорт даних: Розмічені дані можна експортувати в різних форматах, таких як JSON, CSV та CoNLL.

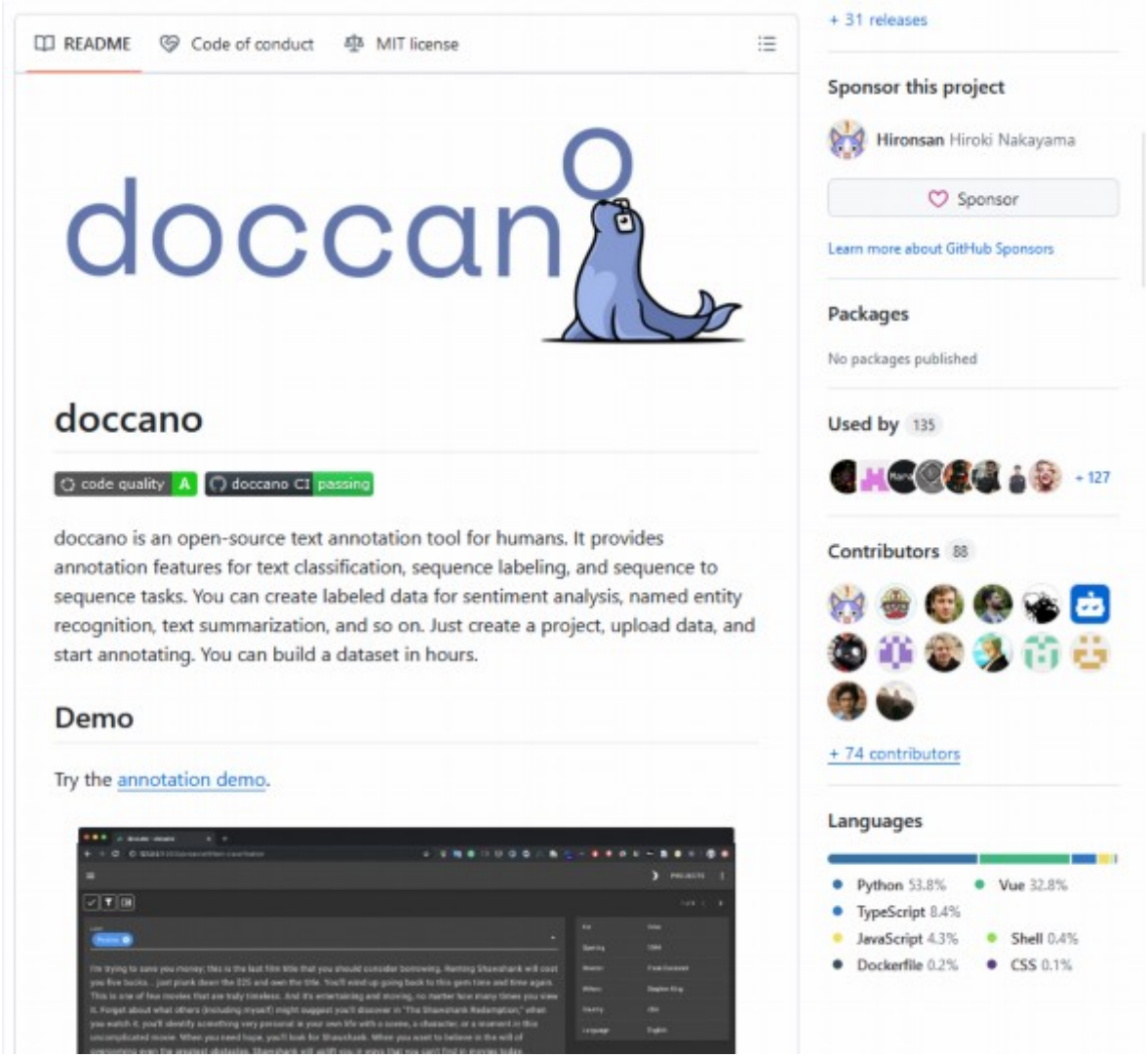

Рис. 2.7. Проект doccano на github

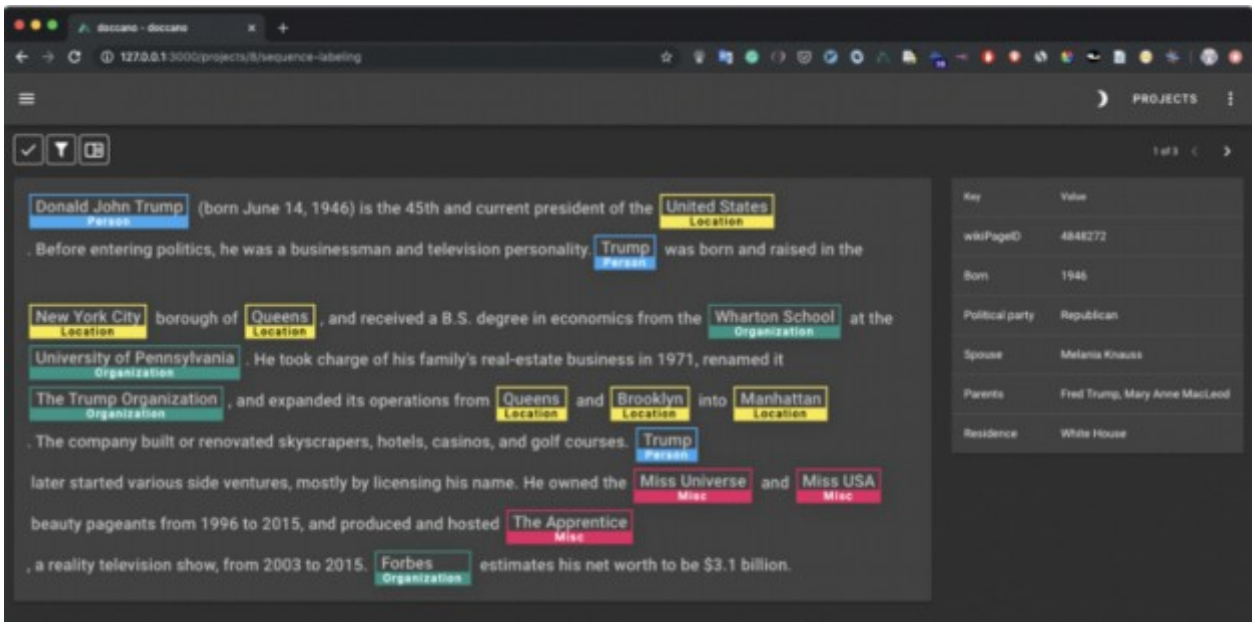

Рис. 2.8. Приклад роботи doccano

Після завершення процесу анотації платформа doccano створить два файли jsonl, які охоплюють різні деталі, такі як оригінальний текст, витягнуті об’єкти, їх відповідні розташування, мітки та зв’язки. Кожен рядок у файлі jsonl відповідає даним у наборі даних. Один із файлів призначений для наземного набору правдивих даних, а інший – для необробленого набору даних. Рисунок 2.9 зображає приклад даних у файлах jsonl.


```
{
    "id":2351,
    "text":"Implementing software modularity is a notoriously difficult task. Interoperability with a large code base written by a diverse community
            is also difficult to manage. At Eclipse, we have managed to succeed on both counts. In June 2010, the Eclipse Foundation made available its Helios
            coordinated release, with over 39 projects and 490 committers from over 40 companies working together to build upon the functionality of the base
            platform. What was the original architectural vision for Eclipse? How did it evolve? How does the architecture of an application serve to encourage
            community engagement and growth? Let's go back to the beginning.",
    "entities":[
        {"id":142060,"label":"design","start_offset":0,"end_offset":32},
        {"id":142061,"label":"structure","start_offset":38,"end_offset":64},
        {"id":142063,"label":"architecture","start_offset":277,"end_offset":303},
        {"id":142064,"label":"design","start_offset":239,"end_offset":246},
        {"id":142065,"label":"structure","start_offset":424,"end_offset":438},
        {"id":142066,"label":"structure","start_offset":277,"end_offset":303},
        {"id":142070,"label":"design","start_offset":547,"end_offset":558},
        {"id":142071,"label":"requirements","start_offset":578,"end_offset":598},
        {"id":142072,"label":"requirements","start_offset":603,"end_offset":609},
        {"id":142073,"label":"architecture","start_offset":239,"end_offset":257}],
    "relations":[{"id":1848,"from_id":142070,"to_id":142071,"type":"meet"},
    {"id":1849,"from_id":142070,"to_id":142072,"type":"meet"},
    {"id":1850,"from_id":142064,"to_id":142063,"type":"embody"},
    {"id":1852,"from_id":142073,"to_id":142066,"type":"represent"},
    {"id":1853,"from_id":142073,"to_id":142065,"type":"represent"}],
    "Comments":[]
}
```


Рис. 2.9. Приклад даних у файлах jsonl

Через довжину вихідного тексту в даних, яка перевищує оптимальну довжину для більшості моделей, у цьому дослідженні проводиться додаткова обробка обох наборів даних. Кожні дані в обох наборах даних розділені на багато підмножин. Текст кожного даного згортається в одне речення, яке включає принаймні одну трійку. На рисунку 2.10 зображено приклад даних у наборах даних.

Рис. 2.10. Приклад даних у наборах даних

Елемент 'text' зберігає оригінальний текст, тоді як елемент 'spo_list' зберігає витягнуті трійки. У словнику 'spo_list' елемент 'predicate' зберігає зв'язки між сутностями, тоді як елементи 'object' і 'subject' зберігають витягнуті сутності, діючи як ціль і джерело зв'язків відповідно. Елемент 'object_type' зберігає мітки об'єктів, а елемент 'subject_type' зберігає мітки суб'єктів. Усі дані в наборі даних відповідають цій специфічній структурі.

Зрештою, базовий набір даних містить 732 дані, тоді як вихідний набір даних містить 2254 дані.

Необроблений набір даних піддасться процесу стратифікації, щоб забезпечити репрезентативність розподілу вибірок між різними класами. Згодом набір даних буде розділено на три підмножини: навчальні набори даних, набори даних перевірки та набори даних тестування зі співвідношенням 8:1:1. Є 1803 даних у навчальному наборі, 225 даних у перевірочному тесті та 226 даних у тестовому наборі.

2.4.3. Розширення навчального набіру даних

Завдання вилучення інформації зазвичай вимагає великого навчального набору даних. Оскільки дані 1803 здаються недостатніми для навчального набору, життєво важливо розширити навчальний набір. Для досягнення цієї мети можна використовувати метод зворотного перекладу. Оскільки комерційно доступні системи перекладу не можуть гарантувати 100% точність під час перекладу того самого речення іншими мовами, але значення зазвичай не сильно відрізняється, якщо ми перекладемо речення іншою мовою, а потім знову мовою оригіналу, ми отримаємо нове речення зі схожим значенням, але дещо іншим формулюванням. Це нове речення розглядатиметься моделлю як нові дані. За допомогою цього методу навчальний набір можна розширити.

Метод зворотного перекладу передбачає виконання наступних кроків:

- Замінити усі малі літери в кожній анотованій сутності великими, щоб вони збереглися в перекладі.

- Використати Google Translate для перекладу тексту з англійської на іншу мову та назад. У цьому проекті текст буде перекладено китайською мовою, а потім знову перекладено англійською.

- Замінити великі літери на малі.

Цей процес створить точні зразки даних, які демонструють схожість з вихідними даними, але не є ідентичними. На рисунку 2.11 показано порівняння речення, створеного за допомогою методу зворотного перекладу, з оригінальним реченням. Перше речення на рисунку 2.11 представляє

оригінальне речення, тоді як друге речення представляє нове речення, створене методом зворотного перекладу. Обидва речення містять ту саму трійку.

	entity	label	relationship	entity	label	sentence
original sentence	Eclipse	Architecture	represent	foundation	Structure	Eclipse provided a component-based platform that could serve as the foundation for building tools for developers.
new sentence	Eclipse	Architecture	represent	foundation	Structure	Eclipse provids a component-based platform that can be used as a foundation for developers to build tools.

Рис. 2.11. Порівняння даних, отриманих за допомогою методу зворотного перекладу з вихідними даними

Оскільки деякі дані є ідентичними до та після перекладу, загалом генерується лише 1628 нових фрагментів даних. Зараз у навчальному наборі 3431 фрагмент даних. Таблиця 2.1 показує кількість даних у кожному наборі даних.

Таблиця 2.1.

Кількість даних у кожному наборі даних

	ground truth dataset	training set	valiation set	testing set
the number of data	732	3431	225	226

2.4.4. Недубльовані трійки

Під час наступних етапів оцінки цього проекту необхідно використовувати трійки в наборі наземних даних як еталон для порівняння. Щоб забезпечити точність результатів, необхідно виключити повторювані трійки. Після завершення процесу дедуплікації було ідентифіковано загалом 691 чітку трійку. Таблиця 2.2 відображає кількість трійок, присутніх у кожній відповідній групі.

Таблиця 2.2.

Кількість трійок у кожній відповідній групі

Type of triples	number of triples
Architecture-represent-Structure	87
Architecture-guide-Design	63
Design-embody-Architecture	92
Design-meet-Requirements	79
Design-describe-Structure	98
Design-describe-Data model	94
Design-describe-Behaviour	3
Design-describe-User interface	94
Requirements-use-Application domain knowledge	26
Structure-implement-Implementation	6
Data model-implement-Implementation	50
Behaviour-implement-Implementation	7
User interface-implement-Implementation	35
Implementation-run-Execution platform	25
Execution platform-support-Implementation	7

Висновки до розділу

В даному розділі розглянуто ключові аспекти проєктування дизайну онтологій, створення архітектури, побудови зв'язків, вибору методології та підготовки даних. Аналіз концепцій знань в архітектурі дозволив визначити основні елементи, які потрібно враховувати при створенні онтологій. Включення таких концепцій сприяє формуванню структурованої моделі, що відображає предметну область.

Вибір мови для створення онтології є важливим етапом, який впливає на зручність використання і масштабованість моделі. Проведена перевірка дозволила підтвердити відповідність мови заданим вимогам і забезпечити можливість інтеграції з іншими системами.

Виконано процес підготовки даних:

Ідентифіковано надійні джерела даних, що забезпечують високу якість інформації для навчання.

- Виконано систематизацію процесів для формування структурованих і релевантних наборів даних.

- Використано методи доповнення даних, що сприяють поліпшенню узагальнюючої здатності моделей.

- Забезпечено унікальність і несуперечливість даних, що важливо для коректної роботи онтологій і мінімізації помилок.

Таким чином, реалізовані підходи та методології сприяють ефективному створенню онтологій, які здатні забезпечувати точне представлення знань, гнучкість і інтегрованість у складних інформаційних системах.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ МОДЕЛЕЙ ТА МЕТОДІВ ДЛЯ ПОБУДОВИ ГРАФА ЗНАНЬ

3.1. Методологія вилучення інформації. Побудова моделі

У цьому розділі описано методологію, яка використовується для вилучення інформації, показано переваги обраної моделі та представлено повний огляд її фундаментальної структури. Робочий процес цього етапу зображено на рисунку 3.1.

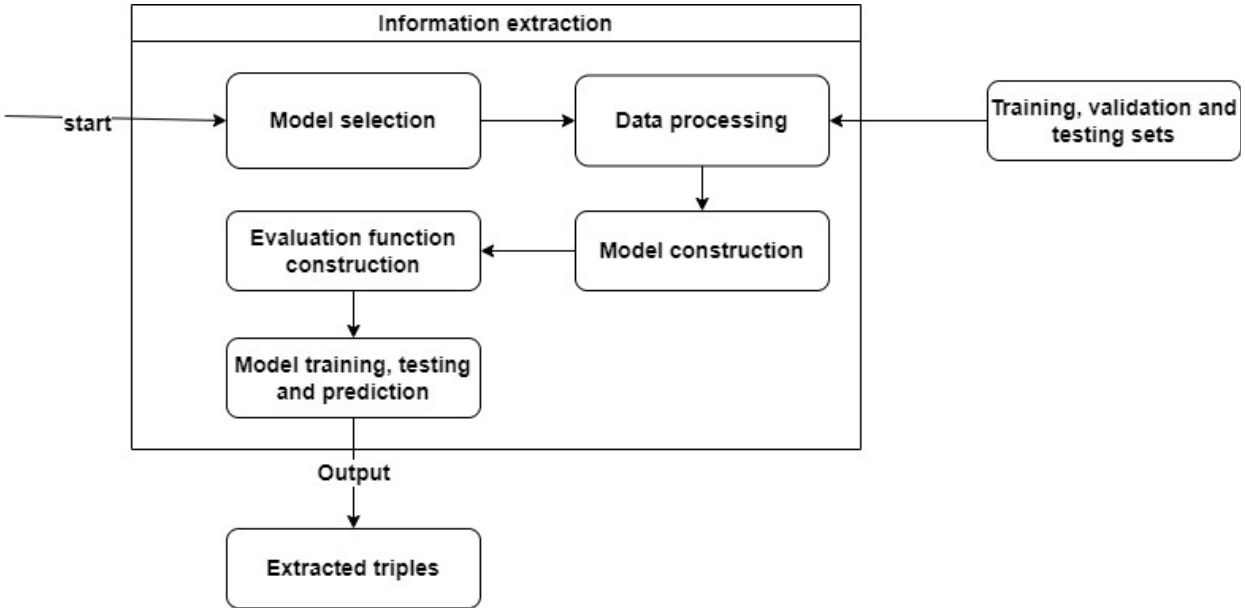

Рис. 3.1. Етапи процесу вилучення інформації

3.1.1. Вибір моделі

Щоб досягти мети отримання інформації, необхідно ретельно підібрати відповідну модель. CasRel [34], що є абревіатурою від Cascade Relation Extraction, є життєздатним варіантом. CasRel використовує низку конструкцій нейронних мереж, у тому числі рекурентні нейронні мережі (RNN), згорткові нейронні мережі (CNN) і попередньо навчені моделі на основі трансформаторів, такі як BERT [7]. Ці архітектури використовуються для ефективного захоплення контекстної інформації та залежностей у

вхідному тексті. Запропонована модель демонструє вищу ефективність порівняно з моделлю SpERT (Span-based Joint Entity and Relation Extraction) [8] і демонструє більшу простоту порівняно з моделлю UIE [20], обидві з яких є широко визнаними моделями спільного вилучення.

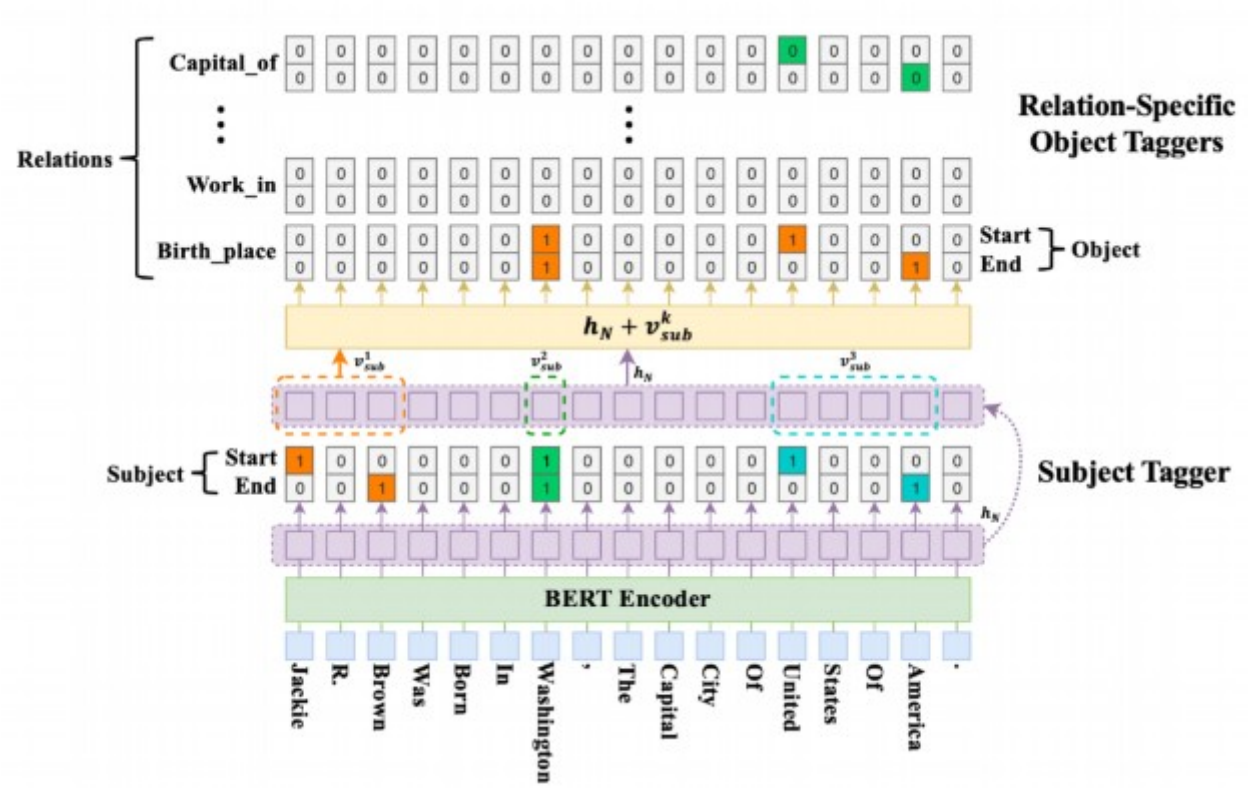

Рис. 3.2. Структура моделі CasRel

На рисунку 3.2 наведено структуру моделі [34]. Базовий рівень моделі представляє токенізований вхід, створений токенізером BERT [8]. Потім кодер BERT використовується для кодування вхідних даних, що призводить до вбудовування слів, які потім використовуються для прогнозування початкової та кінцевої позицій суб’єкта за допомогою методології двійкової класифікації. Зокрема, лінійний рівень і сигмоїдна функція активації використовуються для визначення того, чи є маркер початковим чи кінцевим маркером предметної сутності. На рисунку 3.2, модуль «Subject Tagger» представляє структури, відповідальні за прогнозування положення суб’єктів.

Результатом цього шару є випадково вибраний об’єкт із багатьох об’єктів-кандидатів, передбачених модулем «Subject Tagger».

Після отримання виводу від модуля «Subject Tagger» почне працювати модуль «Relation-Specific Object Tagger». Модуль «Relation-Specific Object Tagger» відповідає за прогнозування матриці об’єкт-відношень обраного суб’єкта.

Модель випадковим чином вибере одного суб’єкта та одразу передбачить матрицю об’єктних зв’язків для цього суб’єкта, а потім повторить кроки кілька разів. Завдяки цій ітераційній процедурі модель CasRel може передбачити сутності та зв’язки в заданому контексті.

3.1.2. Обробка даних

Основна мета обробки даних полягає в тому, щоб використовувати токенізатор BERT для кодування оригінальних речень у навчальному наборі. Відображення зсуву, маски, послідовності позицій і матриці позицій включені в кожен екземпляр даних. Щоб полегшити обробку навчального набору даних, необхідно перетворити його на вхідні дані, необхідні для моделі.

Оброблені дані складаються з 4 додаткових частин:

1. Ідентифікатори токенів: токенізовані вхідні дані речень, представлені у вигляді списку числових ідентифікаторів для кожного токена у вхідному тексті. Він зберігатиметься в оброблених даних у ключі «input_ids».

2. Маски уваги: двійкова маска, що вказує, які елементи модель повинна враховувати під час навчання та логічного висновку, а які слід ігнорувати. Він буде розташований на початку набору оброблених даних.

3. Послідовності позицій: список числових значень, що представляють позицію кожної лексеми у вхідному тексті. В оброблених даних є набір із чотирьох послідовностей, що представляють положення

голови та хвоста суб’єктів та об’єктів відповідно. Послідовності зберігаються під відповідними ключами 'head_seq', 'tail_seq', 'heads_seq' і 'tails_seq'.

4. Матриці позиції: матриця, що представляє відносну позицію кожної лексеми у вхідному реченні відносно всіх інших токенів. Модель використовує матрицю для виявлення зв’язків між токенами у вхідних даних. В оброблених даних є пара матриць, що представляють, відповідно, позицію голови та хвоста об’єктів в одному відношенні. Послідовності розміщуються під відповідними ключами "heads_mx" і "tails mx".

Усі ці компоненти використовуються для побудови остаточного вхідного тензора для моделі, який є багатовимірним масивом, що містить числові представлення вхідних токенів разом із додатковою інформацією, необхідною для вивчення зв’язків між ними. Оброблені дані будуть виглядати так, як показано на рисунку 3.3.

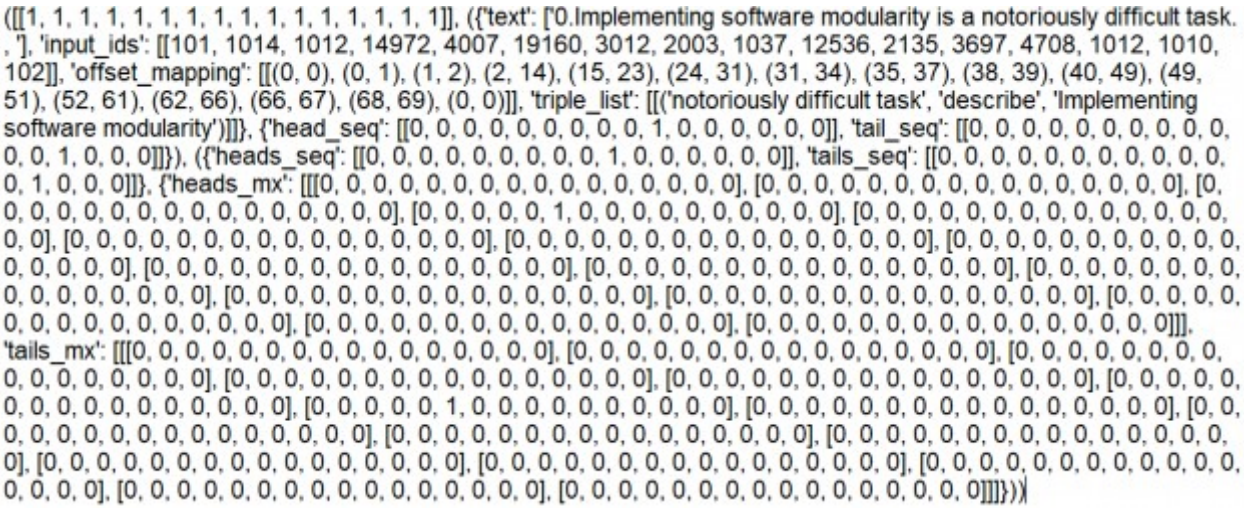

Рис. 3.3. Оброблені дані

Дані будуть використані для навчання моделі. Оскільки сама модель не розпізнає мітки суб’єктів і об’єктів, а лише виділяє трійки, цей проект інтегрує мітки об’єктів у відношення «опис», утворюючи нові зв’язки «опис структури», «опис моделі даних», «описати поведінку» та «описати інтерфейс користувача», щоб зменшити подальшу роботу з розпізнавання міток. Аналогічно, формування міток суб’єктів у відношенні реалізації для

створення нових відносин «реалізація структури», «реалізація моделі даних», «реалізація поведінки» та «реалізація інтерфейсу користувача».

3.1.3. Конструкція моделі

Трансферне навчання — це техніка, за якої модель глибокого навчання, навчена на великому наборі даних, застосовується до іншого набору даних для виконання подібного завдання. Зазвичай ця модель глибокого навчання відома як модель попереднього навчання. Використання попередньо навченої моделі як основи для розв'язування задач може істотно знизити складність завдання. Щоб полегшити навчання за перенесенням, у цьому проекті використовуватиметься попередньо навчена модель «bert-base-uncased», яку можна завантажити зі сторінки BERT GitHub.

Перш ніж модель може бути побудована, її необхідно ініціалізувати. Процедура ініціалізації складається з імпорту попередньо навченої моделі «bert-base-uncased» і подальшого заморожування її параметрів. Потім буде визначено чотири лінійних шари, причому два з лінійних шарів відповідають початковим і кінцевим положенням суб'єктів, а інші два відповідають початковим і кінцевим положенням об'єктів. Враховуючи, що матриця об'єктних зв'язків є двовимірною структурою даних, необхідна кількість зв'язків для двох рівнів – два.

Потім модель передбачить початкове та кінцеве положення суб'єктів. Цього можна досягти шляхом визначення функції, яка використовує токенізатор BERT для перетворення токенів слів у 768-вимірні вбудовування. Потім модель використовуватиме лінійний шар, за яким слідує сигмоїдна функція активації, щоб передбачити, чи токен представляє початок або кінець предмета. Оскільки модель повинна незалежно передбачати початок і кінець суб'єкта, лінійна шарово-сигмоїдна структура активації буде використана двічі. Модель створить 2 послідовності, що містять передбачений початок і кінець усіх суб'єктів відповідно.

Функції втрат важливі в області машинного навчання, оскільки вони забезпечують кількісну міру ефективності моделі по відношенню до конкретної роботи. Фундаментальною метою функції втрат є кількісне вимірювання невідповідності або неточності між очікуваним результатом моделі та об'єктивними значеннями або основною істинністю. З іншого боку, функції втрат відіграють вирішальну роль у процесі оптимізації, оскільки вони пропонують кількісну оцінку ефективності моделі. Основною метою навчання моделі машинного навчання є мінімізація функції втрат, отже мінімізація розбіжності між прогнозованими та цільовими значеннями. Крім того, шляхом обчислення втрат на окремому наборі даних перевірки або тестування, використання функції втрат дає нам змогу оцінити ефективність моделі та провести порівняння з альтернативними моделями чи базовими лініями. Загалом, нижчі значення функції втрат означають кращу продуктивність моделі.

Враховуючи, що прогнозування положення голови чи хвоста є завданням двійкової класифікації, бінарні втрати крос-ентропії є доречними. Загальні втрати повинні дорівнювати сумі втрат суб'єкта, помножених на їх відповідні ваги, і втрат матриці об'єктних зв'язків, помножених на їхні відповідні ваги. Для кожного виду втрат L,

$$L = \frac{1}{N}\sum_{i} -[y_i ln(p_i) + (1-y_i) ln(1-p_i)]$$

Так, що N представляє кількість зразків;

y_i представляє мітку для числа і зразок;

y_i = 0 або y_i = 1;

pi означає ймовірність того, що прогноз вибірки є позитивним.

Загальні втрати становитимуть:

$$Loss = L_{sb} \times W_s + L_{se} \times W_s + L_{ob} \times W_o + L_{oe} \times W_o$$

Де Ws являє собою вагу суб'єктів;

W_0 представляє вагу матриць об'єктного відношення;

L_{sb} являє собою втрату передбачення початку суб'єктів;

L_{se} означає втрату передбачення закінчення;

L_{ob} являє собою втрату передбачення початку матриць об'єктного відношення;

L_{oe} представляє втрату передбачення закінчення матриць об’єктних відношень.

3.2. Побудова функції оцінювання та навчання моделі

Модель передбачить можливість того, що слово в реченні є правильним підметом або об’єктом, а можливість відношення є правильним відношенням між підметом і об’єктом. Тому має бути встановлено поріг. Якщо прогнозована ймовірність більша за порогове значення, вона вважається 1, інакше – 0. Таке завдання можна розглядати як завдання бінарної класифікації. Повторіть процес прогнозування, модель може вивести трійки сутність-зв’язок-сутність у реченнях.

Оскільки передбачення трійок можна розглядати як завдання двійкової класифікації, можна знайти істинно позитивний (TP – true positive), істинно негативний (TN - true negative), хибно позитивний (FP - false positive) і хибно негативний (FN - false negative).

Однак TP, TN, FP і FN не є простими для прогнозування моделі спільного вилучення. Щоб полегшити завдання, буде використано деякі інші характеристики результатів. Функція оцінки підрахує кількість правильно передбачених триплетів, які можна побачити як TP; загальна кількість прогнозованих триплетів, яку можна розглядати як TP плюс FP; і загальна кількість триплетів у наборі правдивих даних, яку можна розглядати як TP плюс TN. Рисунок 3.4 відображає цей вид зв’язку.

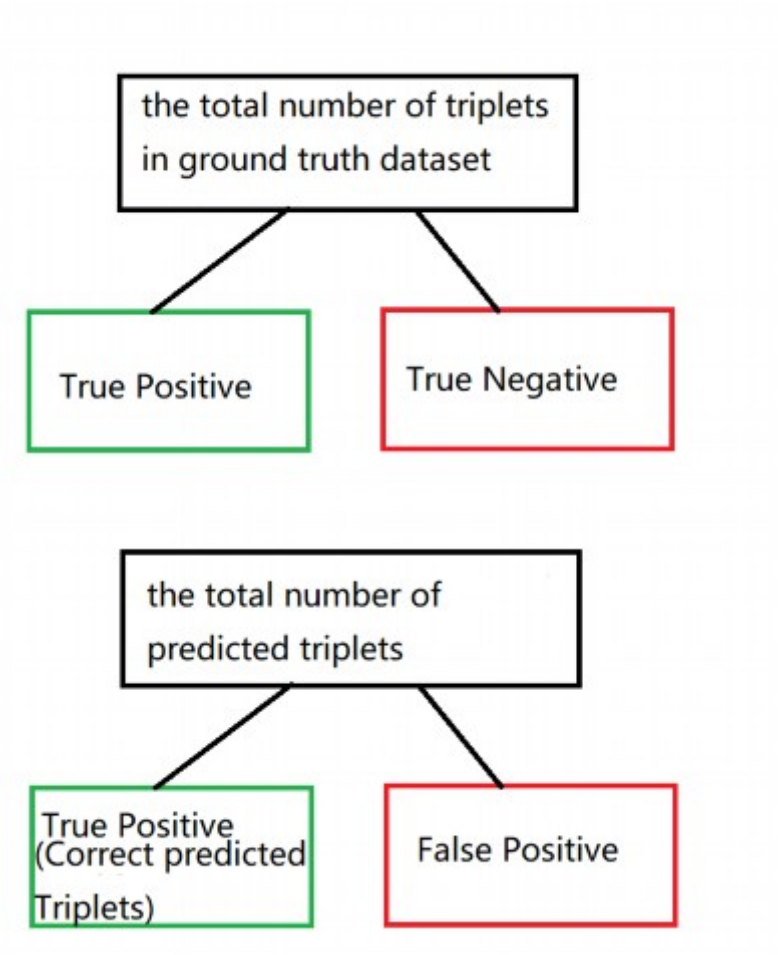

Рис. 3.4 Співвідношення між прогнозованими результатами, даними в наземному наборі правдивих даних і даними TP, FP і TN

Нижче наведено формули для точності, запам’ятовування та оцінки F1. Точність визначається як співвідношення,

$$P = \frac{TP}{TP + FP} = \frac{CT}{PT}$$

TP - це істинно позитивний результат;

FP - це хибно позитивний результат;

CT - загальна кількість правильних трійок;

PT - загальна кількість передбачених трійок.

Показник повноти визначається як відношення

$$R = \frac{TP}{TP + TN} = \frac{CT}{GT}$$

таким чином, що:

TP - це істинно позитивний результат;

TN - це істинно негативний результат;

CT - загальна кількість правильних трійок;

GT - загальна кількість трійок у еталонній відповіді.

А оцінка F1 визначається точністю P та повнотою R:

$$F1 = \frac{2 \times P \times R}{(P + R)}$$

Оцінка ефективності всього етапу вилучення інформації може бути проведена за допомогою точності, запам’ятовування та оцінки F1. Вищі бали свідчать про найкращі досягнення.

3.2.1. Навчання моделі, тестування та прогнозування

Враховуючи, що набір даних є відносно малим у порівнянні з типовими завданнями обробки природної мови, які зазвичай включають понад 10 000 зразків, кількість шарів у моделі було зменшено, щоб підвищити її простоту. Розмір партії та значення епохи становлять 10 і 50 відповідно. Швидкість навчання встановлено на 10^{-5} . Вага негативних зразків становить 0,3, а вага суб’єкта – 3. Модель автоматично записує найкращу модель для подальшого використання.

У процесі навчання втрати моделі постійно зменшуються, що свідчить про те, що модель навчається. Ідеальна крива втрат має монотонно спадний характер, що свідчить про те, що модель постійно покращує свою точність. Однак, на практиці криві втрат можуть мати різні форми, які можуть вказувати на проблеми з навчанням, такі як перенавчання або недостатнє

навчання На рисунку 3.5 зображено криву втрат, яка чітко показує, що втрати наближаються до нуля до кінця процесу навчання.

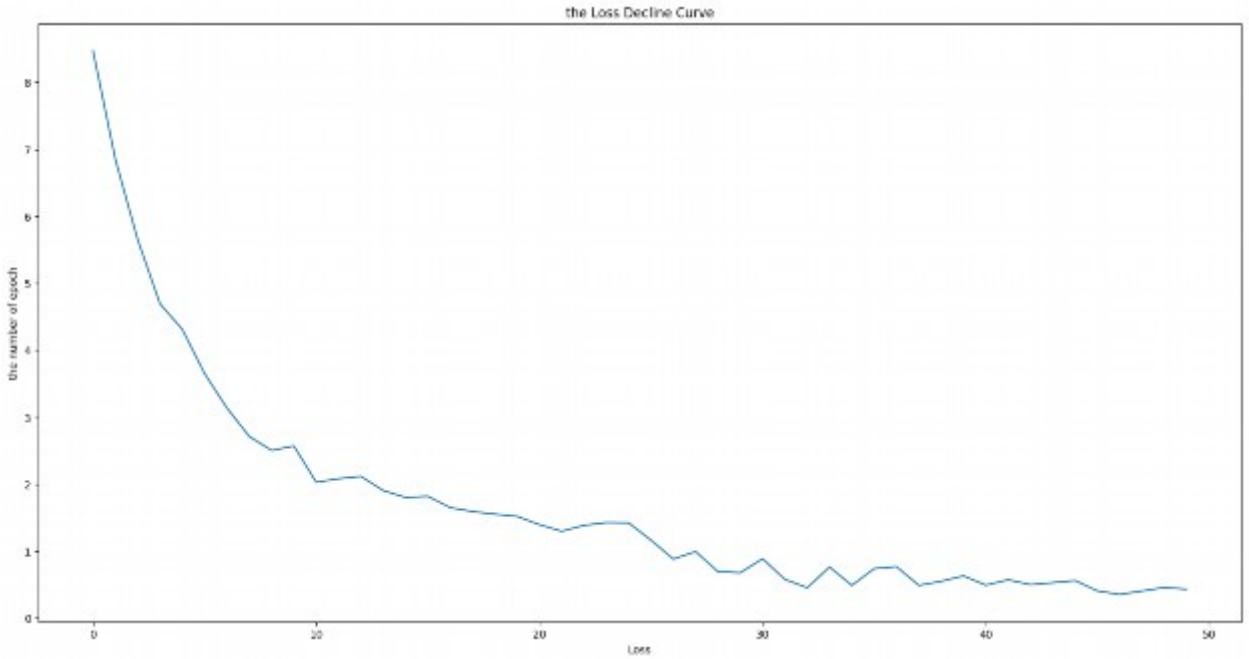

Рис.3.5. Крива втрат

У таблиці 3.1 зафіксовано найкращі показники точності, повноти та F1 для навчального набору, а також точність, повноту та F1 для тестового набору.

Таблиця 3.1.

Точність, запам’ятовування та оцінка F1

name	precision	recall	F1
training set	0.652	0.750	0.696
testing set	0.554	0.419	0.477

Оцінки вказують на те, що модель добре працює з навчальними даними, але дещо погано з тестовими даними, що вказує на переобладнання. Надмірна підгонка — це явище в машинному навчанні, коли модель стає настільки тісно адаптованою до своїх навчальних даних, що неадекватно працює на нових, невидимих даних. Замість вивчення загальних шаблонів або зв’язків, які можна застосувати до інших екземплярів даних, модель

запам’ятовує навчальні дані. Коли модель переобладнується, вона фіксує випадкові коливання або шум у навчальних даних, яких немає в базовому істинному зв’язку. Це може статися, якщо модель надто складна або якщо навчальні дані недостатні або не репрезентативні для всієї сукупності. Для цього проекту навчальний набір містить лише 1715 даних, що трохи недостатньо для завдання NLP. Але це явище несерйозне, тому я поки що залишу його таким і додам більше даних до навчального набору, щоб вирішити його в майбутньому.

Застосуємо навчену модель до документації Eclipse, внаслідок чого буде витягнуто 635 трійок. Ці трійки будуть імпортовані в Protege як екземпляри онтології та чекатимуть перевірки.

3.3. Побудова графу знань

У цьому розділі представлено методологію зберігання графів знань, показано обрану систему керування базою даних графів. Робочий процес цього етапу зображено на рисунку 3.6.

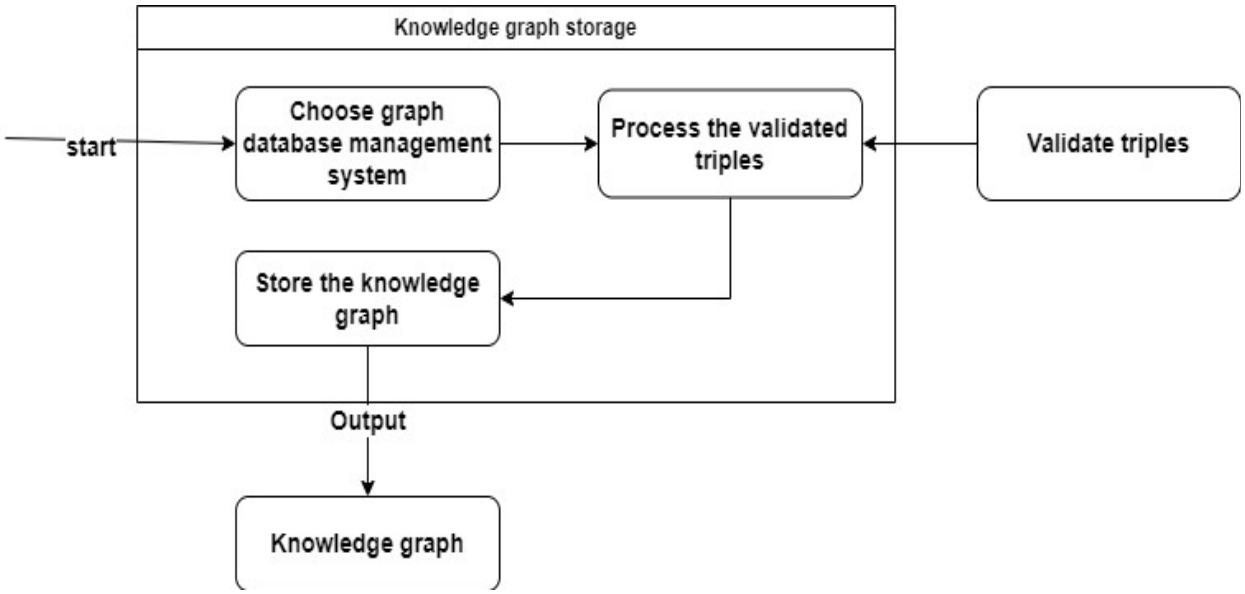

Рис. 3.6. Робочий процес етапу зберігання графа знань

3.3.1. Вибір Neo4j як систему керування графовою базою даних

Neo4j - це система керування базами даних (СКБД) з відкритим кодом, яка належить до класу NoSQL баз даних. Її головна особливість полягає в тому, що вона використовує графову модель даних для зберігання та обробки інформації.Neo4j - це система керування графовими базами даних, спеціально розроблена для ефективного зберігання, запитів та аналізу великих і складних взаємопов'язаних мереж даних. Використання цього продукту є простим. Отримавши визнаний ідентифікатор, користувачі отримують можливість входу в систему. Він широко застосовується в галузі керування графовими даними, яка зосереджена на ефективному управлінні даними з щільною мережею взаємозв'язків.

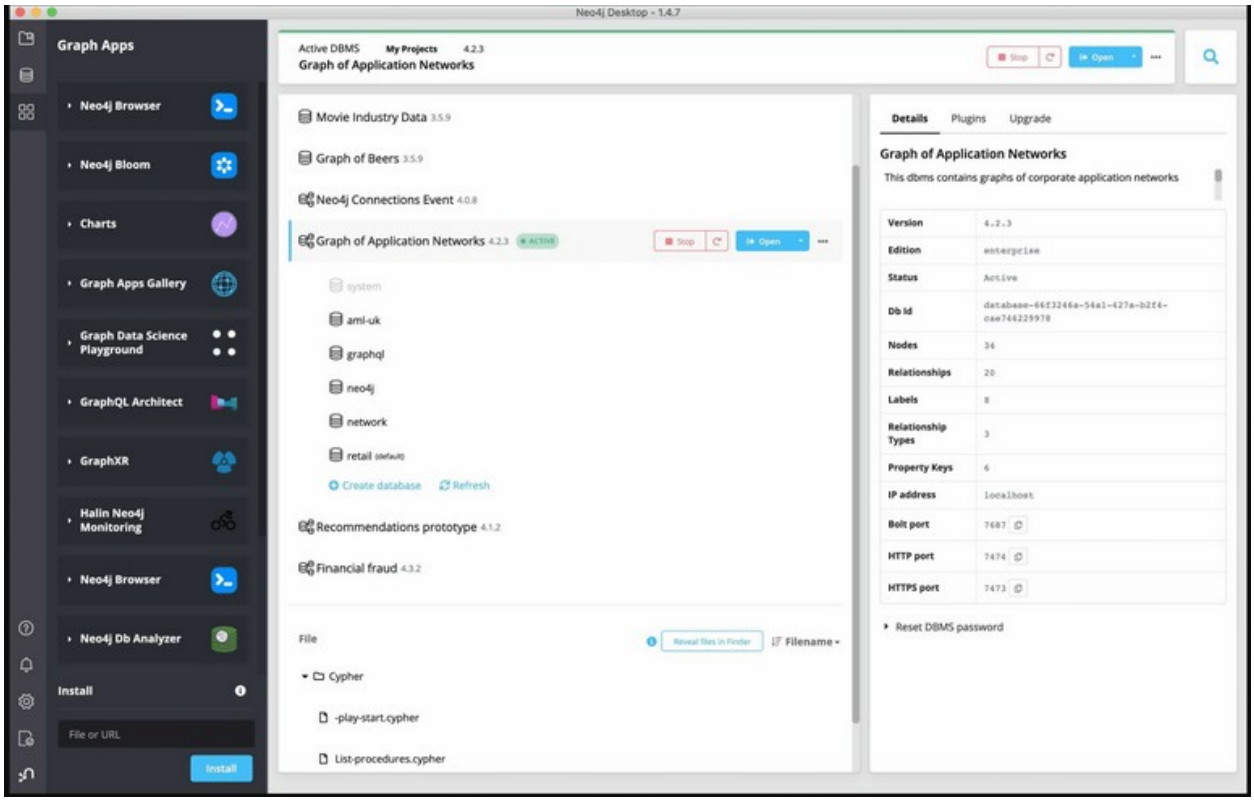

Рис. 3.7. Десктопна версія Neo4j

Швидкий перехід між зв'язками між сутностями в Neo4j робить його добре придатним для застосувань, які потребують аналізу в реальному часі та візуалізації великих мереж даних. Завдяки графовій базі даних, спеціально

розробленій для навігації зв'язками між сутностями, Neo4j стає дуже підходящим варіантом для створення та зберігання графа знань.

3.3.2. Обробка перевірених даних

Для того, щоб імпортувати трійки в граф знань, необхідно упорядкувати їх у зв'язні та добре структуровані текстові матеріали. Проект передбачає зберігання перевірених трійок у файлі Excel. Файл містить кількість очікуваних трійок, дві сутності, що беруть участь у трійках, разом з їх відповідними мітками, та зв'язок, який існує між цими двома сутностями. На рисунку 3.8 показано приклад оброблених результатів передбачення.

id	entity1	label1	relations	entity2	label2
0	open source developer	['design']	describe	altruistic person	['behaviour']
1	open source developer	['design']	describe	fixing bugs	['behaviour']
2	open source developer	['design']	describe	implementing fantastic new features	['behaviour']
3	Eclipse project	['design']	describe	initial code	['design', 'structure']
4	initial code	['design', 'structure']	describe	VisualAge for Java	['structure']
5	IBM	['design', 'requirements']	describe	Object Technology International (OTI)	['structure']

Рис.3.8. Приклад оброблених результатів прогнозування

3.3.3. Зберігання графу знань

У Neo4j дані представлені за допомогою вузлів і зв’язків. У контексті цього дослідження важливо зазначити, що вузли використовуються для представлення сутностей, тоді як зв’язки використовуються для опису зв’язків, які існують між цими сутностями. Використання вузлів і зв’язків у Neo4j полегшує зображення складних зв’язків і залежностей, завдання, яке виявляється складним при використанні традиційних реляційних баз даних. Витягнуті сутності використовуватимуться як імена для вузлів, що містять інформацію. З іншого боку, мітки сутностей будуть мітками вузлів. Зрештою, відносини між сутностями будуть відносинами між вузлами.

Модуль py2neo надає інтерфейс Python, який дозволяє маніпулювати графіками, створеними Neo4j. Використовуючи цю бібліотеку, користувачі можуть зручно вставляти витягнуті трійки в Neo4j.

Згенерований граф знань містить 730 вузлів і 588 зв’язків. Неповне відображення зворотних вузлів пояснюється конфігурацією параметра Initial Node Display. Максимальна кількість вузлів, які можна відобразити, становить 300, як це визначено параметром Initial Node Display у Neo4j. Надане візуальне представлення, як зображено на рисунку 3.9 ілюструє обмежений розділ графа знань.

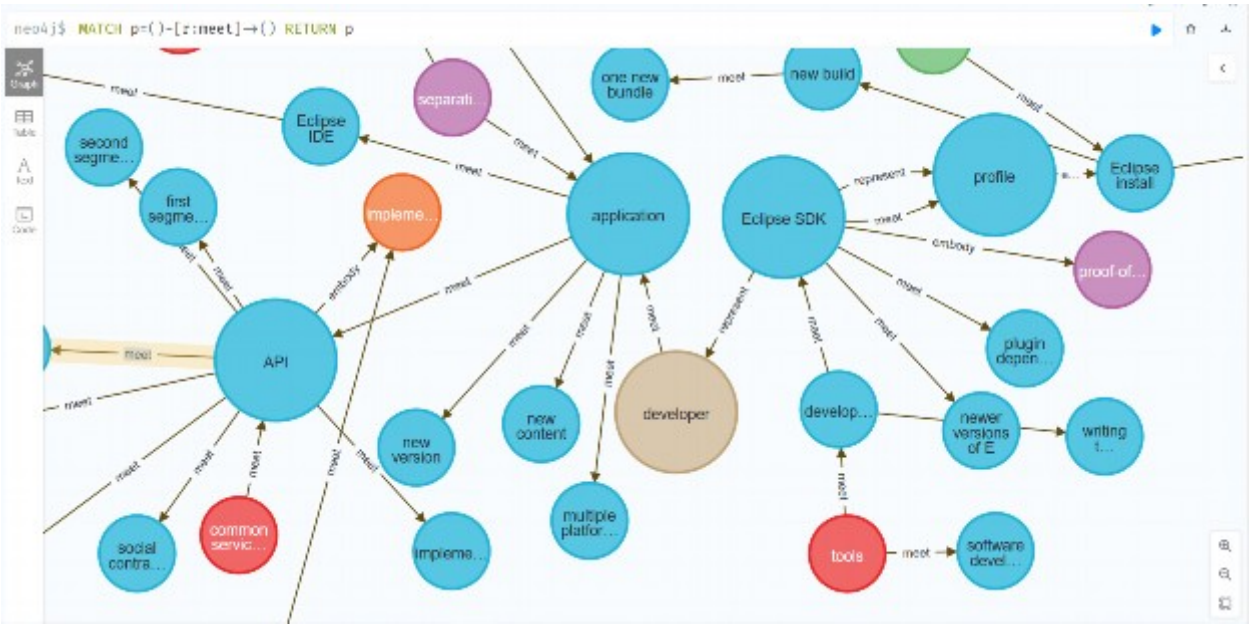

Рис. 3.9. Вигляд певної частини графа знань

3.4. Критерії та оцінка графу знань

У цьому розділі представлено два критерії для оцінки графа знань. Описано техніку, що використовується для оцінки точності та повноти графа знань і розглянуто деякі види результатів помилок.

3.4.1. Оцінки точності і повноти

Основним визначальним фактором для оцінювання графіка знань є точність і повнота. Проводячи порівняння між результатами, отриманими шляхом запиту до графа знань, і даними в основній істині, можна оцінити

точність і повноту графа знань. Відповідно, цей проект має 15 різних видів трійок сутність-зв'язок-сутність. Таблиця 3.2 перелічує всі 15 видів трійок.

Таблиця 3.2.

Представлення 15 різних видів трійок

category 1	relationship	category 2
Architecture	represent	Structure
Architecture	guide	Design
Design	embody	Architecture
Design	meet	Requirements
Design	describe	Structure
Design	describe	Data model
Design	describe	Behaviour
Design	describe	User interface
Requirements	use	Application domain knowledge
Structure	implement	Implementation
Data model	implement	Implementation
Behaviour	implement	Implementation
User interface	implement	Implementation
Implementation	run	Execution platform
Execution platform	support	Implementation

Cypher, декларативна мова запитів, запропонована Neo4j, дозволяє отримувати та маніпулювати даними, що зберігаються в базі даних графів. Мова програмування Cypher відома своїми потужними можливостями та універсальністю в обробці складних запитів і операцій над даними графів. Він має зручний синтаксис, який легко зрозуміти та реалізувати. Враховуючи обмежену кількість потрійних типів, загалом буде сформовано 15 запитів, які відповідають кожному потрійному типу. Запити мають бути такими:

```
MATCH (a:entity label)-[:relationship]→(b:entity label) RETURN a,b
```

У «мітці сутності» та «відношенні» заповнюють мітки сутності та зв'язок трійки, яку потрібно запитати.

Точність графіка знань можна визначити за часткою правильних трійок до загальної кількості запитуваних трійок. Формула наступна:

$$accuracy = \frac{CT}{QT}$$

де:

CT – загальна кількість правильних трійок;

QT — загальна кількість запитуваних трійок.

Рис. 3.10. Результати запиту графа знань

Повноту графа знань можна визначити за пропорцією правильних трійок до загальної кількості трійок у базовому наборі правдивих даних. Формула така:

$$comprehensiveness = \frac{CT}{GT}$$

де:

CT – загальна кількість правильних трійок;

GT — це загальна кількість триплетів істинності.

Після запиту до графа знань знайдено 563 трійки, серед яких 277 трійок є правильними. На рисунку 3.10 показано результати запиту до графу знань. Наданий малюнок відображає запитувані результати, пов’язані з 15 різними трійками. Дані відображаються у трьох стовпцях, що відображають кількість трійок у основній істині, кількість запитуваних трійок і кількість правильно запитуваних трійок. Результати впорядковані в порядку, який базується на кількості трійок у основній правді.

За допомогою результатів можна обчислити точність і повноту графа знань.

Точність:

$$accuracy = \frac{CT}{QT} = \frac{277}{563} = 0.492$$

Комплексність - це:

$$comprehensiveness = \frac{CT}{GT} = \frac{277}{691} = 0.401$$

У таблиця 3.3 наведено точність і повноту графа знань.

Таблиця 3.3.

Точність і повнота графа знань

name	accuracy	comprehensiveness
Querying knowledge graph	0.492	0.401

Ідентичний підхід можна застосувати для обчислення точності та повноти графа знань для різних видів трійок.

Таблиця 3.4.

Точність і повнота різних типів трійок

Type of triples	accuracy	comprehensiveness
Architecture-represent-Structure	0.805	0.379
Architecture-guide-Design	0.500	0.444
Design-embody-Architecture	0.734	0.512
Design-meet-Requirements	0.634	0.459
Design-describe-Structure	0.258	0.170
Design-describe-Data model	0.118	0.667
Design-describe-Behaviour	0.796	0.457
Design-describe-User interface	0.173	0.346
Requirements-use-Application domain knowledge	0.121	0.667
Structure-implement-Implementation	0.327	0.320
Data model-implement-Implementation	0.00	0.00
Behaviour-implement-Implementation	0.484	0.429
User interface-implement-Implementation	0.00	0.00
Implementation-run-Execution platform	0.548	0.680
Execution platform-support-Implementation	1.000	0.286

Оскільки кількість категорій Design-describe-Data Model, Requirements-use-Application do main knowledge, Data model-implement-Implementation та User interface-implement-implementation в рамках основної правди нижча за 10, результати демонструють надмірну залежність і брак суттєвої інформації. Якщо виключити чотири конкретні типи трійок, стає очевидним, що, за винятком категорій платформи Design-describe-User інтерфейс і Implementation-run-Execution, які дають більшу кількість трійок порівняно з основною правдою, усі інші типи є результатом у меншій кількості потрійок, ніж основна правда.

Отже, показники точності цих типів вищі, ніж їхні відповідні значення повноти. Категорія Architecture-represent-Structure має найвищу точність, яка становить 0,805, тоді як Design-describe-User interface має найнижчу точність, яка становить 0,173. Категорія платформи Implementation-run-Execution має

найвищу точність, яка становить 0,680, тоді як Design-describe-Structure має найнижчу точність, яка становить 0,170.

3.5. Аналіз деяких помилкових результатів

Виходячи зі спостережень, описаних у розділі 3.4, можна зробити висновок, що менша кількість трійок в еталонній відповіді призводить до більшої ймовірності появи помилок у графі знань.

Тим часом загальна точність цього дослідження досить низька: лише 49,2% ідентифікованих трійок вважаються правильними. Крім того, загальна кількість точно розпізнаних трійок також обмежена, що становить близько 40,9% від загальної кількості трійок. Після проведення комплексного аналізу конкретних результатів було визначено, що найпоширеніші помилки в результатах запитів можна грубо розділити на 5 груп.

3.5.1. Нерелевантна інформація

Перша категорія помилок стосується вилучення інформації, яка абсолютно не пов'язана з цим проектом. У таблиці 3.5 наведено набір вилучених трійок, які потрапляють під цей конкретний тип помилки.

Таблиця 3.5.

Вилучені трійки, які вважаються нерелевантною інформацією

entity 1	category 1	relationship	entity2	category 2
build	Design	meet	import	Requirements
ann	Design	describe	annotation	Structure
use	Design	describe	Eclipse	Behaviour
Figure 6.3	Structure	implement	clusters	Implementation
metadata	Data model	implement	requirements	Implementation

Дані, представлені в таблиці 3.5, показують, що ця конкретна група трійок надає мінімум цінної інформації, що стосується архітектурних знань. Крім того, деякі з вилучених сутностей навіть не є повним словом.

3.5.2. Дубльовані сутності

Друга категорія помилок стосується випадків, коли дві сутності в трійках виявляються однаковими. У таблиці 3.6 наведено набір вилучених трійок, які потрапляють під цей конкретний тип помилки.

Таблиця 3.6.

Наявність дублікатів сутностей у трійках

entity 1	category 1	relationship	entity2	category 2
consumers	Architecture	represent	consumers	Structure
context	Architecture	guide	context	Design
bundle	Design	embody	bundle	Architecture
filesystem	Design	describe	filesystem	Structure
services	Design	describe	services	Behaviour

Поява цієї конкретної неточності є більш поширеною в результатах. Як правило, у заданому наборі дублікатів сутностей один із двох екземплярів є правильним. У таблиці 3.7 показано відповідні правильні результати 5 прикладів з таблиці 3.6. Однак можливо, що через помилку в прогнозуванні модель робить неправильне припущення, що існує шлях, який починається з правильної сутності та закінчується нею ж. Ця помилка призводить до генерації кількох трійок, де дві сутності, які були помилково ідентифіковані, є однаковими.

Таблиця 3.7.

Правильні трійки, що відповідають неправильним трійкам у таблиці 3.6

entity 1	category 1	relationship	entity2	category 2
API	Architecture	represent	consumers	Structure
framework	Architecture	guide	context	Design
plugin	Design	embody	bundle	Architecture
filesystem	Design	describe	infrastructure issues	Structure
properties	Design	describe	services	Behaviour

Виходячи з даних, представлених у таблиці 3.7, очевидно, що будь-яка з двох сутностей, які дублюються, може потенційно вважатися правильною.

3.5.3. Зворотний порядок джерела та цілі

Одна поширена помилка полягає в точному вилученні інформації, а потім помилковому зворотному порядку джерела та цілі на обох кінцях зв'язків у трійках. У таблиці 3.8 наведено набір вилучених трійок, які потрапляють під цей конкретний тип помилки.

Таблиця 3.8.

Трійки зі зворотним порядком джерела та цілі

entity 1	category 1	relationship	entity2	category 2
pixel perfect	Design	meet	Native widget toolkits	Requirements
open source project	Design	describe	committers	Structure
trace problems	Design	describe	debugger	Behaviour
errors or warnings	Design	describe	view	User interface
Linux	Implementation	run	Early Eclipse SDKs	Execution platform

Вилучені сутності та їх зв'язки в помилкових трійках є точними, за винятком того, що вказівники на зв'язки інвертовані. Незважаючи на свої помилки, ці неточні трійки все ж надають певну інформацію.

3.5.4. Неправильна категоризація

Четвертий тип помилки стосується точності вилучених сутностей, але зв'язки між ними та відповідні мітки цих сутностей виявляються помилковими. У таблиці 3.9 наведено набір вилучених трійок, які потрапляють під цей конкретний тип помилки, а також відповідні правильні мітки.

Таблиця 3.9.

Трійки з неправильними трійками

entity 1	category 1	relationship	entity2	category 2
plugin registry	Design	describe	API	User interface
	Design	describe		Structure
native widget toolkit	Design	describe	operating system calls	Structure
	Design	describe		Behaviour
uninstallation	Implementation	run	bundles	Execution platform
	Behaviour	implement		Implementation
component model	Structure	implement	wider adoption	Implementation
	Design	describe		behaviour
java code	Design	describe	standard widget toolkit	Structure
	Data model	implement		Implementation

Категоризація сутностей у цих трійках є неправильною, що призводить до неможливості їх запиту при пошуку трійок у графі знань, які стосуються певної категорії. Отже, можна стверджувати, що ці трійки не надають достовірної чи точної інформації.

3.5.5. Неповне вилучення інформації

Остання категорія помилок стосується випадків, коли інформація вилучається не повністю. У цьому конкретному сценарії вилучені сутності мають нестачу вмісту порівняно з сутностями, присутніми в еталонній відповіді. Ця невідповідність особливо поширена в трійках "Дизайн-відповідає-Вимогам", оскільки сутності в цьому типі трійок, як правило, мають значну довжину. У таблиці 3.10 наведено набір вилучених трійок, які потрапляють під цей конкретний тип помилки.

Таблиця 3.10.

Неповні трійки

entity 1	category 1	relationship	entity2	category 2
workspace	Architecture	represent	files	Structure
filesystems	Design	embody	fragments	Architecture
corporations	Design	meet	Linux kernel	Requirements
framework	Design	meet	debuggers	Requirements
plugin	behaviour	implement	implementation	Implementation

Правильні відповідні трійки:

1. робоча область - представляє - колекцію файлів та метаданих
2. файлові системи Linux та Windows - втілюють - фрагменти
3. корпорації - відповідають - внеску в ядро Linux
4. фреймворк - відповідає - мовно-специфічним налагоджувачам
5. плагін - реалізує - приватні деталі реалізації

Ці неточні трійки надають обмежену кількість інформації, але вони також можуть вводити в оману, збільшуючи ймовірність того, що читач зіткнеться з помилковою інформацією.

Крім того, варто зазначити, що існують додаткові категорії помилок, включаючи випадки, коли записи містять зайву інформацію. Однак через їх нечасте виникнення ці помилки можна класифікувати як спорадичні, і вони не будуть детально розглянуті.

Висновки до розділу

У третьому розділі представлено процес реалізації моделей та методів для побудови графу знань, що включає всі етапи — від вилучення інформації до оцінки результатів.

Методологія вилучення інформації передбачала ретельний аналіз вимог до графу знань, на основі чого було обрано відповідну модель. Процес обробки даних включав очищення, нормалізацію та підготовку, що сприяло зменшенню шумів і забезпечило якісне формування базових структур. Конструкція моделі враховувала складність джерел даних і вимоги до семантичної повноти, що дозволило створити узгоджену та ефективну модель для вилучення знань.

Навчання моделі здійснювалося за допомогою побудованої функції оцінювання, яка забезпечила адаптивність моделі та її здатність до обробки великих обсягів даних. Використання метрик точності та повноти у процесі навчання дозволило відстежувати ефективність моделі та вносити корективи для підвищення її продуктивності.

Для зберігання графу знань було обрано систему керування базами даних Neo4j, яка забезпечила високу швидкість роботи, масштабованість та можливість виконання складних запитів. Перед інтеграцією до графу всі дані було перевірено та оброблено, що забезпечило їх цілісність і несуперечливість. Розроблена структура зберігання даних забезпечила ефективний доступ до інформації для її подальшого використання.

Оцінювання графу знань здійснювалося за критеріями точності та повноти, які продемонстрували високий рівень відповідності створеної

моделі вимогам. Водночас, аналіз помилкових результатів дозволив виявити низку недоліків, таких як включення нерелевантної інформації, дублювання сутностей, зворотний порядок джерела та цілі, а також неправильна категоризація. Неповне вилучення інформації свідчить про необхідність вдосконалення алгоритмів для забезпечення більшої глибини та точності аналізу. Таким чином, реалізовані моделі та методи підтвердили свою ефективність для побудови графу знань.

ВИСНОВКИ

У магістерській роботі досліджено предметну область вилучення даних для побудови графів знань, розроблено методологію побудови онтологій, створено дизайн архітектури графу знань, реалізовано моделі та методи вилучення інформації, а також здійснено їх інтеграцію у графову базу даних.

У першому розділі проаналізовано особливості предметної області, визначено ключові аспекти процесу вилучення даних і побудови графів знань. Проведено аналіз схожих проєктів, що дозволило ідентифікувати сучасні тенденції та виклики у цій сфері. Розглянуто концепцію графів знань і особливості побудови онтологій як основи для моделювання предметної області. Детально описано процес вилучення інформації, включаючи зв’язування сутностей та вилучення відношень. Також досліджено існуючі платформи та інструменти для зберігання знань, що забезпечило обґрунтований вибір технологій для реалізації завдання.

У другому розділі розроблено архітектуру графу знань, що включає концепції знань, фреймворк онтології та зв’язків між сутностями. Вибір мови для побудови онтологій базувався на потребі забезпечити інтегрованість, масштабованість та відповідність стандартам. Проведено підготовку даних, що включала ідентифікацію джерел, створення навчальних наборів, їх розширення та виключення дубльованих трійок. Це дозволило сформувати якісну основу для побудови графу знань.

У третьому розділі реалізовано методологію вилучення інформації, яка включає вибір моделі, її навчання, а також побудову функції оцінювання для забезпечення точності вилучення даних. Побудовано граф знань із використанням Neo4j як платформи для керування графовою базою даних. Розроблено ефективні методи інтеграції перевірених даних і забезпечено зручне зберігання графу для подальшого використання. Оцінювання графу здійснювалося за допомогою метрик точності та повноти, що підтвердило відповідність моделі заданим вимогам.

У результаті аналізу помилкових результатів виявлено основні проблеми, серед яких нерелевантна інформація, дублювання сутностей, зворотний порядок зв’язків, неправильна категоризація та неповне вилучення інформації. Це дозволило окреслити напрями для вдосконалення моделей і методів, зокрема оптимізацію алгоритмів вилучення, покращення обробки даних та підвищення точності класифікації.

Таким чином, результати роботи підтверджують ефективність реалізованих підходів для побудови графів знань, проте виявлені недоліки свідчать про необхідність подальшого вдосконалення та адаптації методів до нових умов і потреб.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In The Semantic Web:6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007+ ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings, pages 722–735. Springer, 2007.
2. Berners-Lee, T., Fischetti, M., & Foreword by Dertouzos, M. (2001). Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor. Harper San Francisco.
3. Len Bass, Paul Clements, and Rick Kazman. Software architecture in practice. Addison-Wesley Professional, 2003.
4. Staab, S., & Studer, R. (Eds.). (2009). Handbook on Ontologies. Springer.
5. Domingos, P. (2015). The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World. Basic Books.
6. Russell, S., & Norvig, P. (2021). Artificial Intelligence: A Modern Approach (4th Edition). Pearson.
7. Mitchell, T. M. (1997). Machine Learning. McGraw Hill.
8. Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.
9. Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked Data: Principles and State of the Art. Semantic Web.
10. Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. OSDI.
11. Knuth, D. E. (1997). The Art of Computer Programming: Volume 1: Fundamental Algorithms. Addison-Wesley.
12. Pearl, J. (1988). Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann.
13. Prabhu, R. (2017). Graph Databases in Action. Manning Publications.

14. Gruber, T. R. (1993). A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition.
15. Hogan, A., Harth, A., Umbrich, J., et al. (2012). An Empirical Survey of Linked Data Conformance. Journal of Web Semantics.
16. Eiter, T., Ianni, G., & Krennwallner, T. (2009). Answer Set Programming for the Semantic Web. AI Magazine.
17. Miller, G. A. (1995). WordNet: A Lexical Database for English. Communications of the ACM.
18. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv.
19. Mohamed Soliman, Matthias Galster, and Paris Avgeriou. An exploratory study on architectural knowledge in issue tracking systems. In Software Architecture: 15th European Conference, ECSA 2021, Virtual Event, Sweden, September 13-17, 2021, Proceedings, pages 117–133. Springer, 2021.
20. Mohamed Soliman, Amr Rekaby Salama, Matthias Galster, Olaf Zimmermann, and Matthias Riebisch. Improving the search for architecture knowledge in online developer communities. In 2018 IEEE International Conference on Software Architecture(ICSA), pages 186–186. IEEE, 2018.
21. Mohamed Soliman, Amr Rekaby Salama, Matthias Galster, Olaf Zimmermann, and Matthias Riebisch. Improving the search for architecture knowledge in online developer communities. In 2018 IEEE International Conference on Software Architecture (ICSA), pages 186–186. IEEE, 2018.
22. Witten, I. H., & Frank, E. (2005). Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann.
23. Gärdenfors, P. (2004). Conceptual Spaces: The Geometry of Thought. MIT Press.
24. Silviu Cucerzan. Large-scale named entity disambiguation based on wikipedia data. In Proceedings of the 2007 joint conference on empirical

methods in natural language processing and computational natural language learning (EMNLP-CoNLL), pages 708–716, 2007.

25. Nickel, M., Tresp, V., & Kriegel, H. (2011). A Three-Way Model for Collective Learning on Multi-Relational Data. ICML.
26. Quillian, M. R. (1968). Semantic Memory. Semantic Information Processing.
27. Guha, R., McCool, R., & Miller, E. (2003). Semantic Search. WWW Conference Proceedings.
28. Rajaraman, A., & Ullman, J. D. (2011). Mining of Massive Datasets. Cambridge University Press.
29. Barabási, A.-L. (2016). Network Science. Cambridge University Press.
30. Kleinberg, J. (1999). Authoritative Sources in a Hyperlinked Environment. Journal of the ACM.
31. Hammond, T., Sheth, A., & Kochut, K. (2002). Semantic Web and Mining Applications. IEEE Computer Society.
32. Hartig, O., Pérez, J., & Thompson, B. (2020). Foundations of RDF Query Processing. ACM Transactions on Database Systems.
33. Suchanek, F. M., Kasneci, G., & Weikum, G. (2007). YAGO: A Core of Semantic Knowledge. WWW Conference Proceedings.
34. Sheth, A., Ramakrishnan, C., & Thomas, C. (2005). Semantics for the Semantic Web: The Implicit, the Formal, and the Powerful. International Journal on Semantic Web and Information Systems.
35. Shadbolt, N., Berners-Lee, T., & Hall, W. (2006). The Semantic Web Revisited. IEEE Intelligent Systems.
36. Dean, M., & Schreiber, G. (2004). OWL Web Ontology Language Reference. W3C Recommendation.
37. Blanco, R., & Lioma, C. (2012). Graph-Based Term Weighting for Information Retrieval. ACM Transactions on Information Systems.

38. Navigli, R., & Ponzetto, S. P. (2012). BabelNet: The Automatic Construction, Evaluation, and Application of a Wide-Coverage Multilingual Semantic Network. Artificial Intelligence.
39. Gabrilovich, E., & Markovitch, S. (2007). Computing Semantic Relatedness Using Wikipedia-Based Explicit Semantic Analysis. IJCAI.
40. Bordes, A., Usunier, N., García-Durán, A., et al. (2013). Translating Embeddings for Modeling Multi-Relational Data. NIPS.
41. Mitchell, M., & Lapata, M. (2010). Composition in Distributional Models of Semantics. Cognitive Science.
42. Richardson, M., & Domingos, P. (2006). Markov Logic Networks. Machine Learning.
43. Rusu, A., et al. (2020). Meta-Learning for Graph Neural Networks. ICLR.
44. Yih, W.-t., Chang, M.-W., Meek, C., & Pastore, J. (2015). Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base. ACL.
45. Jeen Broekstra, Arjohn Kampman, and Frank Van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In The SemanticWeb—ISWC 2002: First International SemanticWeb Conference Sardinia, Italy, June 9–12, 2002 Proceedings, pages 54–68. Springer, 2002.
46. Amy Brown and Greg Wilson. The architecture of open source applications, volume ii, volume 2. Lulu. com, 2012.
47. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.
48. Markus Eberts and Adrian Ulges. Span-based joint entity and relation extraction with transformer pre-training. arXiv preprint arXiv:1909.07755, 2019.
49. Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property

graphs. In Proceedings of the 2018 international conference on management of data, pages 1433–1445, 2018.

50. Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: anowl 2 reasoner. Journal of automated reasoning, 53:245–269, 2014.
51. Nicola Guarino, Daniel Oberle, and Steffen Staab. What is an ontology? Handbook on ontologies, pages 1–17, 2009.
52. Junheng Hao, Muhao Chen, Wenchao Yu, Yizhou Sun, and Wei Wang. Universal representation learning of knowledge bases by jointly embedding instances and ontological concepts. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining, KDD ’19, page 1709–1719, New York, NY, USA, 2019.