Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Автоматизоване наповнення графів знань з неструктурованих текстів

Текстова частина до кваліфікаційної роботи
за спеціальністю 122 «Комп’ютерні науки»

Керівник кваліфікаційної роботи
Старший викладач Андрощук М.В.
_____ (підпис)
“____” _____ 2025 р.

Виконав студент
Анісімов Є.О.
_____ (підпис)
“____” _____ 2025.р.

Київ 2025

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики Факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

доцент, кандидат наук

С.С. Гороховський

_____(підпис)

“____”_____ 2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на кваліфікаційну роботу

студенту Анісімову Євгену Олександровичу факультету інформатики 4-го курсу
Тема: Автоматизоване наповнення графів знань з неструктурованих текстів

Зміст ТЧ до кваліфікаційної роботи:

1. Історичний огляд та сучасний стан баз знань
2. Теоретичні основи автоматизованого наповнення графів знань та Graph Question Answering
3. Проєктування та реалізація системи
4. Тестування та оцінювання ефективності системи
5. Висновки

Дата видачі “___” _____ 2025 р.

Керівник _____(підпис)

Завдання отримав _____(підпис)

Тема: Автоматизоване наповнення графів знань з неструктурованих текстів

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	27.09.2024	
2.	Пошук тематичної літератури	09.11.2024	
3.	Ознайомлення з літературою	31.12.2024	
4.	Написання текстової частини кваліфікаційної роботи	24.02.2025	
5.	Реалізація практичної частини	15.03.2025	
6.	Тестування практичної частини	17.03.2025	
7.	Внесення змін до кваліфікаційної роботи відповідно до зауважень наукового керівника	27.04.2025	
8.	Створення презентації	12.05.2025	
9.	Попередній захист роботи	26.05.2025	
10.	Захист кваліфікаційної роботи	05.06.2025	

Студент Анісімов Є.О.

“____” _____

Керівник Андрощук М.В.

“____” _____

Зміст

Анотація

У кваліфікаційній роботі досліджено проблему автоматизованого наповнення графів знань з неструктурованих текстів та реалізовано прототип системи, що поєднує сучасні методи обробки природної мови та технології відповідей на запитання за допомогою графів. Запропонований сценарій включає: динамічне визначення типів сутностей за допомогою великих мовних моделей, розпізнавання іменованих сутностей, генеративне вилучення відношень, збереження фактів у графовій базі даних Neo4j та створення векторного індексу для описів відношень. Для пошуку релевантного контексту застосовано комбінацію семантичного пошуку методом k-найближчих сусідів та алгоритму Personalized PageRank, після чого велика мовна модель генерує відповідь на запитання користувача природною мовою. Практична цінність полягає у можливості швидше будувати графи знань на основі текстових корпусів, придатні для виконання запитів, що є актуальним для корпоративних систем управління знаннями, аналітичних платформ та інтелектуальних асистентів.

Вступ

В сучасному інформаційному суспільстві спостерігається експоненційне зростання обсягів даних, причому переважна більшість їх генерується у вигляді неструктурованих текстів: новинні статті, наукові публікації, звіти компаній, дописи в соціальних мережах тощо. Цей величезний масив інформації містить цінні знання, однак їх вилучення, структурування та ефективне використання залишається складним завданням. Для подолання цієї проблеми ключовим інструментом стають бази знань, зокрема представлені у вигляді графів знань, які моделюють факти про світ як сукупність сутностей та відношень між ними.

Проте ручне наповнення таких баз є надзвичайно трудомістким, дорогим і нездатним встигати за темпами появи нової інформації. Тому завдання автоматизованого наповнення графів знань з неструктурованих текстів набуває виняткової актуальності. Воно лежить на перетині обробки природної мови (Natural Language Processing, NLP), інформаційного вилучення (Information Extraction, IE) та технологій представлення знань. Розробка ефективних методів та інструментів для автоматичного перетворення текстових даних у структуровані знання є критично важливою для розвитку інтелектуальних систем, покращення пошуку інформації та підтримки прийняття рішень у різноманітних галузях.

Наукове та практичне значення роботи полягає у дослідженні та інтеграції сучасних підходів NLP та IE для побудови динамічних графів знань, а також у розробці механізмів взаємодії з цими графами за допомогою запитів природною мовою (Graph Question Answering, GQA). Практична

цінність визначається можливістю створення системи, здатної автоматично обробляти великі обсяги текстової інформації, формувати на їх основі граф знань та надавати користувачам доступ до цих знань у зручній формі запитань-відповідей. Це може знайти застосування в корпоративних системах управління знаннями, аналітичних платформах, освітніх ресурсах та інтелектуальних асистентах.

Предметом дослідження є методи, алгоритми та архітектурні рішення для вилучення сутностей і відношень з текстів, їх представлення у вигляді графа знань та реалізації механізму відповідей на запитання до цього графа.

Метою кваліфікаційної роботи є розробка, реалізація та оцінювання програмної системи для автоматизованого наповнення графа знань з неструктурованих текстів з підтримкою функціональності GQA.

Структура роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, заключних висновків та списку використаних джерел. У першому розділі розглянуто історію та еволюцію баз знань. Другий розділ присвячено теоретичним основам NLP, IE, автоматизованого наповнення графа знань та GQA. У третьому розділі описано проєктування та реалізацію розробленої системи. Четвертий розділ містить опис тестування, аналіз отриманих результатів та їх оцінку. У висновках підсумовано результати роботи та окреслено можливі напрями подальших досліджень.

Розділ 1. Історичний огляд та сучасний стан баз знань

1.1 Експертні системи та перші бази знань

Експертні системи стали одним із визначальних етапів у розвитку технологій баз знань. Їх основна ідея полягала у моделюванні процесу мислення спеціалістів у певній вузькій галузі з метою ухвалення рішень на основі накопиченого досвіду. Головним компонентом таких систем була база знань, яку формували вручну: розробники тісно співпрацювали з експертами, щоб описати факти, закономірності та правила в формі «якщо-то». Відповідно, на відміну від звичайних програм, логіка виведення результатів тут будувалася не лише на алгоритмах пошуку рішень, а й на доменних знаннях, які потрібно було акумулювати в системі [1].

Одним із найвідоміших прикладів є MYCIN, створена у 1970-х роках у Стенфордському університеті для діагностики бактеріальних інфекцій крові та вибору відповідної антибіотикотерапії. Щоб реалізувати логіку прийняття рішень, у MYCIN сформували базу знань із приблизно 600 правил, що описували зв’язки між симптомами, збудниками захворювань і варіантами лікування. Для пошуку рішення система використовувала метод зворотного ланцюга, аналізуючи правила з кінця, щоб визначити, які саме з них застосувати для поточної симптоматики пацієнта [3].

Процес створення експертних систем був трудомістким. Щоб додати нову діагностичну гіпотезу чи розширити базу знань новими фактами, розробники та експерти мали переписувати частину правил чи додавати нові вручну. Це ускладнювало масштабування подібних рішень на інші галузі або великі обсяги знань. Саме тому перші експертні системи здебільшого

обмежувалися вузькими предметними областями, де кількість правил залишалася керованою.

Іншим прикладом є DENDRAL – система, яка допомагала хімікам визначати структури органічних сполук за мас-спектрами. Як і MYCIN, DENDRAL базувалася на великій кількості ручного кодування правил та фактів, зібраних від фахівців. Незважаючи на ефективну роботу у вузькій галузі, цей підхід не був гнучким і не піддавався легкому масштабуванню. Проте саме в таких системах сформувалися концепції логічного виведення, правил типу «якщо-то» та факторів невизначеності, які досі лежать в основі численних сучасних рішень [3].

Загалом у ранніх експертних системах переважали так звані rule-based підходи. Однак у міру розвитку штучного інтелекту з’явилися й інші парадигми для опису та зберігання знань, наприклад:

- Експертні системи на основі фреймів (Frame-Based Expert Systems)
- Експертні системи з нечіткою логікою (Fuzzy Logic-Based Expert Systems)
- Експертні системи на основі нейронних мереж (Expert Systems Based on Neural Networks) [3]

Перехід до Frame-Based систем у 1980-х роках дозволив описувати складнішу інформацію про об’єкти. Однак і в цьому випадку розширення бази знань все одно було надзвичайно трудомістким. Згодом з’явилися нечіткі та нейронно-орієнтовані експертні системи, що частково вирішували

проблему жорсткої логіки та ручного введення правил. Проте для всіх цих систем лишалася ключова проблема: як швидко та безпомилково отримувати нові знання з реальних даних без повної залежності від людини [2].

Це стимулювало дослідників до пошуку нових підходів, здатних більш гнучко та масштабовано представляти знання. Подальший розвиток технологій збереження знань поступово привів до появи семантичних мереж та онтологій, які розглядаються у наступному підрозділі.

1.2. Еволюція баз знань

Для подолання зазначених вище недоліків експертних систем почали розвиватися альтернативні підходи представлення знань, ключовими з яких стали семантичні мережі та онтології. Семантичні мережі представляли знання як граф (рис. 1.1), що складався з вузлів, які відповідали певним поняттям, і зв’язків, які описували відносини між цими поняттями. Цей підхід забезпечував наочність представлення знань і дозволяв проводити базове логічне виведення [5]. Однак такі мережі мали недоліки, зокрема нечіткість визначення зв’язків і труднощі зі зростанням їхніх розмірів, що значно ускладнювало їхню масштабованість.

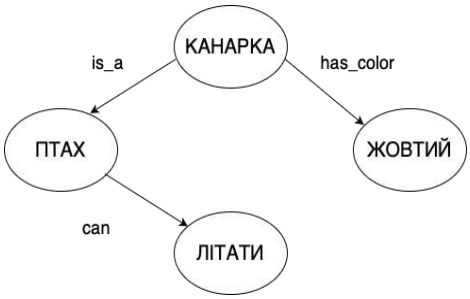

Рисунок 1.1 Приклад простої семантичної мережі

Наступним важливим етапом стала поява формальних онтологій – чітко визначених структур, які описують термінологію та взаємозв’язки в конкретних предметних областях. Цей підхід став центральним у концепції Semantic Web, яку запропонував Тім Бернерс-Лі на початку 2000-х років. Онтології дали змогу більш ефективно і зрозуміло представляти знання, використовуючи стандартизовані мови, такі як RDF і OWL.

Завдяки використанню RDF та OWL було реалізовано перші масштабні графи знань, серед яких найвідоміші – DBpedia та YAGO. DBpedia використовує автоматизоване вилучення структурованих даних з відкритих джерел, переважно з Вікіпедії, інтегруючи їх у єдиний зв’язний граф знань. YAGO поєднує онтологічний підхід з автоматизованими алгоритмами обробки інформації, що дозволяє йому досягати високої точності та масштабованості [4] [5].

Окрім згаданих графів, активного розвитку набули й такі сучасні графи знань, як Wikidata та Google Knowledge Graph. Wikidata, як відкритий граф знань, активно підтримується спільнотою, містить структуровані факти про понад сто мільйонів сутностей і постійно оновлюється завдяки краудсорсингу, що робить її однією з найдинамічніших графів знань сьогодення. Google Knowledge Graph є комерційним прикладом графа знань, який застосовується для покращення результатів пошуку (рис. 1.2), надаючи користувачам миттєвий доступ до структурованих знань безпосередньо у пошуковій видачі [6].

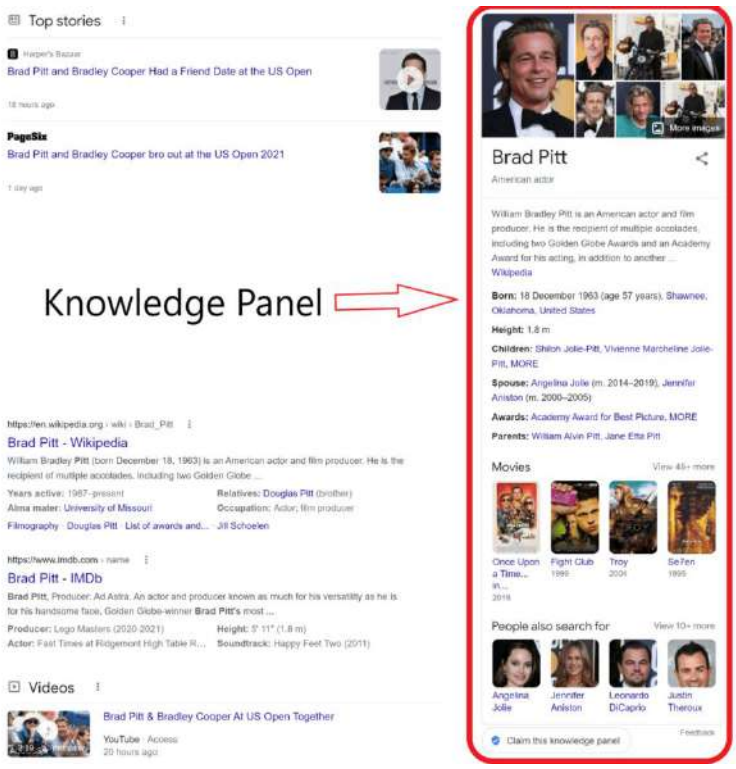

Рисунок 1.2 Приклад використання графа знань у пошуковій системі Google

Ці графи знань мають значні переваги над класичними експертними системами та іншими онтологічними моделями, серед яких висока масштабованість, можливість автоматичного оновлення та розширення графа знань, а також здатність легко інтегрувати різнорідні інформаційні ресурси в єдину цілісну систему. Завдяки використанню методів автоматизованого вилучення інформації, таких як IE та NER графи знань здатні ефективно підтримувати актуальність та швидко адаптуватись до змін предметної області [7].

Розділ 2. Теоретичні основи автоматизованого наповнення графів знань та Graph Question Answering

2.1 Основні поняття та підходи Natural Language Processing

Одним із ключових чинників, який дає змогу успішно вилучати структури знань із великої кількості текстових даних, є систематичне застосування методів NLP. Під ними розуміють сукупність теоретичних та прикладних методів, спрямованих на те, щоб комп’ютер міг аналізувати й інтерпретувати тексти так, як це робить людина. На відміну від формальних мов, природна мова характеризується варіативністю синтаксису, багатозначністю слів та розгалуженими семантичними зв’язками, що значно ускладнює її автоматизований аналіз [5].

Ранні системи NLP спиралися переважно на ручне кодування лінгвістичних правил та граматик [3]. Згодом, унаслідок стрімкого зростання обсягів даних, більшу популярність отримали статистичні методи: від найпростіших моделей на основі частотних показників до складних алгоритмів, зокрема Hidden Markov Models і Conditional Random Fields. Ці методи використовували певну кількість навчальних прикладів, на яких моделі вчились автоматично визначати мовні патерни. Проте по справжньому вагомий прорив відбувся з появою глибинних нейронних мереж. Ці мережі надали можливість враховувати контекст у кожній позиції тексту та опрацьовувати складні зв’язки між словами і фразами [10]. Як наслідок, точність розв’язання багатьох NLP завдань, таких як розпізнавання сутностей або класифікація тексту, суттєво зросла. Для подальшого розуміння процесів автоматизованого наповнення графа знань можна

виокремити кілька основних завдань NLP, які є найбільш значущими в контексті вилучення інформації з текстів:

- Токенізація. Розбиття суцільного тексту на окремі слова, розділові знаки, числові дані тощо. Це перший крок у будь-якому NLP процесі, від коректної реалізації якого залежить точність усіх наступних етапів.

- Лематизація та стемінг. Лематизація приводить кожне слово до його базової форми, тоді як стемінг обрізає закінчення. Обидва підходи зменшують розмаїття форм слів, що спрощує визначення основних понять. У мовах зі складною морфологією, як-от українська, якісна лематизація допомагає знизити обсяг помилок під час подальшого аналізу.

- Розпізнавання частин мови. Автоматичне визначення, чи є слово іменником, дієсловом, прикметником тощо. Цей крок є критичним для розуміння синтаксичних зв’язків між словами та подальшого виявлення іменованих сутностей.

- Синтаксичний аналіз. Дає змогу з’ясувати, у якій структурі перебувають слова в реченні: хто виконує дію, над чим або над ким вона виконується, якими є другорядні члени речення. Існують підходи на основі дерев фраз та дерев залежностей. Обидва дають змогу чіткіше встановлювати зв’язки в тексті, що надалі лягають в основу формалізації відношень між сутностями.

- Розпізнавання іменованих сутностей. Один із найважливіших етапів у побудові графа знань, оскільки забезпечує автоматичне виявлення назв організацій, осіб, місць, дат, подій тощо. Кожна з цих іменованих сутностей може стати вузлом у майбутньому графі знань.

- Семантичний аналіз. Включає в себе кілька підзавдань: розв’язання багатозначності (Word Sense Disambiguation), визначення семантичних ролей (Semantic Role Labeling), виявлення кореференції (Coreference Resolution). Ці процеси допомагають точніше зрозуміти зміст речення і, зокрема, «хто», «що» та «як» у ньому описується [16].

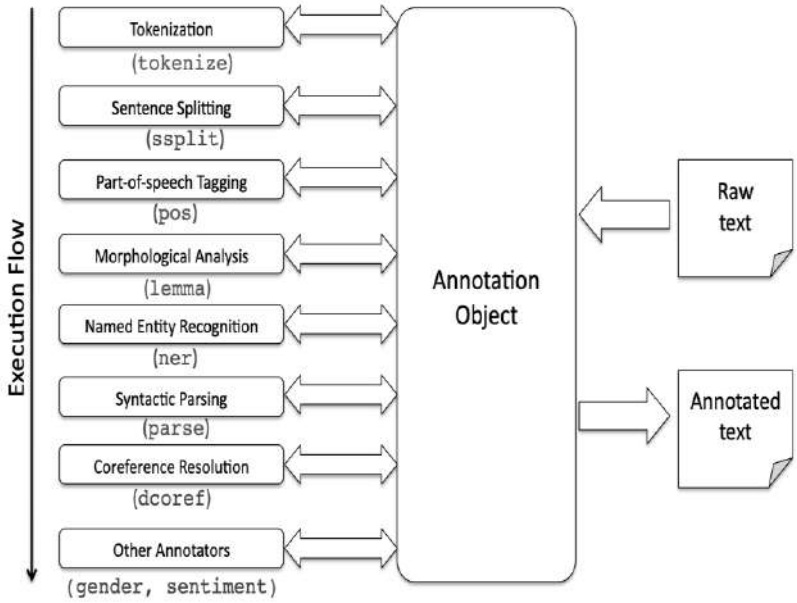

Рисунок 2.1 Етапи пайплайну обробки природної мови

У сучасній практиці NLP усе більшого поширення набувають нейронні архітектури, особливо трансформерні моделі, що забезпечують контекстне представлення слів у реченні. Такі підходи особливо корисні для систем, які працюють із великими обсягами даних і різними мовами, адже дозволяють ефективно перенавчати модель під конкретну предметну галузь або

завдання. Наприклад, моделі типу BERT можуть бути використані для високоточного розпізнавання іменованих сутностей чи вилучення відношень у тексті.

Важливість NLP для автоматизованого формування графів знань полягає в тому, що більшість інформації у світі існує саме в неструктурованій формі: статті, новини, звіти, довідники тощо. Щоб ці дані стали придатними для машинних алгоритмів на рівні семантичного пошуку чи формальних запитів, їх необхідно перетворити в певний структурований формат (наприклад, триплети «суб'єкт – предикат – об'єкт». Саме на цьому етапі в нагоді стають методи NLP:

- виявляють потенційні сутності;
- зв'язують їх з можливими відношеннями чи подіями;
- допомагають визначити контекст і перевірити цілісність вилученої інформації [10] [11].

2.2 Техніки інформаційного вилучення

Інформаційне вилучення відіграє центральну роль у побудові автоматизованих графів знань, оскільки дає змогу перетворювати великі обсяги неструктурованого тексту на впорядковані факти та відношення, що стають придатними для подальшої роботи алгоритмів. Основна мета IE полягає у визначенні, які сутності та події описані в тексті, як вони пов'язані між собою, а також за яких умов ті чи інші явища відбуваються. Завдяки

цьому можна формувати структуровані записи, що пізніше інтегруються в графи знань і забезпечують їх актуальність і повноту.

Одним із перших і найважливіших напрямів інформаційного вилучення є розпізнавання іменованих сутностей (NER). Цей підхід дає змогу виявляти в тексті назви людей, організацій, географічних локацій, дат та інших типів об’єктів, котрі можуть бути розглянуті як окремі вузли в графі знань. Порівняно з ранніми rule-based системами, сучасні методи NER переважно спираються на статистичні й нейронні моделі, здатні працювати зі складним синтаксичним і семантичним контекстом [17]. Завдяки цьому якість і точність NER суттєво зросли, що є важливим чинником у завданнях, пов’язаних із побудовою великих баз знань.

Ще одним важливим завданням у межах IE є вилучення відношень. Після визначення іменованих сутностей постає необхідність з’ясувати, у який спосіб вони взаємодіють і які зв’язки мають. Для прикладу, у тексті може згадуватися факт придбання однією компанією іншої або посилання на автора й назву його книги [10].

Окремим напрямом є вилучення подій. На відміну від статичних фактів, події є динамічними та описують, що саме відбувається, хто є учасником, де й коли це відбувається, а також які наслідки це має [9]. Аналіз подій складніший за ідентифікацію сутностей чи відношень, адже часто вимагає врахування контексту декількох речень або й цілих документів.

Серед інших технік інформаційного вилучення виокремлюють Open Information Extraction, яке дає змогу визначати широке коло фактів без попереднього обмеження переліком конкретних відношень [10]. Така

універсальність Open IE робить його корисним у випадках, коли розробник не має заздалегідь сформульованої онтології або має справу з великою кількістю різнорідних джерел. Утім, через відсутність наперед визначених шаблонів результати Open IE зазвичай потребують фільтрації та нормалізації, перш ніж бути інтегрованими в граф знань.

Важливим доповненням до перелічених завдань є семантичне зіставлення (Entity Linking), яке відповідає за те, щоби різні текстові згадки певного об’єкта були правильно пов’язані з тим самим вузлом у графі знань [6]. Це особливо актуально для масштабних систем, оскільки неправильне об’єднання або ж дублювання сутностей може спотворити цілісність і якість графа знань. У багатьох галузях, де використовуються аналітичні та пошукові системи, навіть незначні помилки в зіставленні сутностей здатні призвести до відчутних спотворень у результатах.

Нарешті, у прикладних розробках часто застосовують гібридні підходи, що поєднують статистичні та rule-based методи задля кращої точності та спеціалізації в певній предметній області [4]. Такий симбіоз дозволяє налаштовувати систему відповідно до конкретних вимог, водночас спираючись на гнучкі механізми машинного навчання для вирішення нетипових або складних випадків. Швидкі темпи поширення нової інформації вимагають і регулярного оновлення вилучених фактів, тож сучасні рішення у сфері IE здатні динамічно переобробляти великі масиви даних, тим самим підтримуючи актуальність і повноту графа знань.

2.3 Архітектури та методології автоматизованого формування графів знань

Автоматизоване наповнення графів знань з неструктурованих текстів часто використовує пайплайнові архітектури, що включають послідовні кроки: попередню обробку, вилучення сутностей, визначення відношень та збереження фактів. Цей підхід дозволяє комбінувати спеціалізовані інструменти і показав успіх у задачах Knowledge Base Population [9]. Основним недоліком є накопичення похибок, коли помилки ранніх етапів впливають на наступні. Хоча наскрізні моделі, що напряму генерують триплети, можуть пом’якшити цю проблему, вони часто обмежені коротким контекстом.

Однією з ключових тенденцій останніх років стало використання великих мовних моделей у системах автоматизованого вилучення знань [10]. На відміну від вузькоспеціалізованих алгоритмів, LLM здатні розуміти семантику тексту і виконувати одразу кілька завдань на основі контексту. Наприклад, сучасні трансформерні моделі можуть працювати в режимі zero-shot, генеруючи потрібні мітки без додаткового навчання на конкретному датасеті. Існують навіть спеціалізовані лейблери на основі мовних моделей, що слугують для гнучкого вилучення іменованих сутностей за заданим переліком типів [17]. Аналогічно, LLM можна залучити й на етапі генерації фактів: модель формує триплети на основі вхідного тексту, фактично поєднуючи в собі функціональність одразу двох класичних модулів – вилучення сутностей та визначення реляцій. Завдяки цьому знижується

кількість ручних правил та шаблонів, адже модель сама виводить потенційні відношення між знайденими сутностями [11].

Для забезпечення масштабованості знань та контекстної узгодженості LLM широко застосовується підхід Retrieval-Augmented Generation (RAG) – генерація з доповненням шляхом пошуку. RAG передбачає, що перед формуванням відповіді або вилучення знань модель спочатку отримує релевантну інформацію з зовнішнього сховища. Тобто LLM інтегрується з механізмом пошуку: з графа знань чи документа відбираються фрагменти, які стосуються поточного запиту або контексту, і подає їх моделі в промпті. Лише після цього модель генерує результат, спираючись тільки на надані знання. Подібна схема дозволяє подолати обмеження контекстного вікна великих моделей та актуалізувати їхній вихід: замість того, щоб намагатися охопити весь корпус текстів одразу, модель працює з невеликими вибірками релевантних даних, які дістала система пошуку. У такий спосіб RAG уможливлює обробку масштабних графів знань, оскільки факти зберігаються у зовнішньому індексі, а не інтегруються безпосередньо у параметри моделі. Важливо й те, що генерація з доповненням пошуком підвищує достовірність результатів: модель менше галюцинує, тобто генерує неіснуючі або непідтверджені дані, що не відповідають джерелам інформації, і частіше видає факти, що підтверджуються знайденими джерелами [25].

Хоча RAG спочатку розроблено для питально-відповідальних систем, цей підхід природно вписується і в процес побудови графів знань. Зокрема, LLM можна використовувати для вилучення фактів поетапно: спершу – виконати

пошук релевантних текстових фрагментів у великому корпусі (наприклад, усіх речень, де згадується певна сутність або потенційна пара сутність-сутність), далі – генерувати знання на основі кожного фрагмента окремо, ітеративно наповнюючи граф новими триплетами [12]. При цьому пошуковий механізм може спиратися як на неструктурований текст, так і на раніше добуті триплети: витягнуті факти у вигляді графа знань можуть бути використані як фільтр або контекст для наступних вилучень [13]. Така інтеграція пошуку і генерації демонструє, як LLM у поєднанні з векторними індексами дають можливість масштабувати побудову графа знань, динамічно залучаючи нові дані по мірі їх появи, без перенавчання моделей [15].

Сучасні системи автоматизованого формування графів знань зазвичай складаються з кількох типових компонентів, об’єднаних в єдиний пайплайн обробки даних. Основні компоненти такої архітектури можна перелічити так:

- Модуль попередньої обробки – виконує нормалізацію і підготовку неструктурованого тексту. Сюди входять очищення даних від зайвих символів, розбиття тексту на речення та абзаци, токенізація, усунення стоп-слів тощо [16].

- Лейблер сутностей – компонент, який розпізнає та позначає сутності у тексті відповідно до визначених категорій: «Персона», «Організація», «Локація» тощо. Лейблер може бути реалізований на основі правил або машинного навчання. Сучасний підхід – використовувати моделі, що дають змогу знаходити довільні сутності на основі промпту [17].

- Генератор відношень – вилучає зв’язки між виявленими сутностями, аналізуючи, чи згадуються вони в одному реченні або абзаці. Прості реалізації можуть базуватися на ручних шаблонах, тоді як розвинені системи послуговуються LLM для безпосереднього формування триплетів «суб’єкт – предикат – об’єкт» [12].

- Векторна база пам’яті – для семантичного пошуку, що зберігає вектори (embedding) фрагментів тексту, сутностей або триплетів. У ній можна швидко знайти найбільш схожі за змістом об’єкти й тим самим забезпечувати контекст для RAG [25].

- Графова база знань – фінальний компонент, де структуруються отримані факти: кожен триплет перетворюється на ребро між двома вузлами графа. Це дає змогу зберігати великі обсяги взаємопов’язаних даних і надалі виконувати ефективні графові запити.

У типовому сценарії всі ці компоненти працюють послідовно як єдина система. Наприклад, для корпусу наукових статей про біологію:

1. Модуль попередньої обробки розбиває текст на речення і очищує його від шуму.

2. Лейблер визначає біологічні сутності (наприклад, види організмів, гени, хвороби).

3. Генератор реляцій аналізує речення з розміченими сутностями і формує триплети.

4. Усі знайдені сутності та зв’язки додаються у векторний індекс для швидкого пошуку семантично подібних об’єктів.

5. Готові триплети зберігаються в Neo4j, де кожному вузлу й ребру призначається свій унікальний ідентифікатор і тип відношення.

6. Виконується ітеративна перевірка та дедублікація: якщо є дублікати сутностей, система намагається звести їх до єдиного вузла, вказуючи альтернативні назви [14].

Описаний workflow демонструє злагоджену роботу усіх компонентів: від сирого тексту – до графової бази знань, придатної до аналітичних завдань та Graph Question Answering. Завдяки залученню векторного пошуку та механізмів RAG створюється гнучка система, що дає змогу динамічно обробляти нові дані, забезпечуючи високу масштабованість і актуальність вилучених фактів [15]. Такий підхід закладає основу для побудови GQA систем, про які детальніше йтиметься у наступному підрозділі.

2.4 Graph Question Answering: концепція та значення

GQA – це різновид систем, які здатні автоматично відповідати на запитання, що використовують базу знань, представлену у вигляді графа знань. Інакше кажучи, GQA система інтерпретує питання, сформульоване природною мовою, та знаходить на нього відповідь, спираючись на знання, збережені у графовій структурі. Такі системи також називають Knowledge Graph Question Answering (KGQA), адже вони оперують графом знань, де вершинами є сутності, а ребрами – зв’язки між ними [17].

Для успішної роботи GQA системи необхідне виконання кількох етапів. Спочатку система аналізує текст запитання: виокремлює в ньому сутності та

визначає, яке відношення або властивість мається на увазі. Важливим аспектом цього етапу є коректне співвіднесення згаданих у питанні назв із відповідними сутностями графа знань та визначення типу запитуваного зв’язку чи атрибута [7]. Далі на основі результатів аналізу формується формальний запит до графової бази знань що відповідає змісту питання [7]. Виконання цього запиту по графу дає змогу отримати усі потенційні результати, які відповідають умовам запитання. На завершальному етапі GQA система оцінює знайдені результати та обирає серед них найбільш релевантну відповідь, подаючи її користувачеві у зручному вигляді.

Наприклад, якщо граф знань містить дані про книги та їхніх авторів, то на запитання «Хто є автором книги X?» GQA система спершу знайде у графі вузол, що відповідає сутності «книга X». Далі, використовуючи зв’язок типу «автор», вона визначить пов’язаний вузол (сутність автора) і на основі інформації з цього вузла поверне ім’я автора книги як відповідь.

Існує кілька підходів до реалізації GQA. Ранні системи використовували шаблонні методи: для певних типових формулювань питань вручну створювалися правила або шаблони перетворення їх у запити до графа знань. Згодом з’явилися підходи на основі семантичного розбору, коли питання трансформується у формальну логічну структуру, а вже на її основі генерується запит мовою типу SPARQL [7]. Сучасні дослідження активно застосовують методи глибинного навчання: нейронні моделі навчаються перетворювати питання або на відповідний запит, або навіть одразу на відповідь, використовуючи великі набори пар «запитання – відповідь» для тренування. Кожен з цих підходів має свої переваги і недоліки: шаблонні

правила забезпечують високу точність для простих запитів, але непридатні для довільних запитань; методи семантичного розбору вимагають експертних граматичних правил і можуть бути складними в розробці; нейронні моделі здатні обробляти різноманітні запитання та знаходити приховані залежності, проте результати їхньої роботи важче інтерпретувати і вони потребують великих обсягів даних для навчання.

Значення підходу GQA полягає в тому, що він дає змогу отримувати більш точні та релевантні відповіді на основі структурованих знань, уникаючи неоднозначності та шуму, притаманних текстовим даним [10]. Завдяки явному моделюванню зв’язків між фактами у графі знань GQA системи можуть відповідати на складні комплексні запитання, які потребують врахування кількох пов’язаних фактів одночасно. Таким чином, GQA поєднує методи NLP з перевагами семантично структурованих графів знань, що підвищує достовірність отриманих результатів [2].

На практиці технології GQA вже застосовуються у різноманітних сферах. Зокрема, їх використовують у семантичних пошукових системах, чат-ботах та інтелектуальних помічниках – всюди, де потрібно швидко надати користувачеві конкретну інформацію з великого масиву знань [5]. Крім того, у контексті автоматизованого наповнення графів знань GQA виконує роль перевірки та ефективного використання здобутих знань. Після побудови графа знань із текстових джерел GQA система дозволяє ставити до неї запитання та отримувати відповіді, демонструючи тим самим практичну цінність створеного графа знань.

2.5 Огляд існуючих датасетів та метрик оцінювання для GQA

Для об’єктивної оцінки та порівняння ефективності систем відповіді на запитання до графів знань, які також часто називають Knowledge Graph Question Answering (KGQA), наукова спільнота розробила та активно використовує стандартизовані датасети та відповідні метрики оцінювання.

Існуючі датасети для GQA/KGQA відрізняються за низкою ключових параметрів. До них належать: джерело графа знань, тип та рівень складності запитань (від простих фактів, що вимагають одного відношення в графі, до складних композиційних питань, що потребують багатоходового логічного виведення – multi-hop reasoning – та агрегації даних), загальний обсяг датасету та мова, якою сформульовані запитання. Наприклад, датасети на кшталт WebQuestionsSP були серед перших і фокусувалися на відносно простих запитаннях до Freebase. Згодом, для оцінки здатності систем працювати зі складнішими сценаріями, з’явилися такі датасети, як ComplexWebQuestions, що містять питання, які вимагають знаходження відповіді через кілька кроків у графі знань [14]. Важливим кроком стало створення датасетів на основі відкритих та активно розвиваних графів знань. Зокрема, LC-QuAD 1.0 та особливо його розширена версія LC-QuAD 2.0 стали популярними для тестування систем на графах знань DBpedia та Wikidata, причому LC-QuAD 2.0 спеціально розроблявся для оцінки роботи зі складними запитаннями у великих графах знань. Інші датасети, такі як MetaQA, цілеспрямовано створювалися для тестування саме багатоходового виведення, а GrailQA та KQA Pro – для оцінки здатності систем до генералізації та роботи з композиційними запитаннями. Вибір конкретного

датасету для оцінювання залежить від дослідницьких завдань та функціональності GQA-системи, яку необхідно перевірити [15].

Для кількісної оцінки якості роботи GQA-систем на згаданих датасетах використовуються стандартні метрики, що походять з галузей інформаційного пошуку, NLP та машинного навчання загалом [5]. Як буде детальніше описано, основними метриками є Точність (Precision), Повнота (Recall) та їх агрегований показник F1-міра (F1-score). Вони дозволяють оцінити, наскільки точно (скільки з наданих відповідей є правильними) та повно (яку частку всіх правильних відповідей було знайдено) система відповідає на запитання [10]. Коли для кожного запитання очікується лише одна або чітко визначена множина правильних відповідей, часто використовують метрику Точність (Accuracy) або Точний збіг (Exact Match, EM), яка вимірює частку запитань, де відповідь системи повністю збігається з еталонною [30]. У випадках, коли GQA система повертає ранжований список потенційних відповідей важливими стають метрики ранжування. Hits@k показує частку запитань, для яких правильна відповідь знаходиться серед перших k запропонованих системою варіантів (наприклад, Hits@1, Hits@5, Hits@10). Середня обернена позиція (Mean Reciprocal Rank, MRR) дає усереднену оцінку позиції першої правильної відповіді у видачі системи по всьому набору запитань [12].

2.6 Огляд сучасних інструментів та фреймворків для NLP та роботи з графами

Сучасні програмні рішення для NLP та роботи з графами знань пропонують широкий спектр можливостей, які можна ефективно поєднувати у межах єдиного пайплайну. Завдяки цьому розробники можуть налаштовувати багатоетапну обробку від початкової токенізації тексту й вилучення сутностей до побудови структурованих графів та відповіді на складні запитання.

Одним із найстаріших і водночас поширених інструментаріїв для NLP у Python є Natural Language Toolkit [18]. Він охоплює модулі для токенізації, лематизації, синтаксичного розбору, а також містить великий набір навчальних корпусів і лексичних ресурсів. За допомогою NLTK можна швидко прототипувати базові рішення з обробки текстів, проте в багатьох прикладних задачах, що потребують високої швидкодії, віддають перевагу spaCy [19]. Цей фреймворк орієнтований на промислові проєкти та забезпечує швидке й точне розпізнавання іменованих сутностей, визначення частин мови й побудову дерев залежностей. Крім того, у spaCy передбачено простий API для інтеграції з іншими бібліотеками або сервісами.

Серед інструментів на мові Java ключове місце посідає Stanford CoreNLP, що надає розширені можливості морфологічного й семантичного аналізу, в тому числі розпізнавання сутностей, виявлення кореференції та визначення семантичних ролей [16]. Існують обгортки для мови Python, що полегшують використання CoreNLP як частини складнішої системи. У контексті глибинного навчання останніми роками стала особливо популярною

платформа Hugging Face Transformers [27], де зібрано велику колекцію попередньо навчених моделей та датасетів. Вона дозволяє легко адаптувати моделі під конкретні завдання завдяки зручним інструментам на базі PyTorch або TensorFlow. Для обробки великих текстових масивів у розподіленому середовищі використовують Spark NLP [20], що базується на Apache Spark та забезпечує масштабованість під час виконання обчислень над великими корпусами.

На етапі формування графа знань у вигляді графа чи RDF сховища існують також різноманітні програмні платформи. Зокрема, Neo4j вважається одним із найпопулярніших рішень для зберігання й аналізу графових даних завдяки зручній мові запитів Cypher [26]. Для сценаріїв, де необхідно дотримуватися стандартів Semantic Web, часто використовують Apache Jena [21]. Цей фреймворк надає засоби для побудови, зберігання та опитування RDF-графів мовою SPARQL, а також підтримує онтологічні розширення. Конкурентним рішенням є GraphDB [24], орієнтований на роботу з великими триплетними сховищами з урахуванням семантичної логіки.

Якщо постає потреба поєднати документне чи ключ-значення сховище з графовим підходом, розглядають ArangoDB [22], що реалізує гібридний модель даних (документ + граф) і пропонує мову запитів AQL, здатну опрацьовувати як документні, так і графові структури. Для надвеликих розподілених графів використовують JanusGraph [23], що дає змогу будувати горизонтально масштабовані системи на базі Cassandra або HBase, зберігаючи сумісність зі стеком TinkerPop. Така опція є важливою для

інфраструктур, де обсяг даних постійно зростає і потрібно паралельне оброблення запитів у кластері.

У контексті побудови єдиної GQA-системи згадані інструменти найчастіше комбінують у межах пайплайну: спочатку відбувається попередня обробка й вилучення фактів, далі результати записуються у граф Neo4j [26] чи RDF сховище Jena [21], а для відповіді на запитання, сформульовані природною мовою, розробляється спеціальний шар, що виконує перетворення питань у Cypher або SPARQL та застосовує евристики або алгоритми поширення по графу, наприклад, Personalized PageRank.

Розділ 3. Проєктування та реалізація системи

3.1 Аналіз вимог до системи та вибір інструментарію

Під час проєктування системи автоматизованого вилучення знань із неструктурованих текстів та імплементації механізму Graph Question Answering було сформульовано низку вихідних вимог, що визначили ключові аспекти вибору технологічного стеку. Зважаючи на науково-дослідний характер роботи, до системи не ставилися суворі вимоги щодо промислової продуктивності, відмовостійкості чи безпеки. Натомість, основний фокус було зроблено на методологічній коректності вилучення сутностей та відношень, адекватному представленні знань у вигляді графа та розробці функціонального прототипу для відповіді на запитання.

З огляду на потребу обробляти тексти різної тематики без прив’язки до конкретного домену, система спроєктована для роботи з такими форматами вхідних даних як txt та jsonl. Для взаємодії з користувачем реалізовано веб інтерфейс на базі фреймворку Streamlit, що дозволяє завантажувати файли. Після завантаження система автоматично сегментує вхідні дані на окремі документи та ініціює вилучення знань: ідентифікацію іменованих сутностей, встановлення семантичних зв’язків між ними та збереження отриманих фактів у графовій базі даних.

Для реалізації ключових етапів формування графа знань та функціональності GQA було обрано наступні програмні засоби. Основною мовою програмування є Python, завдяки великій екосистемі бібліотек для обробки природної мови, машинного навчання та інтеграції з базами даних. Зберігання графа знань реалізовано за допомогою Neo4j, розгорнутої у

Docker контейнері. Neo4j забезпечує ефективне моделювання даних у форматі графа властивостей, де кластеризовані сутності представлені як вузли з унікальним ідентифікатором та текстом, а відношення між ними – як ребра. Кожне ребро зберігає свій ідентифікатор, тип відношення, семантичний опис, згенерований LLM, та динамічну вагу. Для забезпечення ефективного семантичного пошуку, який є частиною запропонованої GQA стратегії, застосовується бібліотека FAISS. Вона функціонує як векторне сховище в оперативній пам’яті та індексує виключно текстові описи ребер, зберігаючи в метаданих кожного вектора ідентифікатор відповідного ребра.

Центральну роль у процесах вилучення сутностей та відношень відіграють LLM, доступ до яких реалізовано через Ollama. Зважаючи на апаратні обмеження (локальне виконання на GPU з 16 ГБ відеопам’яті), було обрано квантизовану модель Gemma-3 12b-it-qat. На першому етапі ця модель генерує набір релевантних типів сутностей для поточного тексту. Ці типи потім використовуються моделлю GLiNER для безпосереднього розпізнавання іменованих сутностей у тексті. Згодом, та сама модель Gemma-3 застосовується для ідентифікації реляційних зв’язків між виявленими сутностями, генеруючи як тип зв’язку, так і його текстовий опис. Саме ці описи індексуються в FAISS, що є необхідним для реалізації обраного підходу до пошуку релевантної інформації для GQA.

Для фінального етапу GQA – генерації відповіді на запитання користувача на основі знайденого контексту – використовується інша LLM: deepseek-r1:14b. Ця модель була обрана через її оптимізацію для завдань міркування та здатності ефективно опрацьовувати наданий контекст, залишаючись при

цьому в межах апаратних обмежень. Ключовою особливістю розробленої системи є специфічний підхід до пошуку релевантного контексту для GQA, який поєднує семантичний пошук за описами ребер за допомогою FAISS та аналіз топології графа за допомогою алгоритму Personalized PageRank, реалізованого в Neo4j. Такий комбінований підхід спрямований на ідентифікацію найбільш релевантних вихідних документів, які потім передаються моделі для генерації відповіді.

Загальний інструментальний стек включає також стандартні бібліотеки для задач NLP та роботи з векторами, такі як spacy, torch та transformers. Використання Streamlit спростило створення інтерактивного інтерфейсу. Хоча поточна конфігурація орієнтована на локальне виконання і не передбачає високої пропускної здатності чи розподіленої обробки, вона закладає основу для подальших досліджень та потенційного масштабування. Модульність архітектури дозволяє експериментувати з різними LLM, алгоритмами вилучення та стратегіями GQA, що є важливим для наукового дослідження впливу окремих компонентів на загальну ефективність системи.

3.2 Архітектура системи

Запропонована система автоматизованого наповнення графа знань та відповіді на запитання спроєктована за модульним принципом для забезпечення гнучкості та можливості подальшого розвитку. Загальна архітектурна схема рішення представлена на рисунку 3.1.

Як видно зі схеми, система складається з чотирьох основних взаємопов’язаних компонентів, що реалізують послідовний пайплайн обробки даних та взаємодії з користувачем:

- Модуль збору та попередньої обробки текстів: відповідає за прийом вхідних даних та їх підготовку до аналізу.
- Модуль вилучення інформації: реалізує ключові етапи ідентифікації сутностей та відношень у текстах.
- Модуль формування графа знань: структурує вилучені факти та зберігає їх у графовій та векторній базах даних.
- Модуль Graph Question Answering: забезпечує обробку запитів користувача та пошук відповідей на основі сформованої бази знань.

Подальші підрозділи детально описують призначення, функціональність та взаємодію кожного з цих модулів.

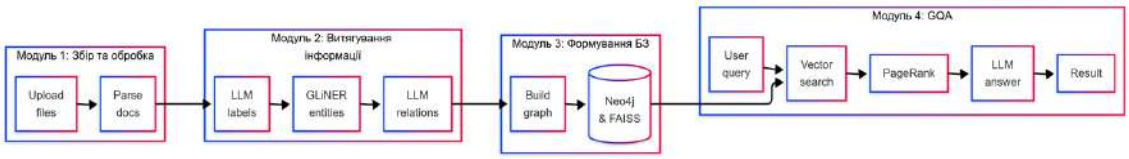

Рис 3.1 Архітектура програми

3.2.1 Модуль збору та попередньої обробки текстів

Процес автоматизованого наповнення графа знань розпочинається з модуля збору та попередньої обробки текстів. На цьому початковому етапі

система отримує вхідну неструктуровану текстову інформацію та готує її до подальшого аналізу засобами інформаційного вилучення.

Основні завдання модуля включають прийом вхідних даних та їх базову підготовку. Поточна версія системи дозволяє користувачеві завантажувати дані через спеціально розроблений веб-інтерфейс. Підтримуються два формати: txt та jsonl.

Після завантаження система автоматично визначає формат файлу та розбирає його вміст. Вміст простого текстового файлу розглядається як єдина текстова одиниця. У випадку JSON файлів кожен рядок інтерпретується як окремий структурований об’єкт. З таких об’єктів система вилучає основний текстовий вміст, а також, якщо вони присутні, асоційований ідентифікатор та інші метадані. Результатом цього етапу є уніфіковане внутрішнє представлення текстових одиниць, готових до передачі на наступні стадії обробки.

Для потенційного підвищення якості вилучення знань, особливо при роботі з великими текстами, можуть бути корисними додаткові кроки попередньої обробки. По-перше, сегментація суцільного тексту на менші семантичні одиниці за допомогою лінгвістичних інструментів дозволила б точніше визначати контекст для вилучення відношень та ефективніше використовувати ресурси LLM, враховуючи їхні обмеження на довжину вхідних даних. По-друге, очищення тексту від сторонніх елементів, які часто трапляються в реальних даних, покращило б якість аналізу. По-третє, текстова нормалізація сприяла б уніфікації даних. Хоча реалізація цих кроків

не була пріоритетом у поточній роботі, вони є важливим напрямком для майбутнього вдосконалення системи.

3.2.2 Модуль вилучення інформації

Центральну роль у перетворенні неструктурованих текстів на структуровані знання відіграє модуль вилучення інформації. Його завдання – ідентифікувати іменовані сутності та встановити семантичні відношення між ними, формуючи дані для графа знань. Процес вилучення поєднує можливості LLM та спеціалізованих інструментів для розпізнавання іменованих сутностей.

Спочатку система динамічно визначає релевантні для корпусу типи сутностей. Для цього використовується велика мовна модель Gemma-3 (12b-it-qat). Ця модель, хоч і має значний розмір завдяки квантизації забезпечує прийнятну продуктивність при локальному виконанні. Модель аналізує вибірку попередньо сегментованих вхідних документів і на основі спеціального промпту генерує список потенційних типів сутностей. Цей список агрегується, нормалізується та усуваються дублікати, що надає системі гнучкості та дозволяє адаптуватися до даних без фіксованої онтології.

Наступний крок – безпосереднє розпізнавання екземплярів сутностей відповідно до попередньо визначених типів. Тут застосовується модель GLiNER (gliner_large-v2.1). Вибір large версії спрямований на досягнення вищої точності. Ключова перевага GLiNER – здатність ефективно працювати

з довільними типами сутностей без донавчання. Модель аналізує фрагменти тексту та ідентифікує спани, що відповідають заданим типам, з урахуванням порогу впевненості. Важливо, що розпізнавання сутностей передує вилученню відношень. Така послідовність обрана свідомо: на етапі RE великій мовній моделі надаються вже верифіковані сутності в межах конкретного контексту. Це обмежує простір пошуку для LLM, фокусуючи її на знаходженні зв'язків між відомими об'єктами, що потенційно знижує ризик генерації хибних або нерелевантних відношень (галюцинацій), які можуть виникати при одночасному виконанні NER та RE [17].

Завершальний етап – виявлення та формалізація семантичних зв'язків між ідентифікованими сутностями. Для цього знову використовується велика мовна модель Gemma-3 (12b-it-qat). Моделі надається текстовий фрагмент та список виявлених у ньому сутностей. На основі спеціального промпту LLM генерує структуровані дані (JSON), що описують відношення як триплети «суб'єкт-предикат-об'єкт» разом із текстовим описом семантики зв'язку. Під час розробки розглядалися альтернативи, зокрема спеціалізовані моделі для RE, такі як GLiREL [28]. Однак експерименти показали, що вони часто вимагають попередньо визначеного набору типів відношень, що обмежувало б гнучкість системи, подібно до обмежень фіксованої онтології сутностей. Також спостерігалася схильність до генерації надлишкових або симетричних відношень (напр., А пов'язаний з В та В пов'язаний з А), що вимагало б додаткової фільтрації. Тому було обрано підхід із залученням LLM для генерації відношень на основі вже ідентифікованих сутностей. Після генерації відношень проводиться усунення дублікатів перед їх фінальним збереженням.

3.2.3 Модуль формування графа знань

Наступним етапом є модуль формування графа знань, який забезпечує персистентне зберігання структурованої інформації, отриманої на етапі вилучення (3.2.2). Цей модуль організовує вилучені сутності та відношення у двох взаємопов’язаних форматах: графовій базі даних Neo4j та векторному індексі FAISS, відповідно до обраної архітектури (3.1). Вхідними даними є множини ідентифікованих сутностей та вилучених семантичних відношень. Результатом роботи є оновлений граф знань, готова для використання модулем Graph Question Answering (3.2.4).

Ключовим завданням модуля є систематичне збереження та структурування вилучених фактів у графовій базі даних. Спочатку, на основі вилучених іменованих сутностей, формуються вузли графа: для кожного унікального текстового представлення сутності створюється окремий вузол з унікальним ідентифікатором. Таким чином, різні згадки однієї й тієї ж сутності, що мають однаковий текст, агрегуються в один вузол.

Далі обробляються вилучені семантичні відношення. Для кожного відношення визначаються вузли, що відповідають текстовим представленням його головної та залежної сутностей. Між цими вузлами створюється (або перевіряється наявність та оновлюється) спрямоване ребро, що відображає дане відношення.

Сформована таким чином структура графа знань, що складається з документів, вузлів та ребер, потім імпортується в графову базу даних Neo4j. При імпорті кожен вузол графа представлення створюється як вузол з міткою :Node та властивостями, що включають його ідентифікатор і текстове

представлення сутності. Кожне ребро представлення створюється як спрямоване ребро з міткою :Edge та властивостями, що містять ідентифікатор ребра (що відповідає ID вихідного семантичного відношення), тип відношення, згенерований LLM текстовий опис семантики зв’язку та початкову вагу. Ця початкова вага може надалі динамічно оновлюватися, наприклад, під час роботи алгоритму Personalized PageRank.

Паралельно з наповненням графа відбувається індексація текстових описів вилучених відношень у векторному сховищі FAISS. Для кожного опису ребра генерується векторний ембединг за допомогою моделі. Цей вектор разом із унікальним ідентифікатором відповідного ребра з Neo4j додається до індексу FAISS. Такий зв’язок між графовою структурою та векторним простором є критично важливим. Індексація саме описів ребер, а не сутностей чи повних текстів, є стратегічним рішенням, що відповідає обраному підходу до GQA, який поєднує семантичний пошук за описами зв’язків із топологічним аналізом графа.

3.2.4 Модуль Graph Question Answering

Модуль Graph Question Answering є завершальною ланкою розробленої системи, що дозволяє користувачам отримувати відповіді на запитання, сформульовані природною мовою, використовуючи структуровану інформацію, яка зберігається у вигляді графа знань. Основна задача цього модуля полягає у виборі релевантних вузлів та зв’язків графа на основі запиту, виконанні топологічного аналізу отриманого підграфа та генерації відповіді за допомогою LLM.

Алгоритмічний процес модуля GQA складається з таких послідовних етапів:

1. Семантичне кодування запиту

Першим кроком є перетворення вхідного запиту природною мовою на його векторне представлення. Для цього застосовується модель BAAI/bge-m3, яка генерує нормалізований до одиничної довжини вектор. Цей вектор далі використовується для пошуку найбільш релевантних описів ребер у векторному просторі FAISS.

2. Векторний пошук релевантних ребер

Згенерований вектор запиту використовується для виконання пошуку k найближчих сусідів (k-NN, де k зазвичай дорівнює 5) у векторному сховищі FAISS. В результаті отримується список ідентифікаторів ребер графа та їхніх значень косинусної подібності, які відображають релевантність відповідних ребер до запиту.

3. Оновлення ваг ребер графа

На основі результатів векторного пошуку, модуль динамічно оновлює ваги ребер у графовій базі даних Neo4j. Кожному знайденому ребру присвоюється вага, що відповідає значенню його косинусної подібності. Іншим ребрам встановлюється базова вага, що дозволяє алгоритму враховувати лише найбільш значущі зв’язки при подальшому аналізі.

4. Топологічний аналіз за допомогою Personalized PageRank

Для ранжування вузлів графа виконується алгоритм Personalized PageRank, реалізований у Neo4j Graph Data Science. В якості «насіннєвих»

вузлів використовуються кінцеві вузли ребер, отриманих на попередньому етапі. Результатом роботи PPR є вектор значущості вузлів графа, який демонструє, наскільки кожен вузол є релевантним для початкового запиту.

5. Агрегація ваг документів

Кожен вузол графа пов’язаний з одним або кількома документами. Загальна вага документа визначається як сума ваг усіх вузлів, пов’язаних із цим документом. Ці отримують свою оцінку релевантності, що дозволяє сортувати їх для наступного етапу.

6. Генерація відповіді за допомогою LLM

Документи з найвищими вагами передаються великій мовній моделі deepseek-r1:14b. Модель отримує запит користувача та цей контекст для генерації розгорнутої відповіді. В результаті повертається очищений від службових тегів текст відповіді, що надається користувачеві.

Цей підхід об’єднує переваги семантичного та топологічного аналізу, що дозволяє компенсувати недоліки кожного з них окремо. Семантичний пошук забезпечує початкову релевантність, тоді як Personalized PageRank враховує глобальну структуру зв’язків у графі. Динамічне оновлення ваг ребер мінімізує обчислювальні витрати, а інтеграція з LLM дозволяє створювати відповіді, які зрозумілі кінцевому користувачеві.

Попереднє тестування на графі з ~10 тисячами ребер показало, що середній час виконання одного запиту складає близько 10 секунд, що є прийнятним для прототипу, запущеному на локальній відеокарті.

3.3 Інтерфейс користувача: опис та особливості

Для забезпечення ефективної взаємодії користувача із системою було розроблено веб інтерфейс із застосуванням фреймворку Streamlit. Вибір цього інструменту зумовлений його простотою, високою швидкістю прототипування, а також можливістю створювати інтерактивні елементи безпосередньо у коді. Це значно спрощує демонстрацію функціональності системи, усуваючи потребу у створенні складного фронтенду.

Основний робочий простір інтерфейсу структурно поділений на дві вкладки, що відповідають ключовим функціям системи: «Document Processing» та «Question Answering».

Вкладка «Document Processing» призначена для наповнення графа знань. У межах цієї вкладки користувач має можливість:

1. Завантажувати неструктуровані текстові дані у форматах .txt або .jsonl через стандартний елемент завантаження файлів. Після успішного завантаження файли зберігаються для наступної обробки, а користувач отримує відповідне повідомлення.

2. Після завантаження файлу активується кнопка «Process Document». Натискання цієї кнопки ініціює вилучення знань, детально описане у попередніх розділах.

3. На цій вкладці також передбачені кнопки для роботи з графовою базою даних Neo4j: «Очистити базу даних Neo4j» та «Вставити дані у Neo4j», які стають доступними після завершення процесу обробки документів.

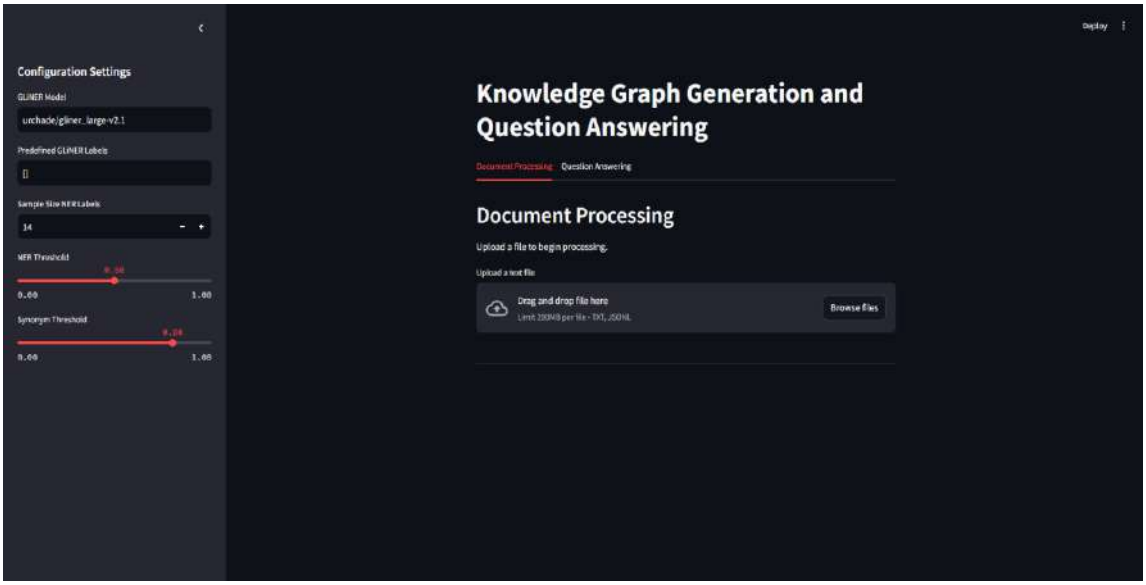

Рисунок 3.2 – Загальний вигляд інтерфейсу користувача системи

Крім основного робочого простору, інтерфейс містить бічну панель для налаштування конфігураційних параметрів системи. Тут користувач може змінювати назви моделей, встановлювати порогові значення для NER чи визначення синонімів, яке буде реалізоване в майбутньому або вказувати лейбли для моделі GLiNER.

Вкладка «Question Answering» надає інтерфейс для взаємодії з модулем GQA. Користувач вводить своє запитання природною мовою у текстове поле «Enter your question» і натискає кнопку «Get Answer». Після цього система активує GQA-пайплайн. Згенерована відповідь відображається безпосередньо на сторінці.

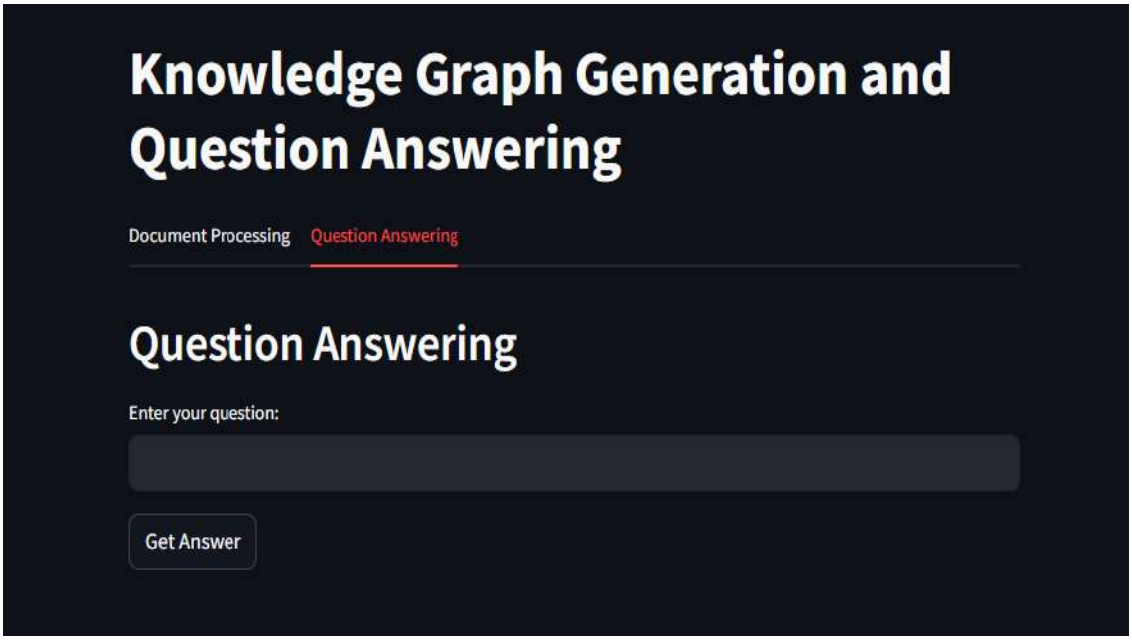

Рисунок 3.3 Вигляд вкладки Question Answering

Розділ 4. Тестування та оцінювання ефективності системи

4.1 Методологія тестування

Для комплексної оцінки ефективності запропонованого рішення було застосовано дві методології тестування:

Перша методологія полягала в оцінці якості вилучення фактів за допомогою бенчмарку MINE (Measure of Information in Nodes and Edges), запропонованого авторами системи KGGen [32]. Тестування передбачає ручне виокремлення 15 ключових фактів із тексту статті обсягом близько 1000 слів, після чого оцінюється здатність системи відтворити ці факти за допомогою побудованого графа знань. Семантичний пошук та додатковий аналіз великою мовною моделлю, а саме OpenAI o4-mini, визначають, чи можна зробити висновок про факт на основі наявної інформації у графі.

Друга методологія базувалася на кількісному оцінюванні здатності системи знаходити релевантні документи в графі знань. Для цього використовувався датасет MuSiQue-Ans [29], що містить 100 запитань, для кожного з яких визначений один еталонний документ із необхідною інформацією. Тестування складалося з таких етапів: система отримувала запитання природною мовою, після чого повертала список документів, ранжованих за комбінованим критерієм семантичної близькості та алгоритму Personalized PageRank. На завершальному етапі визначалося, на якій позиції вперше з’являється еталонний документ. Для аналізу було застосовано класичні рангові метрики: Mean Reciprocal Rank (MRR), Hits@k та Precision@k для k = 1, 3, 5, 10. У цій постановці для кожного запиту існує рівно один еталонний документ, тому Precision@k дорівнює Hits@k/k і

теоретично не може перевищити 1/k. MRR визначає середню позицію першого релевантного документа, Hits@k – частку запитань, для яких правильний документ знаходиться у перших k результатах, Precision@k – точність вибору k найбільш релевантних документів.

4.2 Результати тестування та їх аналіз

У цьому розділі представлені детальні результати, отримані при застосуванні обох методологій тестування, та їхній аналіз.

4.2.1 Результати тестування бенчмарком MINE

Для забезпечення прозорості та відтворюваності результатів, тестування за бенчмарком MINE було проведено з використанням відкритого коду, наданого авторами KGGen, доступного у їхньому офіційному репозиторії на GitHub [33]. У цей код було інтегровано власний пайплайн для вилучення сутностей та реляцій, що замінив стандартний модуль генерації графа знань. Це дозволило об’єктивно порівняти ефективність запропонованого підходу з іншими системами: KGGen, GraphRAG та OpenIE за однакових умов тестування.

У рамках цього експерименту запропонована система показала середню точність відтворення фактів на рівні 59.73%. Для порівняння, оригінальна система KGGen досягає точності 66.07%, GraphRAG – 47.80%, а OpenIE – 29.84% [32]. KGGen має подібний до нашої системи пайплайн, що включає виділення сутностей та зв’язків за допомогою мовних моделей, після чого формується граф знань.

Варто зазначити, що для створення сутностей та відношень у KGGen використовувалась модель OpenAI GPT-4o, тоді як у запропонованій системі – локальна квантизована модель Gemma-3 (12b-it-qat) [32]. Незважаючи на вказані обмеження, отримані результати свідчать про те, що запропонована система є конкурентоспроможною порівняно з наявними сучасними рішеннями та ефективно виконує завдання вилучення знань із текстових джерел.

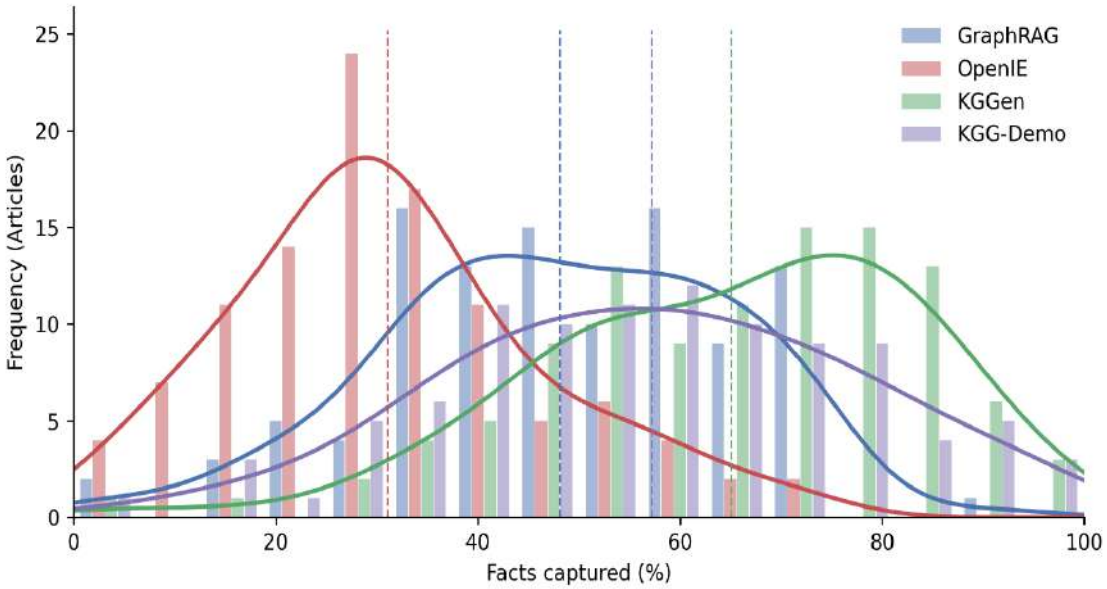

Рисунок 4.1. Порівняльна оцінка якості вилучення фактів

4.2.2 Результати рангового оцінювання

За результатами проведеного експерименту середня обернена позиція (MRR) становить 0,7820, що підтверджує високу загальну ефективність системи, оскільки релевантний документ у середньому розміщується дуже

близько до початку ранжованого списку. Метрика Hits@1 дорівнює 0,6500, тобто у 65% випадків система визначає правильний документ як перший у списку. Показники Hits@3 (0,8800) та Hits@5 (0,9300) додатково підкреслюють здатність системи ефективно знаходити релевантні документи у верхній частині видачі. Найвищий показник Hits@10 (0,9700) демонструє, що майже в усіх випадках потрібний документ потрапляє до десятки найкращих результатів.

Аналіз Precision@k виявляє закономірне зниження точності зі збільшенням кількості документів у видачі: Precision@1 становить 0,6500, що відповідає значенню Hits@1. З огляду на верхню межу $1/3 \approx 0{,}333$, Precision@3 = 0,2933 наближається до максимально можливого значення; аналогічно, Precision@5 (0,1860 з максимумом 0,2) та Precision@10 (0,0970 з максимумом 0,1) демонструють, що система зберігає високу відносну точність навіть при розширенні списку результатів до 10 позицій.

Для контекстуалізації отриманих результатів варто розглянути показники ефективності на схожих завданнях. Хоча пряме порівняння ускладнене через відмінності в архітектурах та цілях пошуку, можна навести деякі приклади. На датасеті MuSiQue state-of-the-art системи досягають F1-міри для кінцевої відповіді близько 60-70%. На схожому датасеті HotpotQA метрики пошуку речень доказів часто знаходяться в діапазоні 70-85%. У класичних завданнях KGQA (напр., на LC-QuAD 2.0) метрики пошуку сутностей, такі як Hits@1, можуть варіюватися від 40% до 75%, а MRR – від 0.5 до 0.8 [30] [31].

Висновки

У кваліфікаційній роботі успішно досліджено проблему автоматизованого наповнення графів знань з неструктурованих текстів та розроблено прототип програмної системи, що реалізує запропонований підхід. Продемонстровано ефективність поєднання сучасних методів обробки природної мови, зокрема великих мовних моделей, для вилучення сутностей та відношень з подальшим структуруванням у графовій базі даних Neo4j та забезпеченням доступу до знань через механізм Graph Question Answering.

Ключовими результатами роботи є аналіз існуючих підходів, що обґрунтував архітектурні рішення, та реалізація обчислювального пайплайну, який включає динамічне визначення типів сутностей, розпізнавання іменованих сутностей, генеративне вилучення відношень та їх збереження. Інтегрований модуль GQA використовує комбінований пошук релевантного контексту для генерації відповідей великою мовною моделлю. Тестування на бенчмарках MINE та датасеті MuSiQue-Ans підтвердило конкурентоспроможність системи у вилученні фактів та ефективність у пошуку релевантних документів. Розроблена система демонструє практичну цінність у якісному перетворенні текстових корпусів на графи знань, актуальні для аналітичних платформ та інтелектуальних асистентів.

Отримані результати відкривають перспективи для подальших досліджень. Важливими напрямками є поглиблення лінгвістичної обробки текстів, зокрема шляхом інтеграції методів розв’язання кореференції та розширеної нормалізації. Подальшого вдосконалення потребують механізми вилучення сутностей та відношень, включаючи тонке налаштування

моделей, покращення промптів для LLM, а також розробку алгоритмів для автоматичної нормалізації та кластеризації вилучених відношень та розв’язання неоднозначності сутностей. Модуль GQA може бути покращений за рахунок експериментів з різними LLM, розробки складніших стратегій пошуку контексту та підтримки запитань, що вимагають агрегації чи багато крокового виведення. Окрім того, актуальними залишаються завдання підвищення продуктивності системи шляхом оптимізації та паралелізації обчислень, а також розширення можливостей користувацького інтерфейсу, зокрема додавання візуалізації графів та інструментів валідації даних.

Список використаної літератури

1. Kejriwal, M. Knowledge Graphs: A Practical Review of the Research Landscape [Електронний ресурс]. – 2022. – Режим доступу: https://www.mdpi.com/2078-2489/13/4/161 (дата звернення: 13.01.2025).

2. Jiang X., Xu C., Shen Y. On the Evolution of Knowledge Graphs: A Survey and Perspective [Електронний ресурс]. – 2023. – Режим доступу: https://arxiv.org/abs/2310.04835 (дата звернення: 14.01.2025).

3. Haocheng Tan A brief history and technical review of the expert system research [Електронний ресурс]. – 2017. – Режим доступу: https://iopscience.iop.org/article/10.1088/1757-899X/242/1/012111 (дата звернення: 14.01.2025).

4. Paulheim H. Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods [Електронний ресурс]. – 2016. – Режим доступу: https://semantic-web-journal.net/system/files/swj1167.pdf (дата звернення: 23.01.2025).

5. Ji S., Pan S., Cambria E., Marttinen P., Yu P. S. A Survey on Knowledge Graphs: Representation, Acquisition, and Applications [Електронний ресурс]. – 2021. – Режим доступу: https://arxiv.org/abs/2002.00388 (дата звернення: 23.01.2025).

6. Nickel M., Murphy K., Tresp V., Gabrilovich E. A Review of Relational Machine Learning for Knowledge Graphs [Електронний ресурс]. – 2015 – Режим доступу: https://arxiv.org/abs/1503.00759 (дата звернення: 09.02.2025).

7. Ji H., Grishman R. Knowledge Base Population: Successful Approaches and Challenges [Електронний ресурс]. – 2011. – Режим доступу: https://aclanthology.org/P11-1115/ (дата звернення: 10.02.2025).

8. García-Silva A., Gómez-Pérez J. M. Autoregressive Language Models for Knowledge Base Population: A case study in the space mission domain [Електронний ресурс]. – 2025. – Режим доступу: https://arxiv.org/abs/2503.18502 (дата звернення: 21.04.2025).

9. Yang Y., Wu Z., Yang Y. *ma ін.* A Survey of Information Extraction Based on Deep Learning [Електронний ресурс]. – 2022. – Т. 12, № 19, 9691. – Режим доступу: https://www.mdpi.com/2076-3417/12/19/9691 (дата звернення: 17.02.2025).

10. Liu P., Gao W., Dong W. *ma ін.* A Survey on Open Information Extraction from Rule-based Model to Large Language Model : arXiv preprint arXiv:2208.08690 [Електронний ресурс]. – 2024. – Режим доступу: https://arxiv.org/abs/2208.08690 (дата звернення: 17.02.2025).

11. Saxena A., Tripathi A., Talukdar P. Improving Multi-hop Question Answering over Knowledge Graphs using Knowledge Base Embeddings [Електронний ресурс]. – 2020. – Режим доступу: https://aclanthology.org/2020.acl-main.412.pdf (дата звернення: 20.02.2025).

12. Liang S., Stockinger K., de Farias T. M. Querying knowledge graphs in natural language [Електронний ресурс]. – 2021. – Режим доступу: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00383-w (дата звернення: 2.03.2025).

13. Song Y., Li W., Dai G., Shang X. Advancements in Complex Knowledge Graph Question Answering: A Survey [Електронний ресурс]. – 2023. – Режим

доступу: https://www.mdpi.com/2079-9292/12/21/4395 (дата звернення: 02.03.2025).

14. Lan Y., He G., Jiang J. *та ін.* Complex Knowledge Base Question Answering: A Survey [Електронний ресурс]. – 2022. – Режим доступу: https://arxiv.org/abs/2108.06688 (дата звернення: 02.03.2025).

15. Dubey M., Banerjee D., Abdelkawi A., Lehmann J. LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia [Електронний ресурс]. – 2019. – Режим доступу: https://jens-lehmann.org/files/2019/iswc_lcquad2.pdf (дата звернення: 02.03.2025).

16. Manning C., Surdeanu M., Bauer J. The Stanford CoreNLP Natural Language Processing Toolkit [Електронний ресурс]. – 2014. – Режим доступу: https://aclanthology.org/P14-5010.pdf (дата звернення: 02.03.2025).

17. Zaratiana U., Tomeh N., Holat P., Charnois T. GLiNER: Generalist Model for Named Entity Recognition using Bidirectional Transformer [Електронний ресурс]. – 2023. – Режим доступу: https://arxiv.org/abs/2311.08526 (дата звернення: 02.03.2025).

18. NLTK Documentation [Електронний ресурс]. – Режим доступу: https://www.nltk.org/ (дата звернення: 02.03.2025).

19. spaCy Documentation [Електронний ресурс]. – Режим доступу: https://spacy.io (дата звернення 02.03.2025).

20. Kocaman V., Talby D. Spark NLP: Natural Language Understanding at Scale [Електронний ресурс]. – 2021. – Режим доступу: https://arxiv.org/abs/2101.10848 (дата звернення 03.03.2025).

21. Apache Jena Documentation [Електронний ресурс]. – Режим доступу: https://jena.apache.org/ (дата звернення 03.03.2025).

22. ArangoDB Documentation [Електронний ресурс]. – Режим доступу: https://www.arangodb.com/docs/stable/ (дата звернення 03.03.2025).

23. JanusGraph Documentation [Електронний ресурс]. – Режим доступу: https://docs.janusgraph.org/ (дата звернення 03.03.2025).

24. GraphDB Documentation [Електронний ресурс]. – Режим доступу: https://graphdb.ontotext.com/documentation/10.6/ (дата звернення 03.03.2025).

25. Lewis P., Perez E., Piktus A. Retrieval-augmented generation for knowledge-intensive NLP tasks [Електронний ресурс]. – 2020. – Vol. 33. – Режим доступу: https://arxiv.org/abs/2005.11401 (дата звернення: 05.03.2025).

26. Neo4j Documentation [Електронний ресурс]. – Режим доступу: https://neo4j.com/docs/ (дата звернення: 05.03.2025).

27. Wolf T., Debut L., Sanh V. Transformers: State-of-the-art natural language processing [Електронний ресурс]. – 2019. – Режим доступу: https://arxiv.org/abs/1910.03771 (дата звернення: 05.03.2025).

28. Boylan J., Hokamp C., Gholipour Ghalandari D. GLiREL – Generalist Model for Zero-Shot Relation Extraction [Електронний ресурс]. – 2025. – Режим доступу: https://arxiv.org/abs/2501.03172 (дата звернення: 05.03.2025).

29. Trivedi H., Balasubramanian N., Khot T., Sabharwal A. MuSiQue: Multihop Questions via Single-hop Question Composition [Електронний ресурс]. – 2022. – Режим доступу: https://arxiv.org/abs/2108.00573 (дата звернення: 28.03.2025).

30. Hoyt C. T., Berrendorf M., Galkin M. A Unified Framework for Rank-based Evaluation Metrics for Link Prediction in Knowledge Graphs [Електронний ресурс]. – 2022. – Режим доступу: https://arxiv.org/abs/2203.07544 (дата звернення: 28.03.2025).

31. Xie T., Wu C. H., Shi P., Zhong R. UnifiedSKG: Unifying and Multi-Tasking Structured Knowledge Grounding with Text-to-Text Language Models [Електронний ресурс]. – 2022. – Режим доступу: https://arxiv.org/abs/2201.05966 (дата звернення: 28.03.2025).

32. Mo B., Yu K., Kazdan J., Mpala P., Yu L., Cundy C., Kanatsoulis Ch., Koyejo S. KGGen: Extracting Knowledge Graphs from Plain Text with Language Models [Електронний ресурс]. – 2025. – Режим доступу: https://arxiv.org/abs/2502.09956 (дата звернення: 06.04.2025).

33. kg-gen: github [Електронний ресурс]. – Режим доступу: https://github.com/stair-lab/kg-gen (дата звернення: 06.04.2025).