НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри
_____ Оксана ТИМОЩУК

«___»_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Методи побудови мереж термінів на основі текстового аналізу та штучного інтелекту»

Виконав :
студент IV курсу, групи КА-04
Мірошниченко Михайло Андрійович _____

Керівник:
канд. техн. наук, ст. викл. каф.
ММСА, Савастьянов В.В. _____

Консультант з економічного розділу:
доцент, к.е.н. Мажара Гліб Анатолійович _____

Консультант з нормоконтролю:
к.ф.-м.н. Статкевич Віталій Михайлович _____

Рецензент:
Пр., д.т.н., з.к.і.б., Ланде Дмитро Володимирович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Оксана ТИМОЩУК

«___»_____2024 р.

ЗАВДАННЯ

на дипломну роботу студенту

Мірошниченку Михайлу Андрійовичу

1. Тема роботи «Методи побудови мереж термінів на основі текстового аналізу та штучного інтелекту», керівник роботи канд. техн. наук, ст. викл. каф. ММСА, Савастьянов В.В. затверджені наказом по університету від «31» травня 2024 р. № 2240-с.

2. Термін подання студентом роботи 11.06.2024

3. Вихідні дані до роботи: статистичні дані по графам створеним ШІ

4. Зміст роботи: дослідження предметної області, алгоритми побудови графів знань та їх перевірки, розробка програмного продукту, ФВА програмного продукту. проф.

5. Перелік ілюстративного матеріалу: презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Мажара Гліб Анатолійович		

7. Дата видачі завдання: 15.04.2024 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формулювання тематики дослідження	15.04.2024 – 21.04.2024	Виконано
2	Аналіз актуальності вибраної теми	22.04.2024 – 29.04.2024	Виконано
3	Підготовка першого розділу дипломної роботи.	30.04.2024 – 19.05.2024	Виконано
4	Підготовка другого розділу дипломної роботи	20.05.2024 – 23.05.2024	Виконано
5	Підготовка третього розділу дипломної роботи	24.05.2024 – 27.05.2024	Виконано
6	Підготовка четвертого розділу дипломної роботи	28.05.2024 – 01.06.2024	Виконано
7	Підготовка презентації для захисту	02.06.2024 – 03.06.2024	Виконано
8	Попередній захист дипломної роботи	04.06.2024	Виконано
9	Захист дипломної роботи	17.06.2024 – 21.06.2024	

Студент Михайло МІРОШНИЧЕНКО

Керівник Володимир САВАСТЬЯНОВ

РЕФЕРАТ

Дипломна робота: 160 с., 9 табл., 38 рис., 16 джерел.

LLM, ГРАФ ЗНАНЬ, PYTHON, OLLAMA, PROMPTING, RAG, GRAG, ШТУЧНИЙ ІНТЕЛЕКТ.

Об'єктом дослідження є процес побудови графів знань, що включає опрацювання набору текстових даних заданої предметної області з приведенням їх у зручний для розуміння формат за допомогою технологій промптингу і LLM, а також методів синтаксичного аналізу тексту.

Постановка задачі: Використовуючи LLM та технології промптингу, створити інструментарій для побудови графів знань, який може працювати з будь-яким текстом і надавати можливість перетворювати його у зрозумілі графічні структури, що можна перевірити на точність та стабільність.

Метою роботи є створення алгоритмів для побудови та перевірки графів знань, забезпечення можливості для користувача вибирати налаштування моделі та виконувати перевірки з наданням статистики щодо різниці графів, а також візуалізувати принципи синтаксичного аналізу речень.

Дипломний проект також передбачає розробку інтуїтивного користувацького інтерфейсу та створення підходу щодо оцінювання результатів для подальшого аналізу результатів користувачами.

ABSTRACT

Thesis: 160 pages, 9 tables, 38 figures, 16 references.

LLM, KNOWLEDGE GRAPH, PYTHON, OLLAMA, PROMPTING, RAG, GRAG, ARTIFICIAL INTELLIGENCE.

The research object is the process of constructing knowledge graphs, which involves processing a set of text data in a given subject area and transforming it into an easily understandable format using prompting technologies and Large Language Models (LLMs), as well as methods of syntactic text analysis.

Task statement: Using LLMs and prompting technologies, create a toolkit for building knowledge graphs that can work with any text and enable its transformation into understandable graphical structures, which can be verified for accuracy and stability.

The goal of this work is to create algorithms for constructing and verifying knowledge graphs, providing users with the ability to select model settings and perform checks with statistics on graph differences, as well as visualize the principles of syntactic sentence analysis. The diploma project also includes the development of an intuitive user interface and the creation of an approach for evaluating results for further analysis by users.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

RDF(Resource Description Framework) — Фреймворк Опису Джерела;

CoT(Chain of thought) — Ланцюг Думок (метод промптингу);

ToT(Tree of Thought) — Дерево Думок (метод промптингу);

RAG(Retrieval Augmented Generation) — Пошуково Покращена Генерація (метод промптингу);

GRAG(Graph Retrieval Augmented Generation) — Графова Пошуково Покращена Генерація (метод промптингу);

LLM (Large Language Model) — ВММ (Велика Мовна Модель);

NLP(Natural Language Processing) — Обробка Природної Мови;

ВСТУП

Живучи в світі з обширним інформаційним середовищем, де щодня кожен може знайти якусь корисну інформацію, хоч то буде новина, або якась наукова стаття, ми стикаємося з численними викликами. З одного боку, це величезна перевага, адже інформація стала доступною як ніколи раніше. Ми можемо в будь-який момент отримати доступ до найновіших відкриттів, технологічних новинок, дослідницьких матеріалів, новинних статей і багатьох інших ресурсів. З іншого боку, з розширенням інформаційного простору, наступає проблема систематизації знань, бо без систематизації тяжко уявити навчання, або навіть знаходження необхідної інформації.

Проблема систематизації полягає в тому, що інформація розпорошена по різних джерелах і форматах, що ускладнює її ефективне використання. Самостійно це робити тяжко, враховуючи масиви інформації для опрацювання. Зростання обсягів даних призводить до того, що людина просто не в змозі обробити всю інформацію самостійно, не кажучи вже про те, щоб її проаналізувати, структурувати і представити в зручній формі.

Однак, з використанням штучного інтелекту, можливо значно спростити будь-який процес, пов'язаний з обробкою великих масивів інформації. Наприклад, ШІ здатний систематизувати медичні бази даних та зробити їх доступними для кожного. Використовуючи машинне навчання та алгоритми обробки природної мови, штучний інтелект може аналізувати медичні записи, дослідження та публікації, створюючи впорядковані та легко доступні ресурси. Це може бути корисним не лише для лікарів, які можуть швидко знайти необхідну інформацію для діагностики та лікування, але й для пацієнтів, які зможуть отримати доступ до перевіреної та актуальної інформації про своє здоров'я.

Метою даної роботи є створення універсальної бази знань на основі статей з ресурсів на кшталт медичного ресурсу PubMed, та з використанням штучного інтелекту, для демонстрації можливостей сьогоденного ШІ та реалізації алгоритму, який може кардинально спростити будь-який набір даних

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ.

1.1 Граф знань та алгоритм його побудови

Для початку реалізації даної задачі потрібно розуміти, що таке граф знань. На це питання нам допоможе відповісти стаття [1]. Варто почати з розповіді про основу — фреймворк опису джерела (Resource Description Framework, RDF). RDF, є стандартом Консорціуму Всесвітньої павутини (W3C), спочатку розробленим як модель даних для метаданих. Він використовується як загальний метод опису та обміну графічними даними. RDF надає різноманітні нотації синтаксису та формати серіалізації даних, при цьому мова Turtle (Terse RDF Triple Language) наразі є найбільш поширеною нотацією.

RDF — це напрямлений граф, складений з триплетних висловлювань. Висловлювання RDF-графа представлені як: 1) вузол для суб'єкта, 2) дуга, що йде від суб'єкта до об'єкта для присудка, і 3) вузол для об'єкта. Кожну з трьох частин висловлювання можна ідентифікувати за допомогою уніфікованого ідентифікатора ресурсу (URI). Об'єкт також може бути літеральним значенням. Ця проста, гнучка модель даних має велику виразну силу для представлення складних ситуацій, відносин та інших цікавих речей, при цьому вона є відповідно абстрактною [1].

RDF був прийнятий як рекомендація W3C у 1999 році. Специфікація RDF 1.0 була опублікована у 2004 році, а специфікація RDF 1.1 — у 2014 році. SPARQL — це стандартна мова запитів для графів RDF. Мови онтологій, такі як RDF Schema (RDFS), Web Ontology Language (OWL) та

SHACL (Shapes Constraint Language), використовуються для опису даних RDF.

Модель даних RDF схожа на класичні підходи концептуального моделювання (наприклад, діаграми сутність–зв'язок або класів). Вона базується на ідеї роблення висловлювань про ресурси (зокрема веб-ресурси) у виразах у вигляді суб'єкт–присудок–об'єкт, які відомі як триплети. Суб'єкт позначає ресурс, а присудок вказує на характеристики або аспекти ресурсу, виражаючи відношення між суб'єктом і об'єктом.

Наприклад, один зі способів представлення поняття "Небо має колір синій" у RDF полягає у триплеті: суб'єкт, що позначає "небо", присудок, що позначає "має колір", та об'єкт, що позначає "синій". Таким чином, RDF використовує суб'єкт замість об'єкта (або сутності), на відміну від типового підходу об'єктно-орієнтованого проектування: сутність (небо), атрибут (колір) та значення (синій).

RDF є абстрактною моделлю з кількома форматами серіалізації (в основному, це спеціалізовані формати файлів). Крім того, конкретне кодування для ресурсів або триплетів може відрізнятися від формату до формату.

Цей механізм опису ресурсів є основною складовою діяльності Семантичної павутини W3C: еволюційного етапу Всесвітньої павутини, на якому автоматизоване програмне забезпечення може зберігати, обмінюватися та використовувати машинночитану інформацію, розподілену по всій павутині, що в свою чергу дозволяє користувачам працювати з інформацією з більшою ефективністю та впевненістю. Проста модель даних RDF та можливість моделювання різних.

Знаючи це ми можемо перейти до розв’язання задачі.

У статті [2] наводиться постановка задачі з пошуку відповідей на запитання на базі великих об'ємів текстової інформації.

Автор показує що ChatGPT не є відповіддю на всі наші потреби. GPT навчений на загальних наборах даних. Він може дуже добре розуміти та інтерпретувати природні мови. Він також може досить добре міркувати. Однак він не є експертом у якійсь певній галузі. Отже, хоча він може мати певні знання по темі, він не може дати глибокі достовірні відповіді. Іноді GPT може взагалі не мати відповіді. У таких випадках він або відмовляється відповідати на запитання, або впевнено їх вигадує (галюцинації).

Автор наводить стандартну архітектуру системи отримання відповідей на основі знань на базі бібліотеки LangChain. Нижче вказані кроки для досягнення KBQA на будь-якому об'ємі документів (рис. 1.1):

— розбийте базу знань на фрагменти тексту;

— створіть числове представлення (вбудовування) для кожного блоку та збережіть їх у векторній базі даних;

— якщо ваші дані статичні, кроки 1 і 2 виконуються одноразово;

— запустіть семантичний пошук, використовуючи запит користувача в цій базі даних, і отримайте відповідні фрагменти тексту;

— надішліть ці фрагменти тексту LLM разом із запитаннями користувача та попросіть надати відповідь.

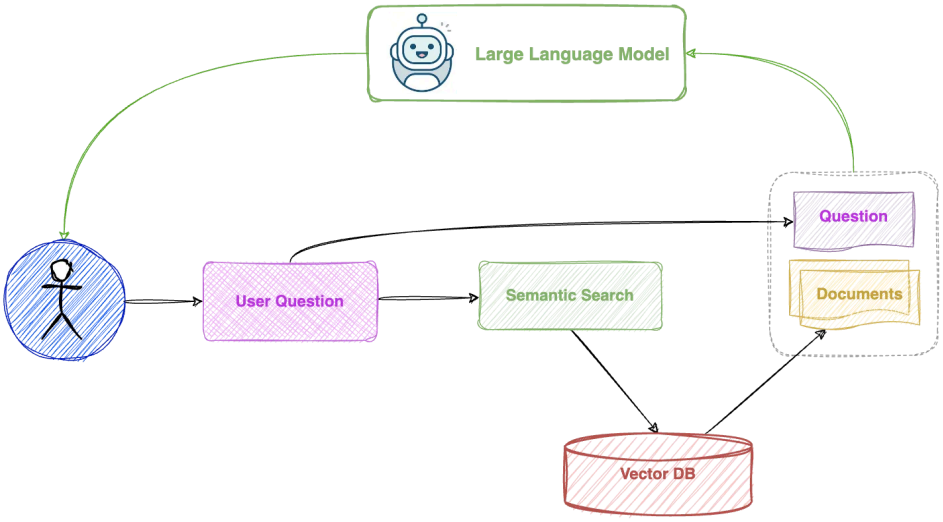


Рисунок 1.1 — Ілюстрація архітектури системи отримання відповідей

Цей підхід добре працює для простих запитань до простої бази фактичних знань. Однак це не працює для складнішої бази знань і складніших запитань, які вимагають глибшого, багаторазового (багатокрокового) обґрунтування. Під обґрунтування з кількома переходами ми розуміємо процес, у якому виконується кілька кроків логічного або контекстуального висновку, щоб дійти відповіді на запитання.

Автор у статті [2] запропонував ідею створення автономного дослідницького агента на базі штучного інтелекту, який може відповідати на складні запитання за допомогою глибоких можливостей багаторазового обґрунтування. Можна розглянути декілька підходів до створення агента (рис 1.2):

— агент ReAct, що використовує стиль міркування "ReAct" (причина та дія), щоб вирішити, який інструмент використовувати для вирішення поставленої проблеми;

— агент Self-Ask, ставить додаткові запитання на основі початкового запитання, а потім намагається знайти проміжні відповіді, використовуючи ці проміжні відповіді, він нарешті приходить до остаточної відповіді.

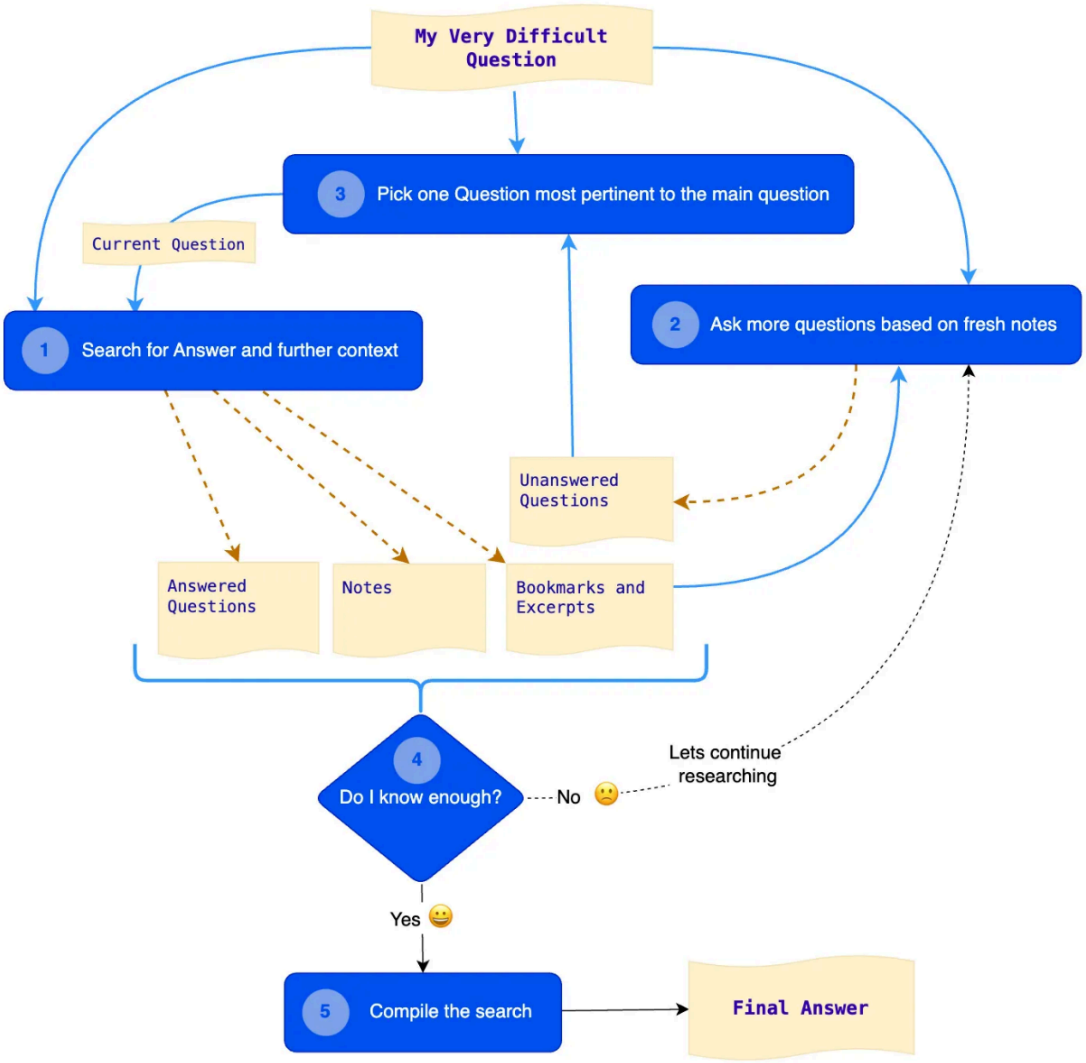


Рисунок 1.2 — Графічне зображення цього процесу роботи агента

Стаття [3] присвячена створенню графа знань на базі великих об'ємів текстової інформації за допомогою LLM (велика мовна модель).

Графи знань (KG) або будь-які інші графи складаються з вузлів і ребер. Кожен вузол KG представляє концепцію, а кожне ребро є зв’язком між парою таких концепцій. Графи знань корисні з багатьох причин. Ми можемо запускати алгоритми графів і обчислювати центральні точки для будь-якого вузла, щоб зрозуміти, наскільки важливою є концепція (вузол) для основної роботи. Ми можемо аналізувати пов’язані та непов’язані набори понять або обчислювати спільноти понять для глибокого розуміння

предмета. Ми можемо зрозуміти зв'язки між, здавалося б, непов'язаними концепціями. Ми також можемо використовувати графи знань, щоб реалізувати Graph Retrieval Augmented Generation (GRAG або GAG) і спілкуватися з нашими документами. Це може дати нам набагато кращі результати, ніж звичайна стара версія RAG, яка має кілька недоліків. Наприклад, пошук контексту, який є найбільш релевантним для запиту, за допомогою простого пошуку семантичної подібності не завжди ефективний. Особливо, коли запит не надає достатнього контексту щодо свого справжнього наміру або коли контекст складається з фрагментів великого корпусу тексту.

Давайте розглянемо наступний текст:

Mary had a little lamb,
You've heard this tale before;
But did you know she passed her plate,
And had a little more [3]*!*

А нижче одне з можливих представлень цього тексту як KG (рис. 1.3).

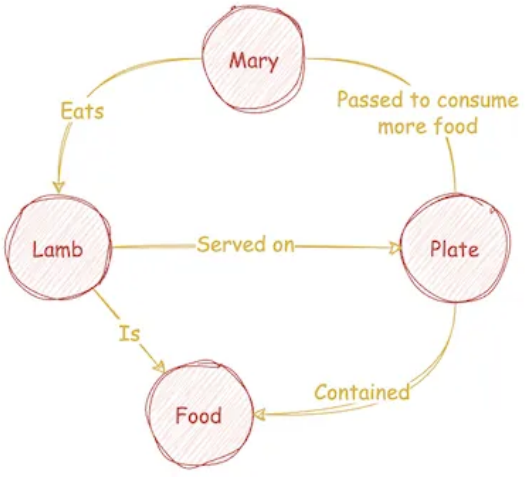


Рисунок 1.3 — Приклад графу знань

Автор запропонував алгоритм пошуку концепцій та відносин відношень між ними (рис. 1.4).

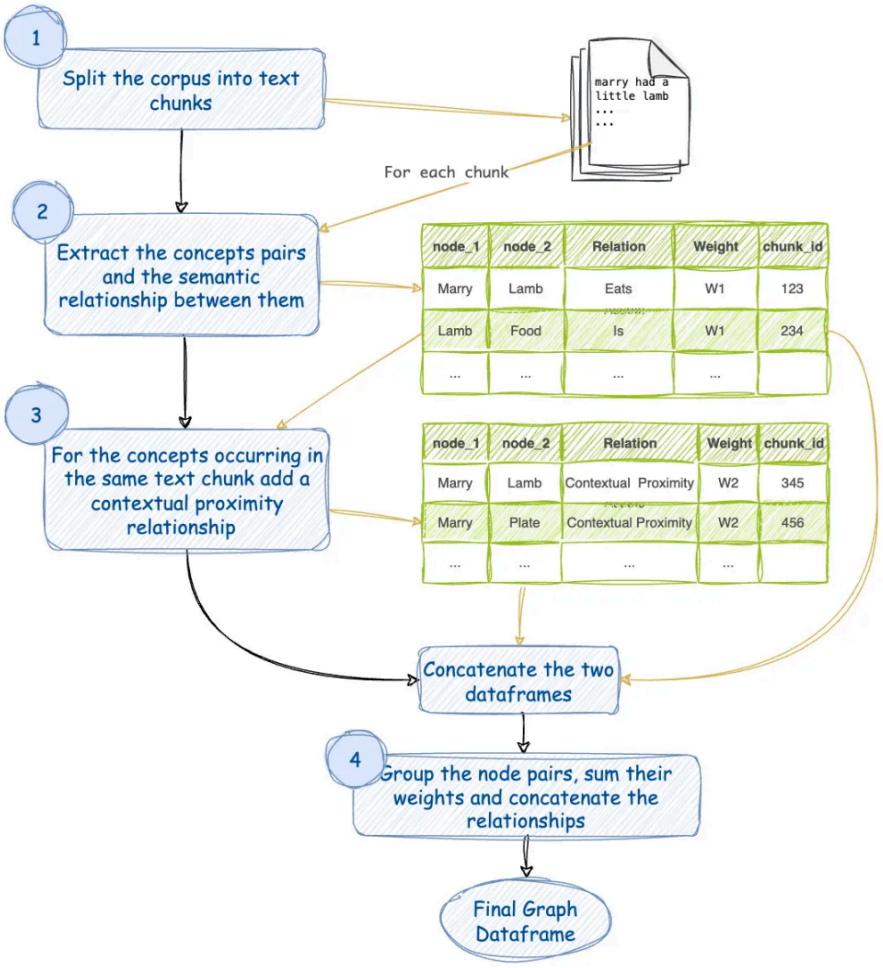


Рисунок 1.4 — Ілюстрація алгоритму рішення

На першому етапі ми розбиваємо текст на порції для подальшої для подальшого процесингу за допомогою LLM. На другому етапі, за допомогою LLM, в кожній із порцій тексту ми виділяємо основні концепції та зв'язки між ними. На третьому етапі ми оприділяємо контекстуальну близькість концепцій з в рамках однієї порції тексту, це дозволяє нам додатково пов'язати концепції між собою. На четвертому етапі ми будуємо фінальний граф.

Загально, стаття хоч і дає алгоритм, який в основному використовується для даної задачі, але його можна покращити, бо на даний момент він одноразовий, так як відсутнє повноцінне зберігання даних, що можна допрацювати, наприклад використанням бази даних, де ми можемо зберігати аналіз кожної статті, та кожен елемент аналізу, для того щоб створити один загальний граф.

Надалі в цьому розділі розглянемо всю доступну теоретичну інформацію, що може знадобитись для реалізації цього алгоритму, або його покращення. Почнемо огляд з головного – виду промптингу.

1.2 Огляд видів промптингу

З джерелом [4] зосередимось на промптингу, а саме на методах Ланцюг Думок (CoT), Дерево Думок(ToT), та Пошуково покращена генерація(RAG). А почнемо з того, що таке промптинг, це процес структурування тексту, який може бути інтерпретовним та зрозумілим для моделі породжувального ШІ. В загальному є багато методологій, але найбільш поширеними є вищеназвані CoT, ToT, та RAG, тож пройдемось по ним.

1.2.1 Chain of Thought

Ланцюг думок (англ. Chain of Thought) — це вид промптингу, який дозволяє складні можливості мислення через проміжні кроки мислення. Його можна поєднати з кількома запитами для отримання кращих результатів на

більш складних завданнях, що вимагають мислення перед відповіддю (рис. 1.3).

Prompt:

```
The odd numbers in this group add up to an even number: 4, 8, 9, 15, 12, 2, 1.
A: Adding all the odd numbers (9, 15, 1) gives 25. The answer is False.
The odd numbers in this group add up to an even number: 17,  10, 19, 4, 8, 12, 24.
A: Adding all the odd numbers (17, 19) gives 36. The answer is True.
The odd numbers in this group add up to an even number: 16,  11, 14, 4, 8, 13, 24.
A: Adding all the odd numbers (11, 13) gives 24. The answer is True.
The odd numbers in this group add up to an even number: 17,  9, 10, 12, 13, 4, 2.
A: Adding all the odd numbers (17, 9, 13) gives 39. The answer is False.
The odd numbers in this group add up to an even number: 15, 32, 5, 13, 82, 7, 1.
A:
```

Output:

```
Adding all the odd numbers (15, 5, 13, 7, 1) gives 41. The answer is False.
```

Рисунок 1.3 — Приклад CoT з джерела

Як можна побачити, ітеративно, штучний інтелект доходить до правильного висновку, але в цьому і є ключова проблема використання цього промпту для задачі з побудовою графу знань — не оптимальне рішення. Штучний інтелект покроково проведе аналіз, але це не оптимальне рішення, так як аналіз довгого тексту займатиме купу часу, окрім цього, аналіз тексту хоч і є покроковою задачею, але це задача, яка полягає в розбитті на менші частини, де ми розбиваємо текст, на уривки, уривки на речення і аналізуємо кожне, тобто аналіз речень є ключовою задачею, яку важко виконати покроково, окрім цього, покращення результатів відбувається методом "Питання/Відповідь", що є проблематичним для тексту, в якому є хоча б декілька сторінок.

1.2.2 Tree of Thought

Дерево думок (англ. Tree of Thought) — промптинг потрібний для складних завдань, які потребують дослідження або стратегічного попереднього перегляду, традиційні або прості методи натхнення не вистачають "Дерево Думок" (Tree of Thoughts, ToT) — нещодавно запропонований фреймворк, який узагальнює метод ланцюжка думок та спонукає до дослідження думок, які служать проміжними кроками для загального вирішення проблем з допомогою мовних моделей (рис. 1.4).

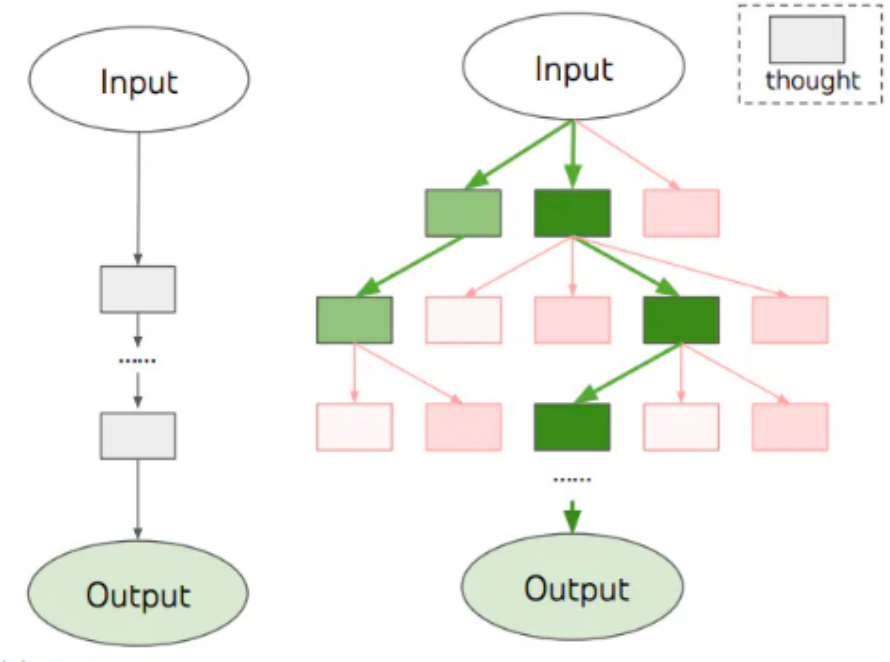


Рисунок 1.4 — Ілюстрація CoT та ToT

ToT зберігає дерево думок, де думки представляють собою послідовності зв'язного мовлення, які служать проміжними кроками до вирішення проблеми. Цей підхід дозволяє мовній моделі самооцінювати прогрес через проміжні думки, зроблені у напрямку вирішення проблеми за допомогою обдуманого процесу мислення. Здатність мовної моделі

генерувати та оцінювати думки поєднується з алгоритмами пошуку (наприклад, пошук у ширину та у глибину), щоб забезпечити систематичне дослідження думок з попереднім переглядом та відстеженням.

Метод ToT хоч і вирішує проблему з постійними питаннями від штучного інтелекту, тому що, завдяки методі з "незалежними експертами" сам собі відповідає на правильність, але не є ідеальним, так як завжди існує можливість того, що перевірка запропонує неправильний варіант, наприклад замість "вовк з'їв вівцю" вийде результат у вигляді "вівця з'їла вовка", або з'являться додаткові зв'язки в графі, але загально метод непоганий, і його можна використати, тому що вищенаведені помилки, можуть бути і в інших методах.

1.2.3 Retrieval Augmented Generation

Мовні моделі загального призначення можуть бути налаштовані для досягнення кількох загальних завдань, таких як аналіз настрою та розпізнавання іменованих сутностей. Для цих завдань загалом не потрібні додаткові знання.

Для більш складних та насичених знаннями завдань можна створити систему на основі мовної моделі, яка отримує доступ до зовнішніх джерел знань для виконання завдань. Це дозволяє досягти більшої фактологічної стабільності, покращує надійність генерованих відповідей та допомагає зменшити проблему "галюцинацій".

Дослідники Meta AI представили метод, який називається "Пошуково покращена генерація" (Retrieval Augmented Generation, RAG), для вирішення таких завдань, що потребують знань. RAG поєднує компонент вилучення

інформації з моделлю генерації тексту. RAG може бути додатково налаштований, і його внутрішні знання можна змінити ефективним способом, не потребуючи повного перенавчання всієї моделі.

RAG отримує вхід і вилучає набір відповідних/підтримуючих документів за допомогою джерела (наприклад, Вікіпедії). Документи поєднуються як контекст з вихідним запитом і подаються на вхід генератору тексту, який створює кінцевий результат. Це робить RAG адаптивним для ситуацій, де факти можуть змінюватися з часом. Це дуже корисно, оскільки параметричні знання мовних моделей є статичними. RAG дозволяє мовним моделям уникнути перенавчання, забезпечуючи доступ до останньої інформації для генерації надійних результатів за допомогою генерації на основі вилучення.

У 2021 було запропоновано загальну рецептуру налаштування RAG. Попередньо навчена модель seq2seq використовується як параметрична пам'ять, а щільний векторний індекс Вікіпедії використовується як непараметрична пам'ять. Нижче наведено огляд того, як працює цей підхід (рис. 1.5):

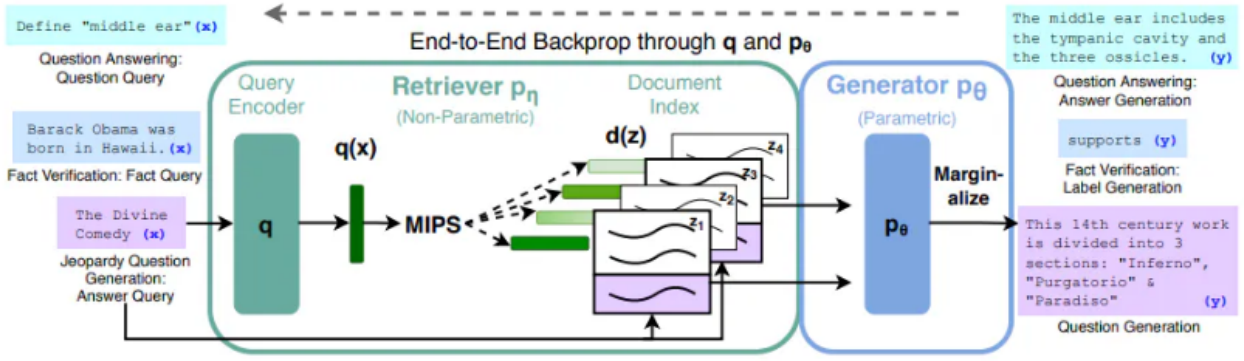


Рисунок 1.5 — Іллюстрація роботи RAG

RAG генерує відповіді, які є більш фактичними, конкретними та різноманітними. Це демонструє потенціал RAG як життєздатної опції для покращення результатів мовних моделей в знаннями насичених завданнях.

Загалом RAG підходить для даної задачі, так як його покращена можливість розпізнавання є дуже корисною в даній ситуації, зокрема можливість розширення бази знань є вирішаючим фактом, що дуже сильно нам допоможе, так як дозволить використовувати побудований штучний інтелект, для вирішення будь-якої задачі, яка пов’язана з окремими базами знань, таких як медицинські, або пов’язані з фізикою, де є обширна термінологія, а у випадку медицинської, ще й з латиницею, хоча навіть зі своїми плюсами, він всеодно може мати ті ж проблеми, що й інші види промптингу, бо хоч, і кількість "галюцинацій" зменшено, але не прибрано. Ключовим плюсом RAG є оптимізованість відносно інших промптингів, через відсутність додаткових процесів, які є наприклад у CoT та ToT.

1.2.4 Graph Retrieval Augmented Generation

Після огляду головних видів промптингу можемо перейти до огляду GRAG (графова пошуково покращена генерація) у статті [5].

Графова пошуково покращена генерація (Graph retrieval-augmented generation, GraphRAG) набирає обертів і стає потужним додатком до традиційних методів пошуку за векторами. Цей підхід використовує структуровану природу графових баз даних, які організовані у вигляді вузлів та відносин, для підвищення глибини та контекстуальності отриманих даних.

Графи чудово відображають та зберігають гетерогенну та взаємопов'язану інформацію у структурованому вигляді, легко захоплюючи складні відносини та атрибути по різноманітних типах даних (рис. 1.6). У порівнянні з цим, векторні бази даних часто мають проблеми з такою структурованою інформацією, оскільки їхній потенціал полягає у роботі з неструктурованими даними за допомогою високовимірних векторів. У програмі RAG можна поєднати структуровані графові дані з пошуком за допомогою векторів у неструктурованому тексті, щоб досягти найкращого з обох світів.

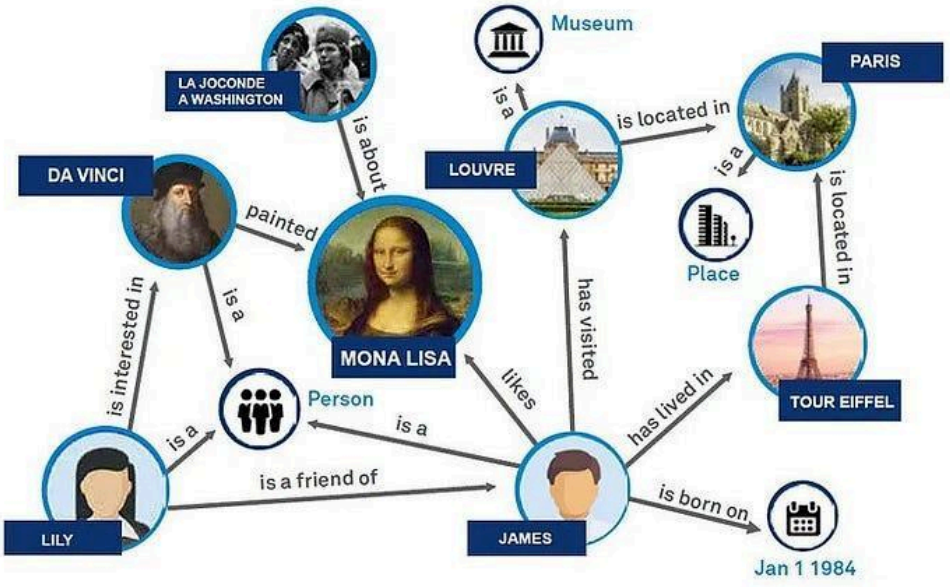


Рисунок 1.6 — Приклад графу

Пошук у самому графі працює за принципом вибору сусідніх вершин, наприклад при пошуку з питанням "Що з’їв вовк", ми ідемо за вершиною "вовк", та оглядаємо сусідні вершини, ми можемо отримати результати "вовк з’їв півня", "вовк з’їв вівцю", "вовк з’їв квітку", якщо вони напряму пов’язані зв’язком "з’їв".

Паралельно з цим буде відбуватись пошук за допомогою RAG з використанням пошуку за ключовим словом та векторного пошуку (рис. 1.7), що дозволить отримати найточніші результати.

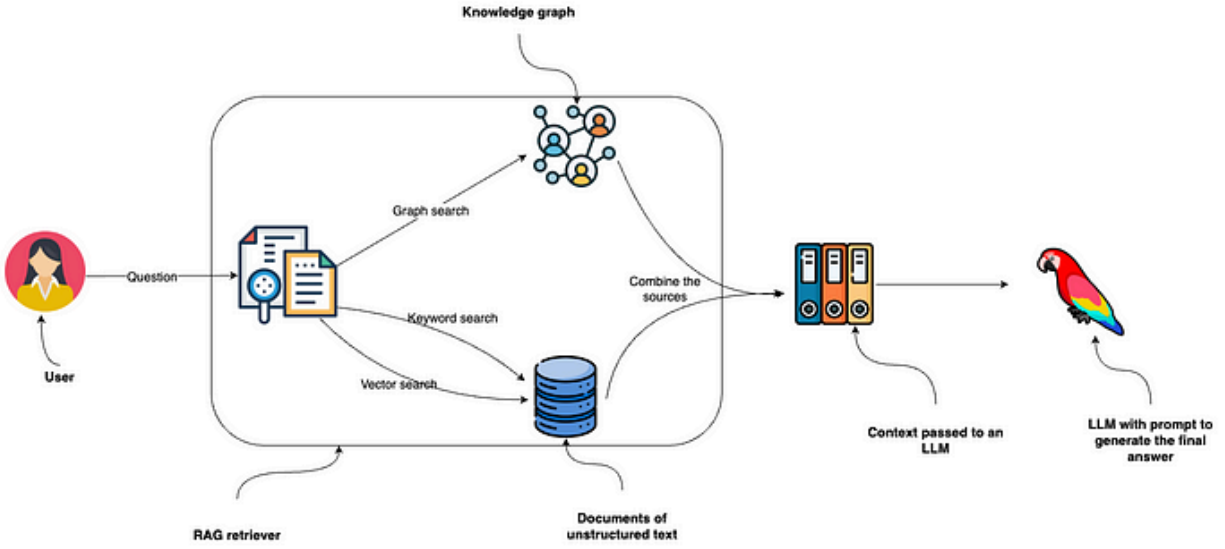


Рисунок 1.7 — Ілюстрація пошуку

Реалізація такого пошуку в порівнянні зі звичайним використанням RAG є складним процесом, але який винагороджується більшою точністю, а також більшою доступністю графу, для користувача, так як не доведеться очима вишукувати зв’язки в графі, зокрема стаття виокремлює бібліотеку Neo4j, яка дозволить побудувати граф, а ще має додатковий функціонал для даної задачі, наприклад взаємодію з промптингом, якого немає у бібліотеці Pyvis наведеній у статті [3].

Після огляду видів промптингу, варто розглянути додатковий теоретичний базис, який краще допоможе зрозуміти краще, як працює промптинг, а також ознайомить з інструментами, які використовує пошук, наприклад BMM та векторний пошук.

1.3 Великі Мовні Моделі

У статті [6] ми можемо побачити загальний огляд ВММ.

Мова є фундаментальним аспектом людського існування, і зі зростанням обсягу даних, що генеруються в Інтернеті, стає більш важливим, ніж будь-коли раніше розвивати ефективні інструменти для обробки та розуміння природної мови. Тут на допомогу приходять великі мовні моделі (Large Language Models, LLMs). ВММ — це системи штучного інтелекту, які обробляють та генерують текст, який дуже нагадує людську мову, що свідчить про значний прогрес у галузі обробки природної мови (Natural Language Processing, NLP) [6].

Ці моделі машинного навчання здатні обробляти величезні обсяги текстових даних та генерувати дуже точні результати. Вони побудовані за допомогою складних алгоритмів, таких як трансформаційна архітектура, які аналізують та розуміють патерни в даних на рівні слова. Це дозволяє LLMs краще розуміти нюанси природної мови та контекст, в якому вона використовується. З їх здатністю обробляти та генерувати текст на небачену раніше шкалу, LLMs стали все більш популярними для широкого спектру застосувань, включаючи переклад мови, чат-ботів та класифікацію текстів.

1.3.1 Поняття мовних моделей

Великі мовні моделі — це базисні моделі, які використовують глибоке навчання у завданнях обробки природної мови та генерації природної мови. Вони призначені для вивчення складності та зв'язків мови, передбачаючи

попереднє навчання на величезних обсягах даних. Це попереднє навчання включає такі техніки, як налаштування, навчання у контексті та нульове/одинарне/кілька-зразкове навчання, що дозволяє цим моделям адаптуватися до певних конкретних завдань. Основна задача ВММ – це розуміння та передбачення тексту [6].

Трансформаційна архітектура є ключовою складовою великих мовних моделей і ґрунтується на механізмі, званим самоувага, який дозволяє моделі оцінювати важливість різних слів або фраз у заданому контексті. Цей механізм дозволяє моделі звертати увагу на різні частини вхідної послідовності для обчислення представлення для кожної позиції і довів свою ефективність у захопленні віддалених залежностей та розумінні нюансів природної мови [6].

1.3.2 Види мовних моделей

Враховуючи фактор великого обсягу даних, який може вивчити будь-яка велика мовна модель, та різні підходи до її навчання можна виокремити види мовних моделей.

1. Моделі авторегресійного (AR) мовлення є типом мовних моделей, де модель передбачає наступне слово у послідовності на основі попередніх слів. З урахуванням контексту ці моделі навчаються передбачати ймовірність кожного слова в навчальному наборі даних. Ця модель прямого поширення передбачає майбутні слова на основі заданого набору слів у контексті. Однак контекстні слова обмежені двома напрямками — або вперед, або назад, що обмежує їхню ефективність у розумінні загального контексту речення або

тексту. Хоча моделі AR корисні в генеративних завданнях, які створюють контекст наперед, вони мають обмеження. Модель може використовувати лише минулий або майбутній контекст, але не обидва одночасно. Це обмежує її здатність розуміти контекст та повністю робити точні передбачення, що впливає на загальну ефективність моделі.

2. Моделі автоенкодера для мовлення — це тип архітектури нейронних мереж, який використовується в обробці природної мови (NLP) для створення фіксованого векторного представлення або вбудовування вхідного тексту, відтворюючи оригінальний вхід зі змаскованої або пошкодженої версії. Цей тип моделювання ґрунтується на ідеї, що хороше представлення вхідного тексту може бути вивчено, передбачаючи відсутні або змасковані слова в вхідному тексті за допомогою навколишнього контексту. Моделі автоенкодування зазвичай використовуються для коротких текстових вхідних даних, таких як пошукові запити або описи продуктів. Вони можуть точно створювати векторні представлення вхідного тексту, що дозволяє моделям ОПМ краще розуміти контекст та значення тексту. Це особливо корисно для завдань, які вимагають розуміння контексту, таких як аналіз настрою, де настрій речення може сильно залежати від навколишніх слів. Узагальнюючи, моделювання мовлення автоенкодером — це потужний інструмент у ОПМ для створення точних векторних представлень вхідного тексту та покращення ефективності різних завдань ОПМ.
3. Гібридні мовні моделі поєднують переваги авторегресійних та автоенкодерних моделей в обробці природної мови. Авторегресійні моделі генерують текст на основі контексту введення, передбачаючи наступне слово в послідовності, виходячи з попередніх слів, тоді як

автоенкодерні моделі навчаються генерувати фіксоване векторне представлення введеного тексту, відтворюючи оригінальний ввід зі змаскованої або пошкодженої версії. Однією з ключових переваг гібридних моделей є їх здатність балансувати зв'язність та різноманіття у згенерованому тексті. Вони можуть створювати зв'язний та різноманітний текст, що робить їх корисними для різних застосувань, таких як чат-боти, віртуальні помічники та генерація контенту. Дослідники та практики також цінують гібридні моделі за їх гнучкість, оскільки їх можна налаштовувати для конкретних завдань, що робить їх популярним вибором в галузі обробки природної мови.

Враховуючи можливості кожної ВММ найкраще підійде гібридна, з її здатністю генерувати текст, задля реалізації пошуку, а також через фактор гарного розуміння тексту, бо без цього не вийде нормальна реалізація графу, крім видів мовних моделей варто ще розглянути їх параметри.

1.3.3 Параметри мовних моделей

Токенізація — це фундаментальний процес в обробці природної мови, що включає розділення послідовності тексту на менші значущі одиниці, відомі як токени. Ці токени можуть бути словами, підсловами або навіть символами, залежно від вимог конкретного завдання ОПМ. Токенізація допомагає зменшити складність текстових даних, що полегшує їх обробку та розуміння моделями машинного навчання.

Вбудування — це важлива складова великих мовних моделей, яка дозволяє відображати слова або токени у щільні вектори низького виміру. Ці

вектори кодують семантичне значення слів у послідовності тексту та навчаються під час процесу навчання. Процес навчання вбудувань полягає в налаштуванні ваг нейронної мережі на основі вхідної текстової послідовності так, щоб отримані векторні представлення відображали відносини між словами.

Механізми уваги в ВММ дозволяють моделі селективно зосереджуватися на конкретних частинах вхідних даних, залежно від контексту задачі. Механізми самоуваги, використані в моделях на основі трансформаторів, працюють за допомогою обчислення скалярного добутку між одним токеном у послідовності та іншими токенами, що призводить до матриці ваг, що представляють важливість кожного токена в порівнянні з кожним іншим токеном у послідовності

Загалом задача полягає у використанні ВММ гібридної форми та покращення її з використанням RAG,оскільки цей алгоритм, потрібен для GRAG і реалізації пошуку, як було наведено вище, інші варіанти промптингу не є ефективними для цієї задачі, через відсутність підключення додаткових масивів інформації та недостатню роботу з контекстом матеріалу.

1.4 Векторний пошук

Стаття [7] зосереджена на пояснені принципу роботи та векторного пошуку.

Векторний пошук, заснований на машинному навчанні та обробці природної мови, представляє собою зміну парадигми у пошуку інформації. Цей трансформаційний підхід захоплює значення та контекст неструктурованих даних, перетворюючи їх на числові представлення.

Використовуючи алгоритми наближених найближчих сусідів (ANN), векторний пошук відзначається у семантичних пошуках, забезпечуючи швидші та більш відповідні результати порівняно з традиційними пошуками за ключовими словами [7].

Векторний пошук потрібен в задачах, де потрібно знайти щось тільки за уявленням про предмет, але не знаючи його назви. Векторний пошук подолає це виклик, дозволяючи користувачам шукати на основі значень, а не лише ключових слів. Завдяки включенню векторних вбудовувань, які виходять за межі тексту і включають в себе відео, зображення та аудіо, цей підхід покращує досвід пошуку, пропонуючи більший і точніший пошук інформації.

На відміну від традиційних методів, які базуються на ключових словах та частоті слів, пошукові двигуни векторного пошуку працюють у просторі вбудовування, використовуючи відстані для представлення схожості. Цей інноваційний підхід дозволяє користувачам знаходити пов'язані дані, шукаючи найближчих сусідів запиту, подолуючи обмеження традиційних методів пошуку (рис.1.8).

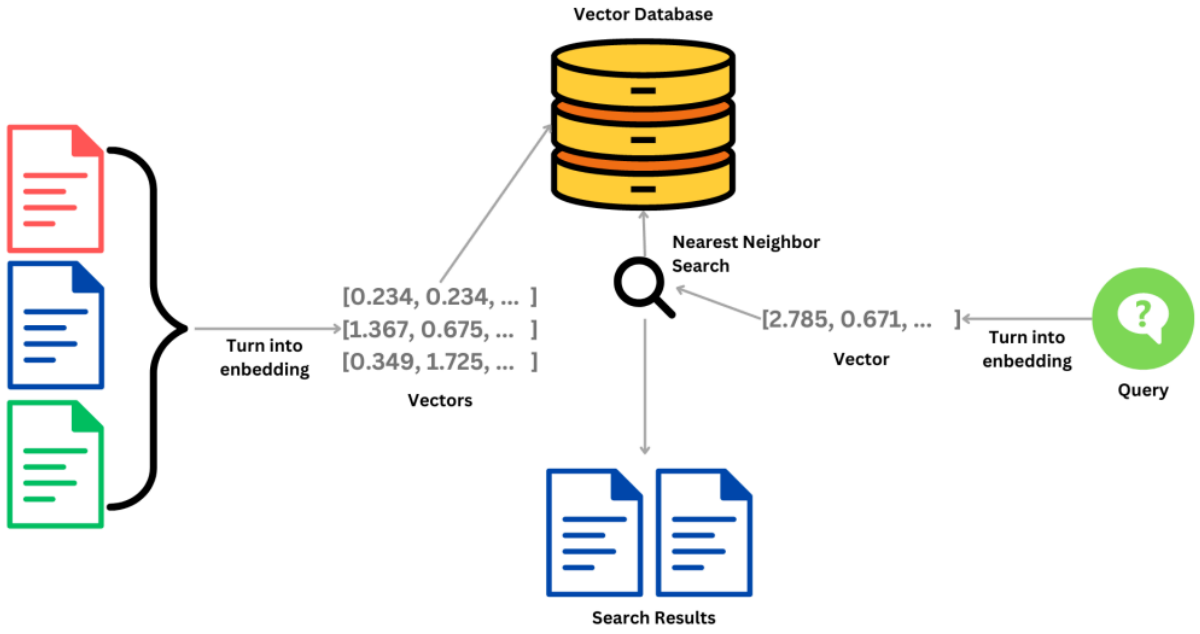


Рисунок 1.8 — Ілюстрація роботи векторного пошуку

Векторний пошук базується на нижченаведених елементах, які в свою чергу є фундаментальними для промптингу і часто використовуються у RAG.

Векторне вбудування: числові представлення даних та відповідного контексту, збережені у високовимірних векторах, які можуть бути навчені на величезних наборах даних для підвищення точності.

Оцінка схожості: основна ідея, що вектори схожих даних та документів будуть демонструвати схожість, що дозволяє ефективно виявляти пов'язане вміст.

Алгоритм ANN: використання алгоритмів наближених найближчих сусідів, які жертвують ідеальною точністю для ефективного виконання у багатовимірних просторах вбудування на великій шкалі.

Загально в статті описується достатня теоретична база про векторний пошук, але не показується його прямий зв’язок з RAG, який ми зможемо побачити у наступній статті.

Після аналізу ключових елементів штучного інтелекту ми можемо перейти до методів покращення його роботи, наприклад подрібнення тексту.

1.5 Розбиття тексту

Стаття [8] зосереджена поясненні принципу того, як правильно розбити текст, і як це покращує роботу RAG та великих мовних моделей. ВММ, хоча здатні генерувати текст, який є як значущим, так і граматично правильним, страждають від проблеми, відомої як галюцинація.

Галюцинація в ВММ — це концепція, де ВММ впевнено генерують неправильні відповіді, тобто вони вигадують неправильні відповіді таким чином, що ми вважаємо їх правдивими. Це була серйозна проблема з моменту появи ВММ. Ці галюцинації призводять до неправильних та фактично неправильних відповідей. Тому було введено пошуково покращену генерацію (RAG).

У RAG ми беремо список документів/частин документів і кодуємо ці текстові документи у числове представлення, яке називається векторні вбудовування, де одне векторне вбудовування представляє один шматок документа і зберігає їх у базі даних, що називається векторним сховищем. Моделі, потрібні для кодування цих частин у вбудовування, називаються кодувальними моделями або бі-кодерами. Ці кодери навчені на великому корпусі даних, що робить їх достатньо потужними для кодування частин документів у векторне представлення.

Пошук в значній мірі залежить від того, як частини проявляються і зберігаються в векторному сховищі. Знаходження вірного розміру частин для будь-якого даного тексту — це дуже складне питання в цілому.

Покращення пошуку може бути здійснено за допомогою різних методів пошуку. Але також може бути здійснено за допомогою кращої стратегії подрібнення. Є декілька варіантів подрібнення, але вони всі потрібні для різного роду задач, про них далі.

Фіксовано розмірне подрібнення: Це найбільш поширений і прямолінійний підхід до подрібнення: ми просто вирішуємо кількість токенів у нашій частині і, за бажанням, чи має бути між ними яка-небудь перекриття. Загалом ми хочемо зберегти деяке перекриття між частинами, щоб переконатися, що семантичний контекст не загубився між частинами. Подрібнення фіксованого розміру буде найкращим варіантом у більшості випадків. Порівняно з іншими формами подрібнення, подрібнення фіксованого розміру є обчислювально дешевим і простим у використанні, оскільки не потребує використання жодних бібліотек обробки природної мови.

Рекурсивне подрібнення: Рекурсивне подрібнення розбиває вхідний текст на менші частини ієрархічним та ітеративним способом за допомогою набору роздільників. Якщо перша спроба розділити текст не дає частин потрібного розміру або структури, метод рекурсивно викликає себе для отриманих частин з іншим роздільником або критерієм, поки не буде досягнуто потрібного розміру або структури частин. Це означає, що хоча частини не будуть точно одного розміру, вони все ще будуть "прагнути" бути схожого розміру. Використовує те, що добре у фіксованому розмірі частин і перекритті.

Подрібнення за специфікою документа: Враховує структуру документа. Замість використання фіксованої кількості символів або рекурсивного процесу, створює частини, які відповідають логічним розділам документа, таким як абзаци або підрозділи. Це допомагає зберегти організацію автора контенту, зберігаючи текст узгодженим. Це робить отриману інформацію

більш актуальною та корисною, особливо для структурованих документів з чітко визначеними розділами. Він може обробляти формати, такі як Markdown, Html і т.д.

Семантичне подрібнення: Розглядає відносини в тексті. Розбиває текст на значущі, семантично повні частини. Цей підхід забезпечує цілісність інформації під час витягування, що призводить до більш точного і контекстно відповідного результату. Це повільніше порівняно з попереднім підходом до подрібнення.

Агентне подрібнення: гіпотеза тут полягає в тому, щоб обробляти документи так, як це робили б люди.

Ми починаємо з верху документа, розглядаючи першу частину як частину.

Ми продовжуємо далі вниз по документу, вирішуючи, чи нове речення або частина інформації належить до першої частини чи має почати нову.

Ми продовжуємо це далі, поки не дійдемо до кінця документа.

Цей підхід все ще тестується і ще не готовий для загального використання через час, необхідний для обробки кількох викликів LLM та вартість цих викликів. Поки що в загальнодоступних бібліотеках немає реалізації.

У даній задачі нас цікавить фіксовано розмірне подрібнення, через його оптимальність, та відсутність специфічних умов для подрібнення тексту.

1.6 Реалізація пошуку

Надалі розглянемо методи та статті, які можуть покращити роботу алгоритму у статті [3], наприклад імплементацію бази даних, або внесення ієрархії.

Стаття [9] демонструє реалізацію RAG з використанням приватної ВММ, що на мою думку є не оптимальним, через фактор того, як створюється персональна ВММ.

Персональна ВММ має свої переваги, завдяки факту того, що її можна створювати під свої вимоги, а також не використовувати сторонні сервіси, але є декілька інших мінусів. Першим мінусом є специфіка навчання, відбувається воно завдяки закачці набору даних, який потрібен для виконання, і чим складніше завдання, тим більша база потрібна, особливо, коли існує потреба у використанні медичної бази знань, і може виникнути проблема, при зустрічі латиниці у тексті, що може викликати конкретні проблеми в подальшому. Друга проблема – складність навчання, а саме факт того, що закачка даних — процес не швидкий, тому є проблематичним, так як на використання для нової обширної бази знань доведеться зачекати годину для використання.

Тим не менш, стаття пропонує цікаве рішення по базам даних, яке можна використати в подальшому, для можливості повторного використання даних, які ми вже обробили, а ще допоможе з реалізацією пошуку. Отже, є потреба у базах даних, щоб ми могли зберігати ці вбудовування та ефективно шукати їх. Отже, для зберігання та пошуку ми потребуємо векторні сховища. Через це можемо отримувати векторні вбудовування, які будуть "найбільш схожими". Основна ідея полягає у тому, що воно робить векторний пошук автоматично.

Стаття розглядає варіант реалізації за допомогою бібліотеки LangChain та векторного сховища FAISS (рис 1.9).

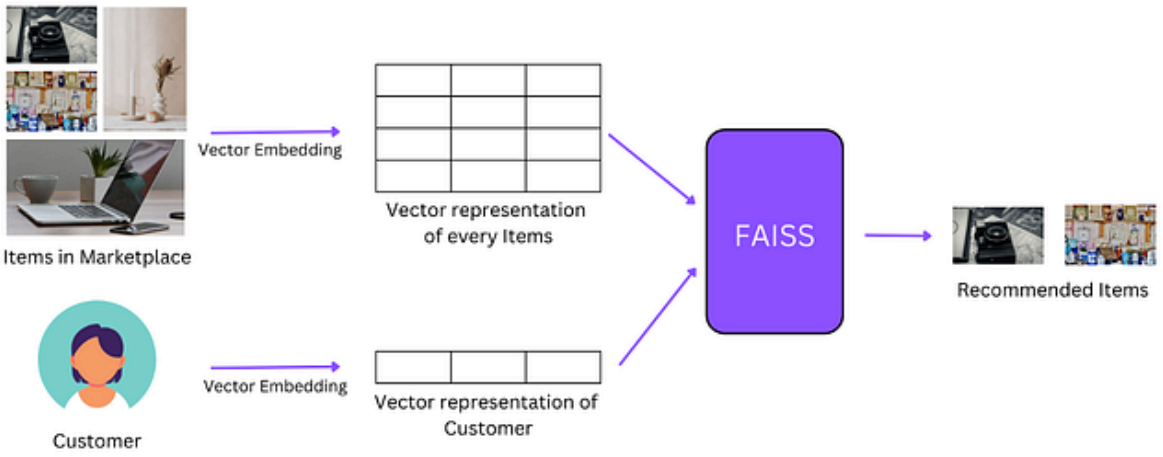


Рисунок 1.9 — Схема векторного сховища

Реалізація пошуку за допомогою баз даних є рішенням, яке дає велику перевагу, для користувача, так як створює можливість вкоротити використаний для розуміння час, бо прибирає потребу в пошуку інформації в графу, а також при правильній реалізації дозволяє створити статичне сховище, що оптимізує час роботи програми.

Векторний пошук є корисним для штучного інтелекту, так як при запиті, навідміну від пошуку за ключовим словом, ми вицілюємо не найближче слово за синтаксисом, а вицілюємо семантику.

1.7 Теоретична основа побудови ієрархії

Перейдемо до статті [10] в якій демонструється створення ієрархії.

Теорія Ієрархії Мережі знань (HNC) вносить нову думку щодо машинного перекладу, який розглядається як складне відображення. HNC може бути розділений на такі три відображення:

G1 — група речень джерельної мови => група речень джерела;

G2 — група речень джерельної мови => група речень цільової мови;

G3 — група речень цільової мови => група речень цільової мови.

G1 — це відображення, отримане з концептуального простору джерельної мови, яке відповідає процесу розуміння системи перекладу. Група речень джерельної мови (SGSL) відображається на групу речень джерела (SGS). G2 — це відображення з джерельної мови на цільову мову на основі концептуального простору мови. Воно відповідає процесу перекладу системи перекладу. Група речень джерела (SGS) відображається на групу речень цільової мови (SGT). G3 — це відображення з концептуального простору на простір цільової мови. Це процес мови. Група речень цільової мови (SGT) відображається на групу речень цільової мови (SGTL). Ці три відображення ґрунтуються на формі концептуального простору мови. Комп'ютер може повністю обробляти природну мову через кілька примітивів концептуального простору мови, таких як концепт, категорія речення та одиниця контексту[10].

У теорії HNC, концепт є нескінченним, тоді як елементи концепту є скінченними; скінчений концепт можна виразити за допомогою скінченних елементів концепту. Основна концепція одиниці, основна концепція та логічна концепція — це три основні концепції абстракції проектування HNC. Вони відображають примітивність та систему абстрактних концепцій. Семантична мережа є ієрархічною структурою подібно дереву, кожен шар

багатьох вузлів виражається числово. Кожен вузол мережі може стартувати з вершин і визначатися унікальним номером, рядок цифр називається концептом символу HNC.

Основні концепції одиниць та логічні концепції — це три концептуальні категорії теорії HNC. Нескінченний концепт природної мови описується через ці три види концепцій. Логіка концепції зазвичай пов'язана з відповідними мовними словами, такими як прийменники та сполучники. Проектування концепції логіки спрямоване на встановлення різноманітних маркерів семантичного уривка. Вони служать аналізу категорій речення у сприйнятті семантичного уривка. Різноманітність концепцій у природній мові виражається як мовні явища. Теорія HNC описує абстрактний концепт з динамічних (v), статичних (g), властивостей (u), значень (z) та ефектів (r). Якщо слово виражає концепт з одного боку, воно буде відоме як один з п'яти концептів. Концепції взаємопов'язані, наприклад, "студент" та "школа", "автомобіль" та "дорога" [10].

1.8 Математичний апарат

HNC обчислює асоціацію між концепціями за допомогою функції кореляції концепцій. Речення семантичного уривка представлене у вигляді формули (1.1):

$$FJ = \sum_{i=0}^{m} (JK_i) + EK + \sum_{j=0}^{m} (JK_i), \quad (1.1)$$

де FJ — представляють речення типу "поки";

JK — представляють узагальнений семантичний фрагмент об’єкта;

EK — Ейгенієвський уривок.

Технологія розуміння мови HNC, яка також називається аналізом категорій речень, може бути розділена на три частини: сприйняття семантичного блоку та гіпотези речення, перевірка речення, аналіз семантичного блоку. Семантичний уривок зазвичай складається з ядра і частини, і EK не є винятком. Іншими словами, EK не є присудковим дієсловом, але є структурним тілом зі складною структурою. Ядро EK з передньої та задньої частин може мати ту частину. Ця частина називається верхами, а після цієї частини — дном. Конституція англійського EK спрямована на полегшення комп'ютерного сприйняття для підтвердження гіпотези речення EK на основі стратегії вибору перекладу. Сприйняття семантичного уривка базується на концепції динамічного (v) концепту, відомої як критерій v. Критерій v використовується як настанова для сприйняття EK. Оскільки концепція v формуватиме EK, вона надає інформацію для EK. Основним дієсловом англійської мови є концепція v,

обробка дієслова стає сприйняттям ЕК в ключовому моменті. Англійський опис частини ЕК розташований перед дієсловом, а саме, верхні ЕК поширені, а нижні — рідкісні в англійській мові. Надійність є однією з найважливіших характеристик тесту. Повторна надійність тесту може використовувати два рази результати тесту для виразу коефіцієнта кореляції між випробуваннями за допомогою наступної формули (1.2):

$$R_n = \frac{\Sigma(X-X^{'})(Y-Y^{'})}{\sqrt{\Sigma(X-X^{'})^2}\sqrt{\Sigma(Y-Y^{'})^2}}, \quad (1.2)$$

де $X^{'} = \frac{1}{n}(\sum^{n}_{i} X_i)$ — середнє значення X

$Y^{'} = \frac{1}{n}(\sum^{n}_{i} Y_i)$ — середнє значення Y

1.9 Оцінка та нюанси побудови речення

У результаті обчислень ми маємо чисельну характеристику на основі якої ми будуємо кореляції та відповідно ієрархії, а перевіряємо ми це за шкалою (рис. 1.10).

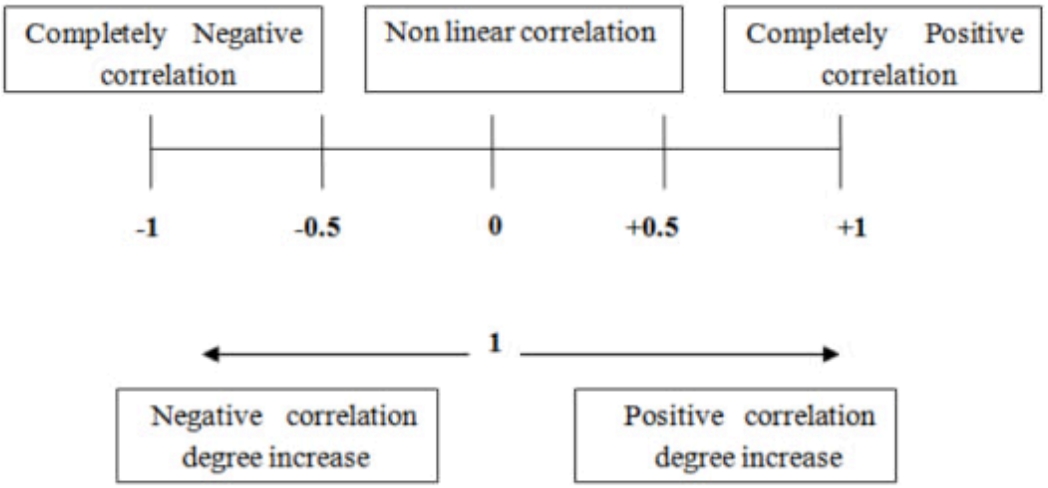


Рисунок 1.10 — Шкала кореляції

Крім того варто враховувати фактор структури звичайних речень в англійській мові, оскільки є конструкції звичайного речення.

Просте речення містить лише присудок або еквівалент дієслова у дієслівній фразі як присудок речення. Просте речення можна описати наступним чином.

Просте = Підмет + Присудок за компонентами.

Підмет = Частина підмету + Склад підмету + Прислівник.

Присудок = Модальні слова + Допоміжні + присудкове дієслово

Компоненти підмета = Об'єкт + підметний склад + обставинний елемент.

Склад підмету = іменникова фраза + участь у фразі + інфінітивна фраза.

Обставинний елемент = фраза прислівника + прийменникові фрази + розділення слова + фраза + інфінітивна фраза.

Тому для побудови ієрархії потрібно це враховувати спочатку знаходиться присудкове дієслово в реченні, потім встановлюється присудок, і після цього обирається присудок як роздільна лінія. Наостанок, формується просте речення, синтаксична семантична структура, заснована на підметі та присудку аналізу фреймів дієслів мови. Англійське речення має лише один

присудок. Присудок обирає присудкове дієслово як центр, а модальні дієслова, допоміжні дієслова та деякі присудкові дієслова тісно пов'язані з модальними прислівниками. Вони використовуються для вираження часу, зміни голосу та інших типів синтаксичних ознак, а також підмет в реченні за особою та числом для забезпечення послідовності, процес аналізу присудка полягає у просуванні назад по введеному реченню, визначаючи центр присудка за допомогою дієслів дійсних чи істинних, крім того, може бути більше одного дієслова чи істинного дієслова, яке використовується для з'єднання. Потім проводиться пошук меж присудка вперед і назад, щоб відповідати структурному шаблону присудка та визначити типи присудка. Нарешті, згідно з центральним присудковим дієсловом і структурним типом, виконується побудова сітчастого каркасу присудка та інших обмежень речення на комп'ютері [10].

Наостанок розглянемо способи перевірки точності нашого рішення.

1.10 Оцінка точності графу

Стаття [11] показує за якими параметрами ми можемо перевірити граф на точність, бо незалежно від того, з яких джерел створюється граф знань, отриманий початковий граф знань зазвичай буде неповним і часто міститиме дублікати, протиріччя або навіть неправильні висловлювання, особливо при використанні даних з кількох джерел. Після початкового створення та збагачення графа знань із зовнішніх джерел важливим кроком є оцінка якості отриманого графа знань. Перейдемо до параметрів та їх означення.

1. Точність відноситься до того, наскільки концепти та відносини, закодовані вузлами та ребрами у графі, вірно відображають реальні

явища. Точність може бути розділена на три виміри: синтаксичну точність, семантичну точність та актуальність.

1.1. Синтаксична точність — це ступінь відповідності даних граматичним правилам, визначеним для предметної області і/або моделі даних. Поширеним прикладом синтаксичної неточності є випадки взаємодії з вузлами типу даних, які можуть бути несумісними з визначеним діапазоном або бути невірно сформованими.

1.2. Семантична точність — це ступінь, до якого значення даних правильно відображають реальні явища, що може бути ускладнено нечіткими результатами екстракції, ненадійними джерелами, вандалізмом тощо.

1.3. Актуальність — це ступінь, до якого граф знань відповідає сучасному стану реального світу. Граф знань може бути семантично точним наразі, але швидко може стати неправильним (застарілим), якщо не буде встановлено процедур для його своєчасного оновлення.

2. Покриття відноситься до уникнення пропуску елементів, які мають відношення до предметної області, що в іншому випадку може призвести до неповних результатів запитів, умовних висновків, упереджених моделей тощо.

2.1. Повнота відноситься до ступеня наявності всієї необхідної інформації в певному наборі даних.

2.2. Презентабельність є пов'язаним аспектом, який, замість того, щоб акцентувати увагу на співвідношенні відсутніх елементів, пов'язаних з доменом, спрямовується на оцінку упереджень високого рівня у тому, що включено / виключено з графа знань.

3. Узгодженість відноситься до того, наскільки добре граф знань відповідає або узгоджений із формальною семантикою та обмеженнями, визначеними на рівні схеми.
    3.1. Послідовність означає, що граф знань вільний від суперечностей відносно певного логічного наслідку, який розглядається. Наприклад, якщо граф стверджує "Міша живе в Києві", то не має бути твердження "Міша живе у Львові".
    3.2. Чинність означає, що граф знань вільний від порушень обмежень, таких як ті, що зафіксовані у виразах форми.
4. Стислість відноситься до включення лише відповідного вмісту (уникнення "перевантаження інформацією"), який подано у лаконічному та зрозумілому способі.
    4.1. Стислість означає уникнення схем і елементів даних, які не мають відношення до домену.
    4.2. Демонстраційна стислість означає ступінь компактного представлення вмісту у графі знань, який може бути знову скороченим або розширеним.
    4.3. Зрозумілість — це легкість інтерпретації даних без двозначності для людей, що включає, принаймні, надання міток і описів, зрозумілих для людини (найкраще у різних мовах).

Хоч стаття і не надає точних методів з використанням формул, але дає хороший набір правил концептуальної перевірки, чого часто достатньо для перевірки графу

1.11 Висновки до розділу

У цьому розділі було розглянуто всі необхідні інструменти та пояснено їх природу і взаємодію, а також задано вектор розвитку реалізації. З двох варіантів покращення було обрано пошук, який дозволяє знаходити всю необхідну інформацію з бази даних, в яку потім буде внесено необхідні статті. Ієрархія, хоча і може мати свої переваги, не зможе надати тієї ж ефективності в оптимізації процесу, як пошук.

Пошук надає можливість швидкого доступу до інформації, незалежно від її обсягу та структури. Використовуючи пошук, користувач може швидко знайти потрібні дані без необхідності переглядати велику кількість ієрархічно організованих файлів або записів. Це суттєво прискорює процес роботи та зменшує витрати часу на пошук необхідної інформації.

З іншого боку, ієрархічна структура даних може бути корисною для організації та систематизації інформації. Однак, у випадках, коли потрібно знайти конкретні дані швидко та ефективно, пошук виявляється більш дієвим інструментом. Ієрархія може ускладнити процес пошуку, особливо якщо структура даних є складною або містить велику кількість рівнів.

Таким чином, було зроблено висновок, що використання пошуку для покращення процесу є більш оптимальним рішенням. Це дозволяє забезпечити швидкий доступ до необхідної інформації та оптимізувати робочі процеси, що є ключовим для ефективної роботи з базами даних. У результаті, пошук надає переваги, які не може забезпечити ієрархія, роблячи його кращим вибором для оптимізації процесу.

Окрім того було знайдено параметри для перевірки графу на точність, що в свою чергу надає нам можливість перевірити дані, які можна отримати після роботи коду.

РОЗДІЛ 2. АЛГОРИТМИ ПОБУДОВИ ГРАФІВ ЗНАНЬ ТА ЇХ ПЕРЕВІРКИ

З переліку методів, що було розглянуто та досліджено в розділі 1 було сформовано системний підхід щодо побудови мереж термінів на основі текстового аналізу та штучного інтелекту, що складається з наступних кроків (рис 2.1).

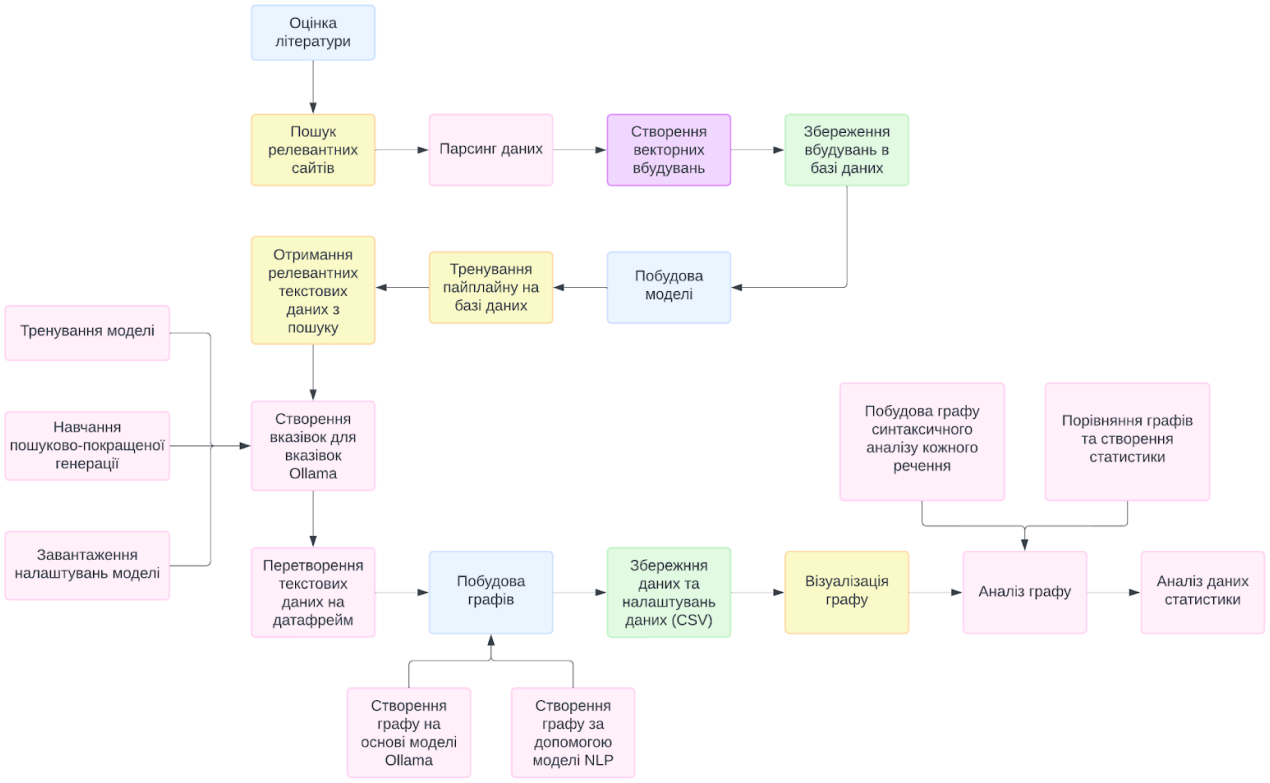


Рисунок 2.1 — Воркфлоу діаграма процесу

У ході реалізації було побудовано п’ять основних алгоритми: пошук по тексту, два підходи побудови графа, граф речення та порівняння графів. Проте варто почати з загальних даних. При реалізації більшості алгоритмів було використано модель Ollama v1.37.0, для якої було реалізовано

завантаження налаштувань, а також було прописано алгоритми для роботи з DataFrame, для яких потрібно мати розбитий на частини текст

DataFrame — це двовимірна таблична структура даних, яка часто використовується для аналізу та обробки даних. Це основний компонент бібліотеки Pandas у Python, і він нагадує електронну таблицю або SQL-таблицю. Нижче наведено деякі ключові характеристики та особливості DataFrame.

1. Рядки та стовпці: DataFrame складається з рядків і стовпців. Кожен стовпець може містити значення різних типів (числові, рядкові, логічні тощо).

2. Позначені осі: Рядки та стовпці мають мітки, що дозволяє легко отримувати доступ до даних та маніпулювати ними. Рядки позначаються індексом, а стовпці — іменами стовпців.

3. Гетерогенні дані: Стовпці в DataFrame можуть містити різні типи даних. Наприклад, один стовпець може містити цілі числа, інший — числа з плаваючою комою, а ще один — рядки.

4. Змінний розмір: Розмір DataFrame можна змінювати, тобто можна додавати або видаляти рядки та стовпці за потреби.

DataFrame — це потужний інструмент для аналізу даних і широко використовується в різних галузях, таких як дата-сайєнс, машинне навчання, фінанси тощо. Їхня гнучкість та широкі можливості, які надає Pandas, роблять їх незамінними для роботи зі структурованими даними у Python.

Почнемо огляд з алгоритмів DataFrame:

1) documents2DataFrame — це основна функція для роботи з текстом, перетворює розбитий текст на датафрейм проходячись по всім уривкам тексту;
2) df2Graph — це функція, яка бере на вхід DataFrame, виконує обробку тексту з використанням заданої моделі та параметрів, а потім повертає список концепцій (entities), отриманих з тексту;

3) graph2Df — це функція, яка приймає на вхід список вузлів, створює з нього DataFrame, видаляє порожні значення в ключових колонках ("node_1" і "node_2"), а потім перетворює текст у цих колонках на нижній регістр для забезпечення єдності даних. Ця функція може бути корисною для попередньої обробки графових даних перед їх подальшим аналізом або візуалізацією;
4) contextual_proximity — це функція, яка бере DataFrame з вузлами та ідентифікаторами текстових фрагментів, і створює новий DataFrame, що представляє контекстуальну близькість між вузлами, які з'являються в одному і тому ж фрагменті тексту;
5) colors2Comunity — це функція, яка створює DataFrame, який призначає кожному вузлу з спільнот унікальний колір і групу. Це може бути корисним для візуалізації спільнот, де кожна спільнота буде мати свій колір.

Маючи розуміння усіх функцій для роботи з датафреймом можна перейти до основних функцій та їх алгоритму. Почнемо з пошуку.

2.1 Алгоритм реалізації пошуку

Пошук було реалізовано для того, щоб в усьому наборі інформації, що буде доступний, була можливість знайти або конкретну інформацію, або набір тексту, який можна помістити у алгоритм створення графу, для того щоб створити систему кореляцій, по якій можна буде знайти взаємодію між елементами. Алгоритм пошуку не є складним (рис. 2.2), проте варто заглибитись в розуміння елементів за якими його було створено.

Рисунок 2.2 — Блок-схема роботи пошуку.

Модель кодування ми створюємо на основі моделі HuggingFace з бібліотеки langchain. Кодування створюється для імплементації бази даних FAISS (Facebook AI Similarity Search) з бібліотеки langchain. Для цього беремо кодування kwargs та зберігаємо у нижченаведеному вигляді розбитий текст.

[-0.038338545709848404, 0.1234646886587143, -0.02864295244216919]

Наступним кроком створюється пайплайн на основі моделі TinyLLama, який буде опрацьовувати запити та шукати схожі відповіді у базі даних, та генерувати відповіді.

2.2 Алгоритм першого підходу побудови графу

Побудова графу знань є ключовим завданням дипломної роботи. Для побудови використовуються почергово функції documents2DataFrame, df2Graph, graph2Df, contextual_proximity та colors2Comunity (рис 2.3). Після завантаження даних наступним кроком буде розбиття тексту та оформлення датафрейму завдяки вищезазначеним функціям. В результаті залишається датафрейм у якому прописано всі вершини для графу та зв’язки. З датафрейму створюються список вершин та список зв’язків. Окрім того ми зберігаємо датафрейм для графу, а також зберігаємо налаштування, за якими працює модель, це знадобиться для відтворення графу, та вирахування моделі, яка буде стабільно працювати, що знадобиться в майбутньому для перевірки, а також вибираємо чи зберігати контекстуальну близькість. Контекстуальна близькість покаує непрямий зв’язок між . Наступним кроком буде створення спільнот. Спільноти у даному випадки – це набори вершин та відповідних вершин, що пов’язують їх, кожній назначається відповідний

колір за допомогою функції colors2Comunity. Потім іде створення простору для графу з бібліотеки networkx. Введення даних було реалізовано у вигляді текстового поля, в яке вводиться необхідний текст.

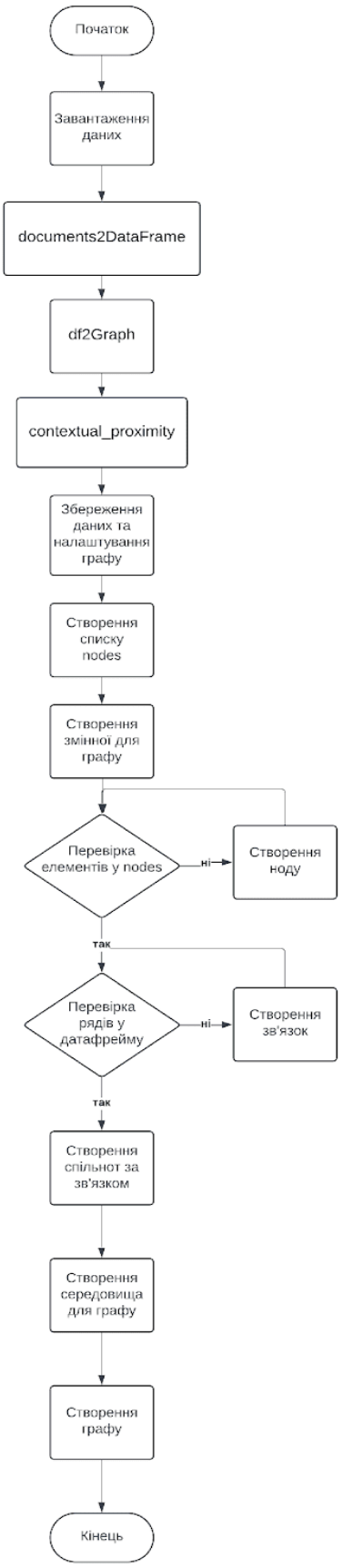


Рисунок 2.3 — Алгоритм побудови графу

2.3 Алгоритм другого підходу побудови графу

Другий алгоритм побудови графу було побудовано як підхід на основі обробки натуральної мови. Результатом має бути граф, який показує набір об’єктів та суб’єктів та їх зв’язок. Основний алгоритм складається з трьох алгоритмів – алгоритму побудови графу, алгоритму отримання зв’язків та алгоритму отримання сутностей.

Алгоритм отримання зв’язків було реалізовано у функції get_relation. Вона та функція get_entities необхідні для функції process_file_content. Загально потрібна імплементація механізму опрацювання природної мови, яка була задана у функції загальній для побудови графу, щоб зробити get_relation та get_entities.

Алгоритм отримання зв’язків було реалізовано у функції get_relation полягає у тому, щоб створити пошукову систему matcher, яка буде використовувати модель з попередньо заданими параметрами, а саме патернами зв’язку. У даному випадку, патерн представляє собою список, що містить один шаблонний список з чотирьох елементів:

1) токен із синтаксичною залежністю ROOT (головне дієслово речення);
2) токен із синтаксичною залежністю prep (прийменник) — не обов'язково ('OP': "?");
3) токен із синтаксичною залежністю agent (агент) — не обов'язково;
4) токен з частиною мови ADJ (прикметник) — не обов'язково.

Після цього відбувається пошук по документам на збіги. З усіх знайдених збігів вибирається останній. Потім витягується відповідний цьому збігу діапазон тексту та повернення рядку з відповідним збігом (рис.2.4).

Рисунок 2.4 — Алгоритм роботи get_relation

Алгоритму отримання сутностей починається з Ініціалізації порожніх рядків для збереження суб'єктів (ent1) і об'єктів (ent2), а також змінні для збереження попереднього токена та його залежності (prv_tok_dep і prv_tok_text), префікса (prefix) і модифікатора (modifier). Після створення змінних переходимо до циклу в якому відбуваються перевірки кожного токена, для розуміння якою частиною речення він є разом з витягом об’єкта та суб’єкта та його збереженням разом з його зв’язками (рис. 2.5)

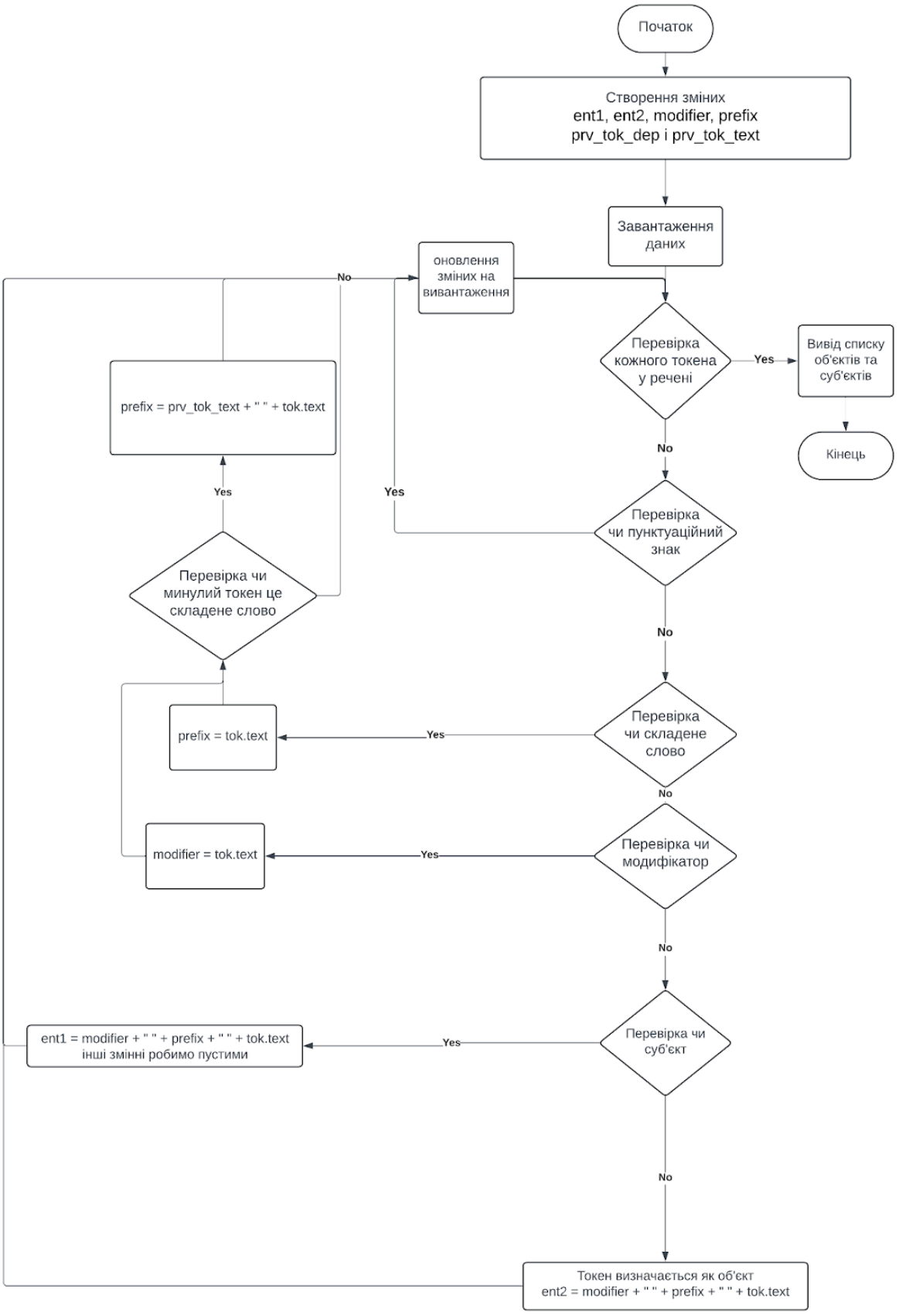


Рисунок 2.5 — Блок-схема алгоритму отримання сутностей

Функція process_file_content є по своїй суті функцією об’єднання, яка має перебрати списки, які були в результаті роботи функцій get_relation та get_entities. Результати переміщують у списки source, target та count, а потім з них утворюють датафрейм, який і повертається для побудови графу.

Функція get_nlp_graph будує граф як і звичайна функція побудови графу, але використовує датафрейм з функції process_file_content.

Загально алгоритм є швидкодіючим у порівнянні зі звичайним, але може мати помилки, через неправильне визначення концептів, а також зв’язки, які є доволі простими, і не мають завжди концептуальну цінність, бо типовим зв’язком є формат “Марічка => їсть=> стейк”. Такий тип зв’язків для аналізу простого текста є чудовим, але може бути проблематичним у ситуації з науковими статтями

2.4 Алгоритм графу речення

Алгоритм графу речення також використовує обробку за допомогою опрацювання природної мови, тільки замість графу концепцій розбиває кожне речення за синтаксичним аналізом, а також будує для кожного простий граф. Після розбиття тексту на окремі речення, іде перевірка кожного моделлю опрацювання природної мови, після чого будуються відповідні графи (рис 2.6).

Рисунок 2.6 — Блок-схема алгоритму графу речень

2.5 Алгоритм порівняння

Алгоритм порівняння потребує в своїй основі датафрейми, потрібні для побудови графу. Спочатку завантажуються датафрейми двох графів, з яких беруться об’єкти, суб’єкти та зв’язки, які потім переносяться у списки за допомогою функцій get_graph_names та get_graph_details. Оскільки створення тексту та графів за допомогою моделі не є точним, і єдиним способом оптимізації є використання налаштувань моделі, таким чином буде виведено правдиві результати, які в свою чергу можна позначити як еталонні, на їх основі і створюється граф, після цього береться граф побудований на конфігураціях, які було вибрано в залежності від побажань, наприклад дати

більшу свободу генерації, або навпаки, забрати можливість генерувати, і змусити працювати виключно за текстом. Визначення різниці між графами працює за принципом перевірки наявності вершин та зв’язків.

Для алгоритму обрахунку використовувались цикли для проходження по спискам граней графу, а потім вершин графу, які стосуються еталонного графу та графу, який оцінюють (рис. 2.7, рис 2.8).

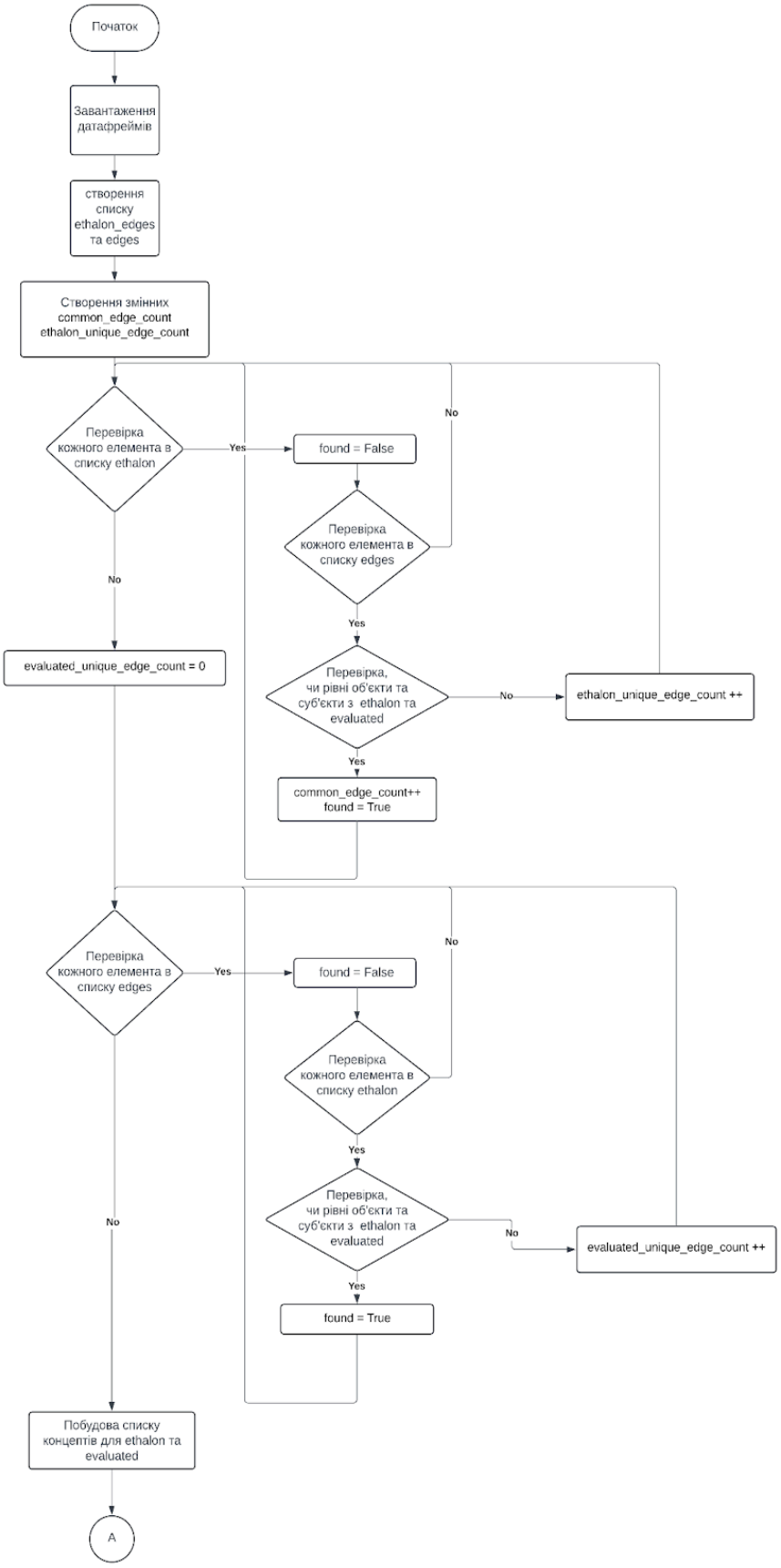


Рисунок 2.7 — Блок-схема порівняння граней

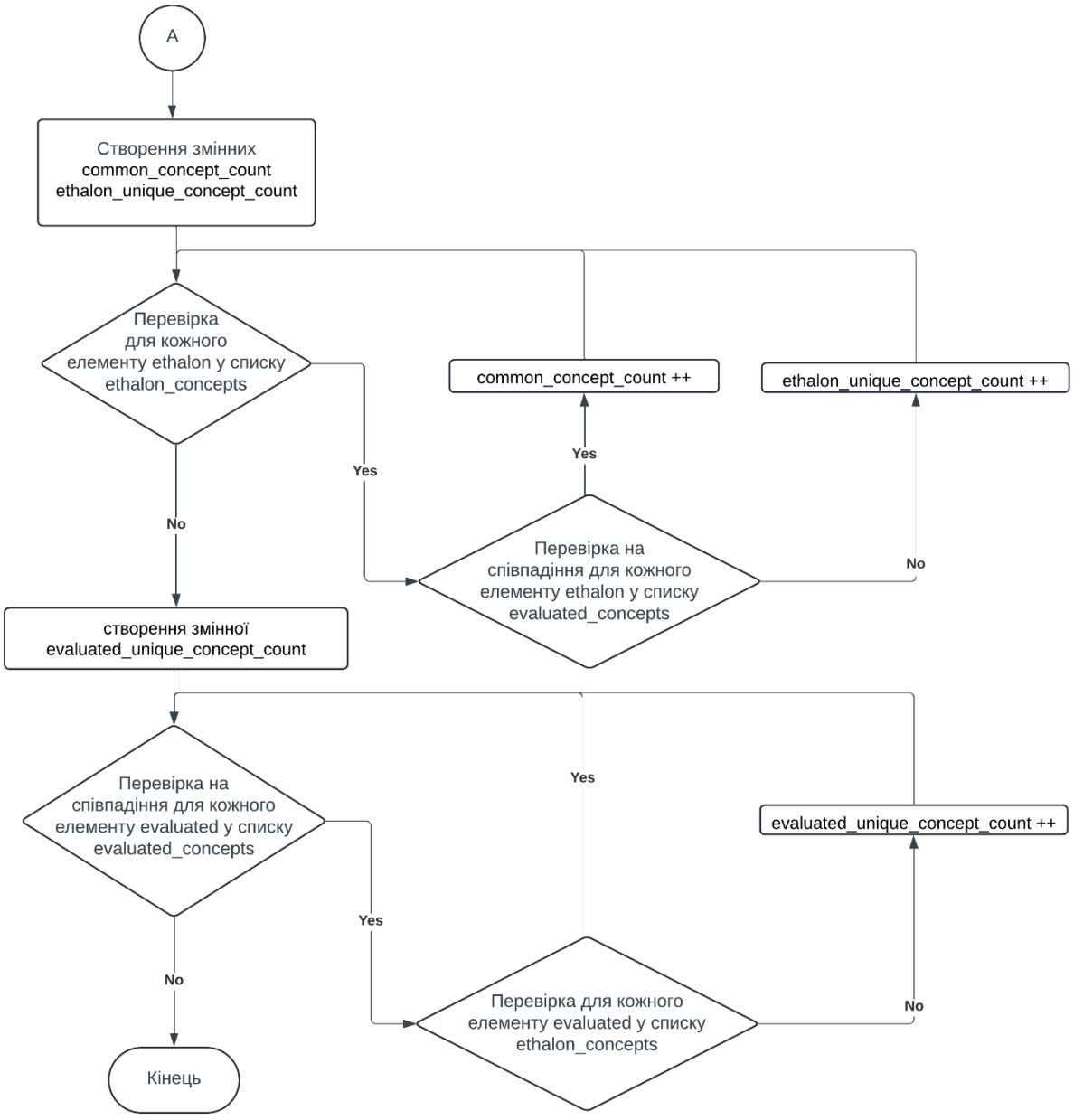


Рисунок 2.8 — Блок-схема порівняння концепцій

У результаті буде видно налаштування графу, текстове відображення датафрейму та статистику, прорoблену алгоритмом.

2.6 Висновки до розділу

У цьому розділі було продемонстровано основні алгоритми функцій необхідних для розв’язання задачі, а саме пошуку, двох методів побудови графу, графів речень, та оцінювання графів. Зокрема блок-схеми для кращого розуміння процесів, зокрема пояснено архітектуру за якою працює програма.

У результаті можливо розмежити ці алгоритми за трьома категоріями - основні, перевіркові та допоміжні. Алгоритми побудови графу є ключовими, так як саме граф є головною частиною, для якої створюються надбудови, так як основною задачею є реалізація автономності створення графів, як джерела знань. Для графа було реалізовано два підходи, зі своїми перевагами. Перший підхід пропонує більш комплексний граф, для якого можна використати модель Ollama та створити спосіб для налаштування графу, з складними зв’язками, де для кожного об’єкта та суб’єкта створюється зв’язок з поясненням їх взаємодії, або зв’язок концептуальної наближеності, який буде демонструвати зв’язки за сенсом, або логікою, які не належать до прямих зв’язків у тексті. Другий підхід в свою чергу демонструватиме простий граф, який покаже взаємозв’язки, які базуються на синтаксичному аналізі речення. До перевіркових належать алгоритм побудови графів речень та оцінки графів. Алгоритм оцінки графу потрібен для перевірки точності створених графів, та їх порівняння на основі ребер та вершин графу, за допомогою такого підходу можна визначити граф, який виступає у ролі еталону, за яким ми можемо перевірити будь-який створений моделлю граф з будь-якими налаштуваннями. Алгоритм графів речень є варіантом перевірки роботи моделі за допомогою синтаксичного аналізу і дозволяє на прикладі побачити роботу LLM для певної статті. Допоміжним є алгоритм пошуку, так як він створювався як варіант оптимізації процесу підготовки тексту, або отримання

відповідей. Ключовим елементом алгоритму пошуку є використання бази даних на основі векторного пошуку, що в свою чергу дає можливість зберігати інформацію зі статей для подальшого використання, і дозволить брати не всю статтю для побудови графу, а брати потрібний уривок з тексту, якщо потрібно.

РОЗДІЛ 3. РОЗРОБКА ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМНОГО ПІДХОДУ ПОБУДОВИ МЕРЕЖ ТЕРМІНІВ У ВИГЛЯДІ ПРОГРАМНОГО ПРОДУКТУ

Для апробації системного підходу в науковій діяльності були обрані предметні області, які відрізняються за наявністю спеціального сленгу та рівнем обсягу тексту, а також за концептуальною наповненістю. Зокрема, взяті на розгляд наступні категорії текстів.

1. Наукова стаття [12] про систему охорони здоров'я у Індії. У статті відсутній науковий спеціальний термінологічний апарат, а також стаття не є змістовною (приблизно 8 сторінок) і за побудовою сконцентрована на одній концепції.
2. Стаття [13], присвячена темі телефону, що включає в себе спеціальний термінологічний апарат, пов'язаний з телефоном та мобільним зв'язком. Стаття є найбільш змістовною з усіх (20 сторінок) і за побудовою не сконцентрована на одній концепції, а розглядає декілька концепцій в логічній послідовності.
3. Стаття [14], присвячена життю видр, і не наповнена спеціальною термінологією. Стаття є найменш змістовною з усіх (5 сторінок) і за побудовою сконцентрована на одній концепції, вбираючи в себе підтеми, що стосуються концепції.
4. Стаття [15], присвячена моделюванню мереж Петрі та включає в себе спеціальний термінологічний апарат. Стаття не є змістовною (6 сторінок) і за побудовою сконцентрована на одній концепції.
5. Стаття [16], описує кліматичні проблеми цього року, проте у ній відсутня спеціальна термінологія. Стаття не є змістовною (7

сторінок) і за побудовою не сконцентрована на одній концепції, а має структуру набору фактів, які є різними концепціями .

Ці категорії текстів були обрані для подальшого аналізу з метою перевірки графів. Тексти в статтях є різносторонніми, що дає можливість різносторонньої перевірки. Всі вищенаведені статті є текстовим блоком, який буде використано в роботі програми.

3.1 Архітектура роботи програми

Програму було реалізовано як веб-додаток за допомогою бібліотеки fastAPI з Python, а також JavaScript для фронт-енду. Загальна структура виглядає як побудова чотирьох відповідних запитів get_answer, get_graph, get_nlp_graph, get_nlp_struct, get_graph_evaluation робота яких іде відокремлено, але можна зробити у відповідній черзі (рис 3.1), виключенням з цього правила буде тільки функція get_graph_evaluation, якій потрібні графи.

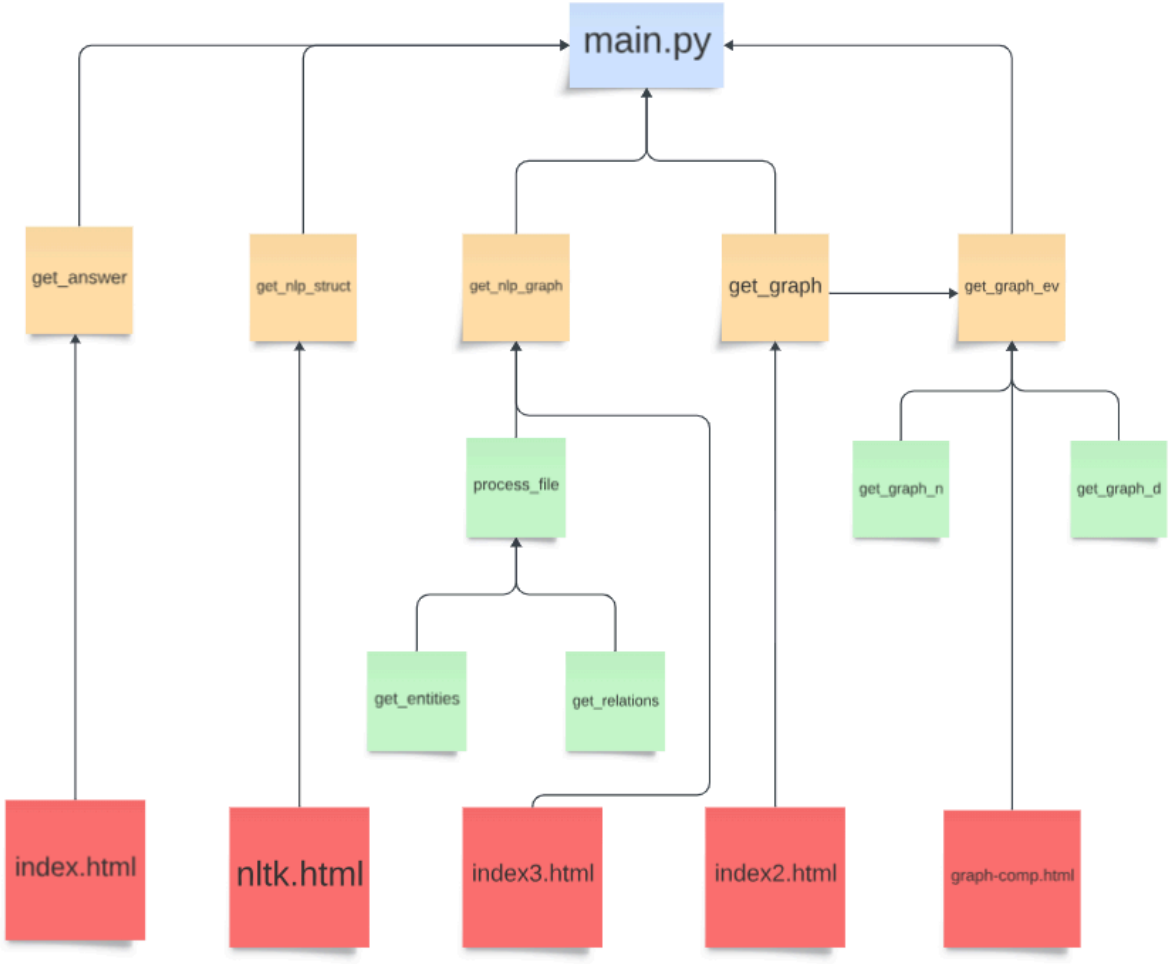


Рисунок 3.1 — Концептуальна карта структури програми.

Для кожного алгоритму було реалізовано відповідний html файл, у якому було прописано функціонал для кожної веб-сторінки, тому запуск ключових алгоритмів (позначені помаранчевим) буде відбуватись окремо, а також вони будуть виводити на окремі сторінки.

3.2 Огляд пошуку

Як і в другому розділі варто почати з пошуку. Його реалізація починається з створення окремого main файлу, в якому прописується алгоритм наведений у розділі 2 і в якому ми завантажуємо статті на опрацювання, які потім заносяться у базу даних FAISS. У API версії пошуку береться реалізація, де база вже встановлена. В файлі index.html було реалізовано інтерфейс для сторінки з полем введення, а також видачу відповідей (рис 3.2).

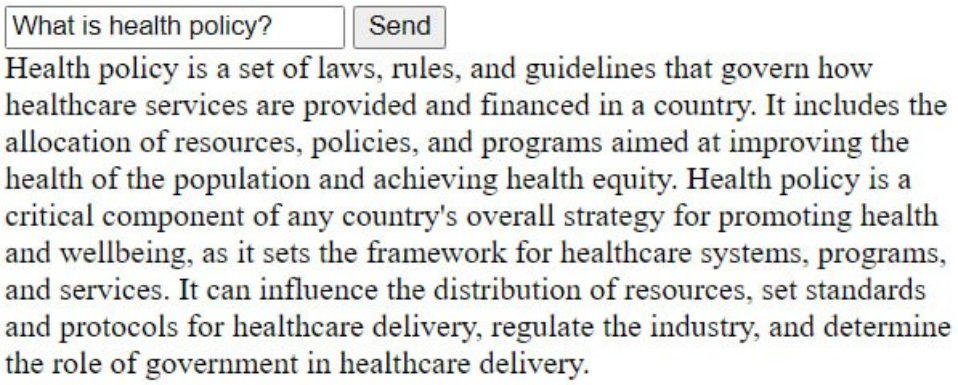


Рисунок 3.2 — Приклад роботи пошуку

Нюансом стала робота з базою даних, пошук є максимально деталізованим, через що векторний пошук може займати приблизно 20 хвилин в одній статті. Ця проблема є частою в деяких задачах, через фактор доступної при розробці техніки, при кращій техніці результати можна було б отримати швидше.

3.3 Огляд побудови графу

При відкритті сторінки створеної файлом index2.html ми бачимо сторінку з текстовим полем, та налаштуваннями, де спочатку треба обрати які опції змінити, і при натиснені на прапорець можна ввести данні, зокрема зберегти з назвою за вибором, або перевіряти на контекстуальну наближеність (рис. 3.3).

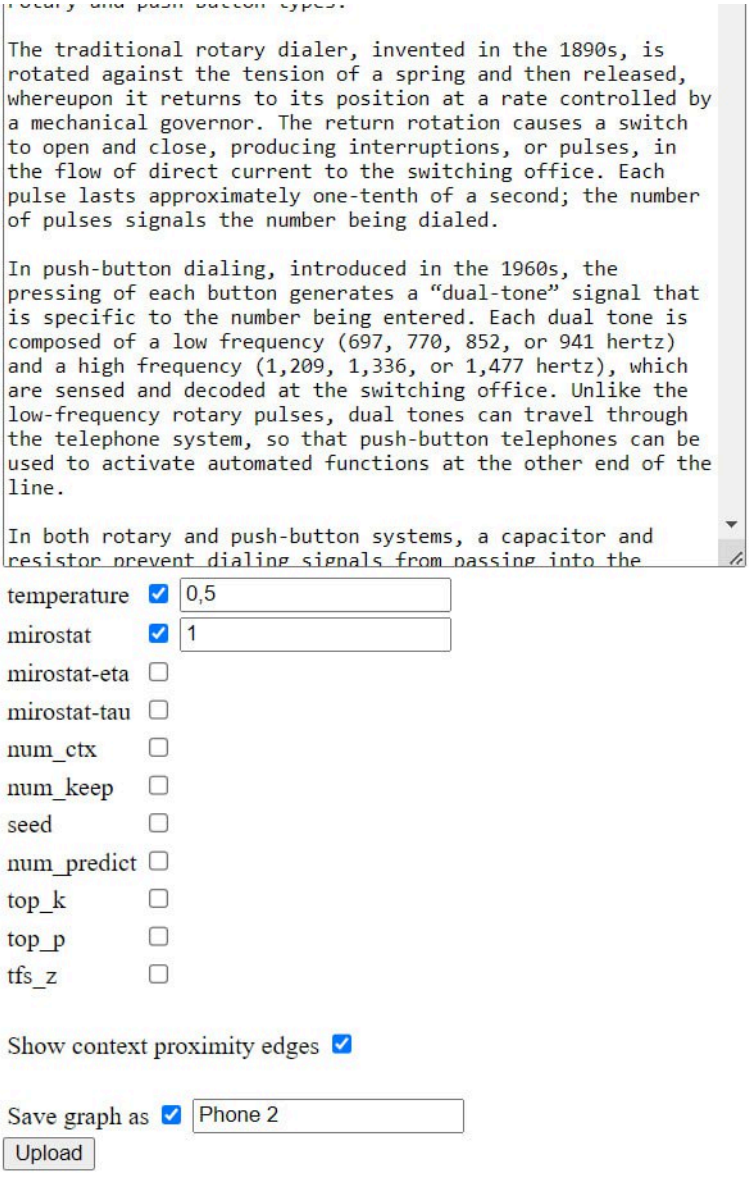


Рисунок 3.3 — Інтерфейс повернення графу

Під час побудови графу, при створені датафрейму для графу ми створюємо ноди та зв’язки, завдяки промту, які потім будуть виводитись у консоль (рис. 3.4) у вигляді набору, де маємо ноду-суб’єкт, ноду об’єкт та їх зв’язок.

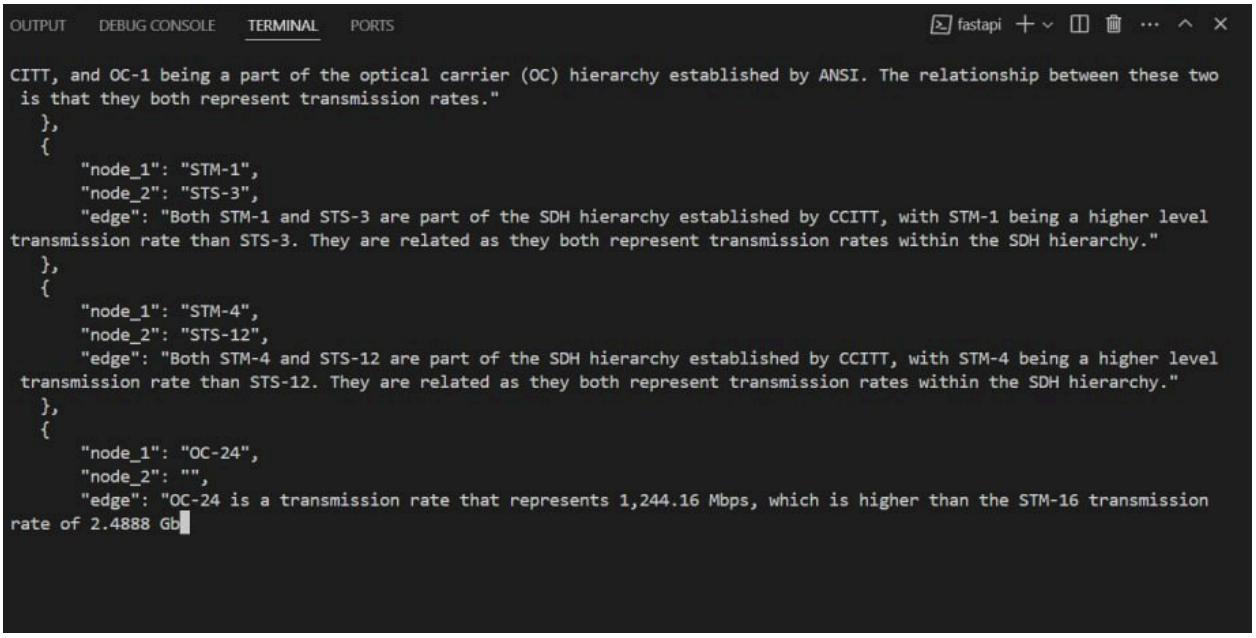


Рисунок 3.4 — Вивід у консоль під час роботи функції get_graph

Після опрацювання всіх наборів будується інтерактивний граф у якому, ми можемо рухати будь-які вершини, або рухатись по простору, в якому демонструється граф (рис 3.5, 3.6).

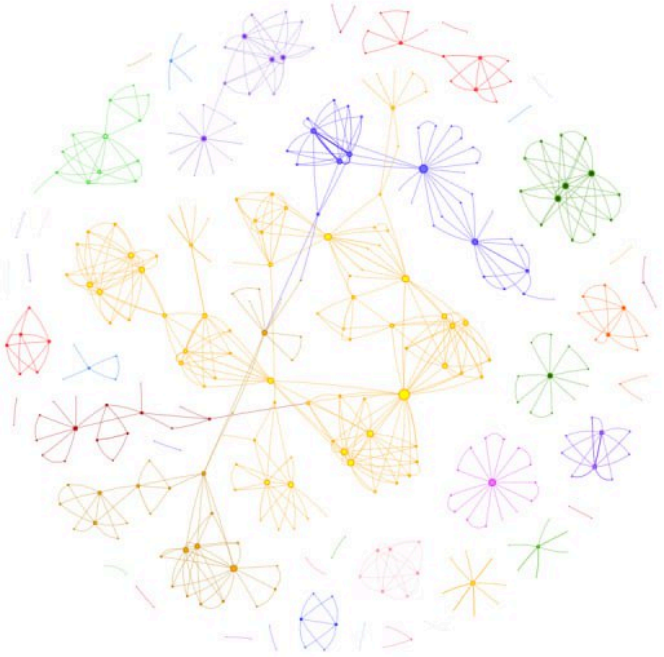


Рисунок 3.5 — Загальне зображення графу

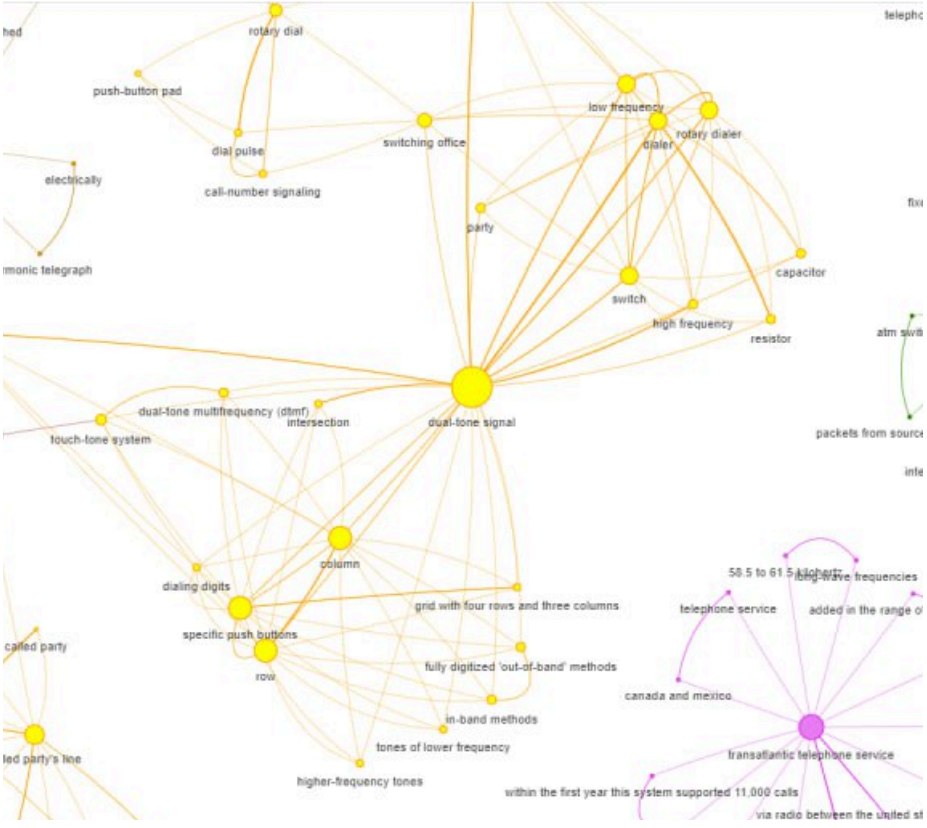


Рисунок 3.6 — Наближене зображення графу

Зокрема можна побачити інтерфейс налаштувань середовища графу (рис 3.7).

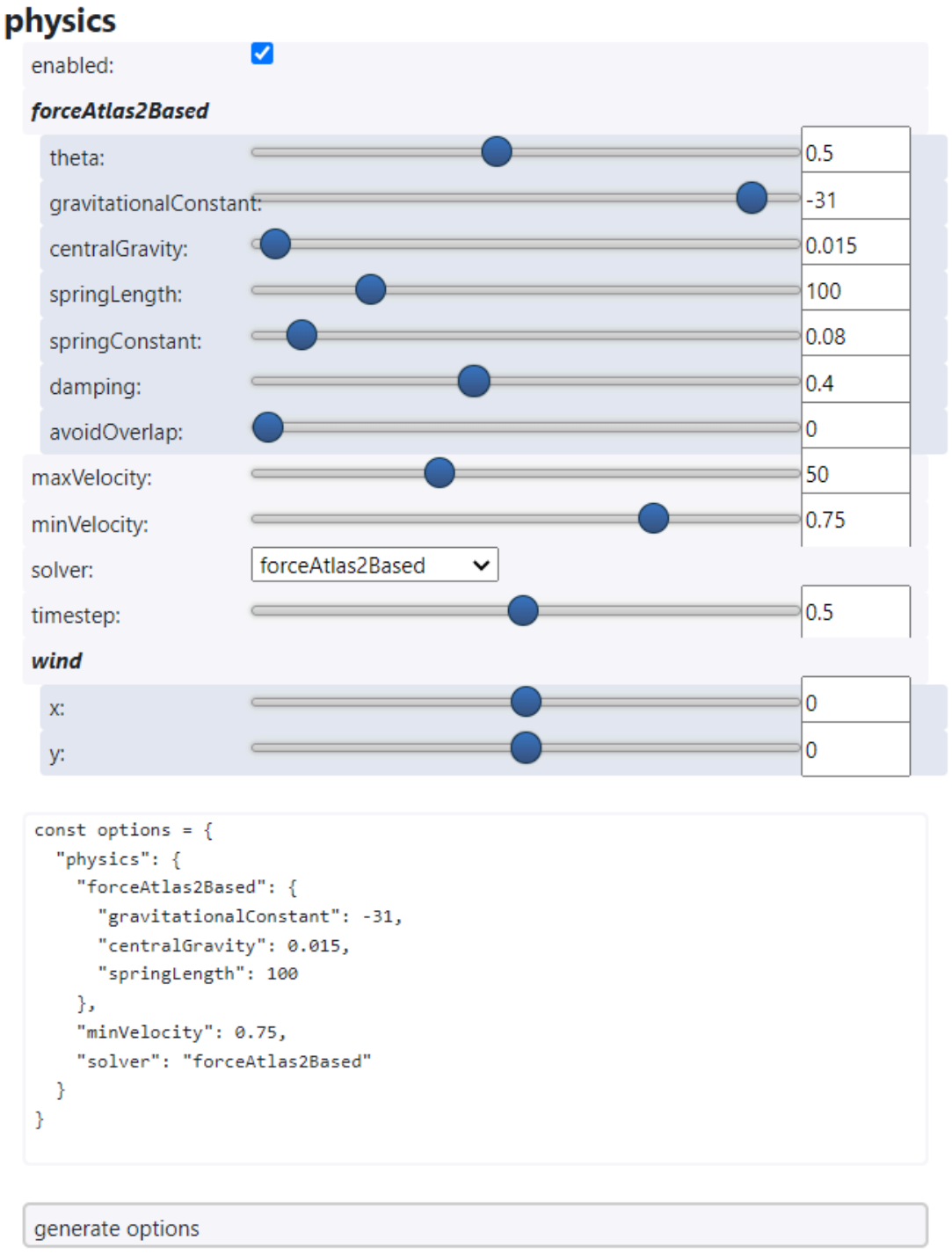


Рисунок 3.7 — Налаштування графу

3.4 Огляд другого підходу побудови графу

При вході на сторінку перше, що можна побачити це поле введення та кнопку завантаження (рис 3.8).

Рисунок 3.8 — Інтерфейс сторінки другого підходу побудови графу

Після введення тексту відбувається побудова графу (рис 3.9), де можна побачити таке ж середовище для графу та панель налаштування як і у підході get_graph.

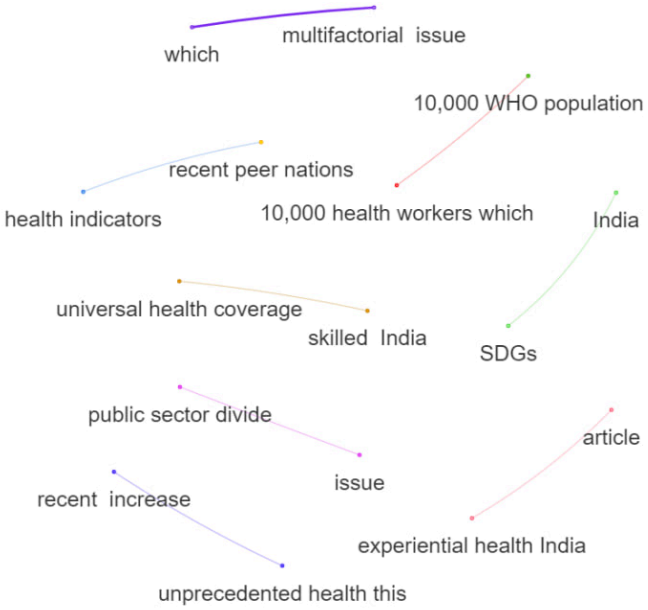


Рисунок 3.9 — Приклад короткого графу за get_nlp_graph

Функція get_nlp_graph потрібна для демонстрації альтернативи. Як основний було обрано метод зображений у get_graph, так як він показує розширений потенціал роботи моделі штучного інтелекту, але має недоліки, які перекриває get_nlp_graph, а саме: повільну роботу, а також складну реалізацію.

3.5 Огляд побудови графу речення

На початковій сторінці розміщено поле для тексту, яку у функції другого підходу побудови графу. Після опрацювання тексту буде побудовано статичні графи для кожного речення у форматі синтаксичного аналізу речення. Такі графи будуть представлені для кожного речення(рис. 3.10, 3.11).

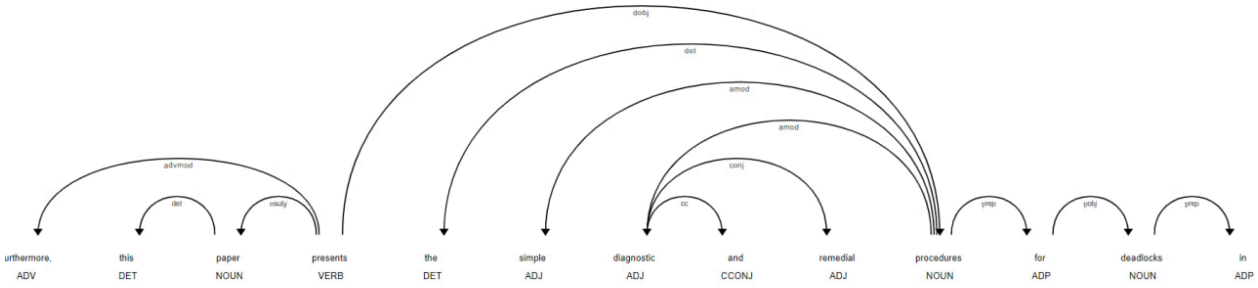


Рисунок 3.10 — Граф одного речення

Рисунок 3.11 — Структура розміщення графів речень

3.6 Огляд побудови статистики

Інтерфейс функції get_graph_evaluation, виглядає як два поля з вибором опцій (рис. 3.12), де можна обрати відповідні графи, якщо їх було збережено.

Рисунок 3.12 — Початковий інтерфейс функції

Після вибору на екрані буде видно налаштування графу, вершини та зв’язки між ними у текстовому форматі (рис.3.13), а також внизу можна буде побачити статистику по кожним унікальним, або спільним вершинам, або граням графу (рис.3.14).

Options:	
temperature	0.5
mirostat	1

Graph:		
application example	ppnrs	contextual proximity
automata	pns	contextual proximity
boer and murata	cordone et al.	mentioned in relation to the exponential growth of the number of siphons with respect to the PN size
boer and murata	liveness property	contextual proximity
chu and xie	li and wei	cited in the context of deadlock detection and analysis
chu and xie	liveness property	contextual proximity
circular waiting	concurrent jobs flow	contextual proximity
circular waiting	deadlock	contextual proximity
circular waiting	deadlock prevention policy	contextual proximity
circular waiting	fms	contextual proximity
circular waiting	job shop scheduling problem	contextual proximity
circular waiting	machine	In FMSs, deadlock occurs due to circular waiting for a specific machine by two jobs.,contextual proximity
circular waiting	mrts (modified reachability trees)	contextual proximity
circular waiting	pn controller	contextual proximity
circular waiting	pn model	contextual proximity
circular waiting	resource allocation system	contextual proximity
circular waiting	two jobs	In FMSs, circular waiting occurs between two jobs for a specific machine.,contextual proximity
complete siphon enumeration	liveness property	contextual proximity

Options:	
temperature	1
mirostat	2

Graph:		
application of proposed approach for deadlock detection and construction of recovery model for ppnrs.	ppnrs	contextual proximity
application of proposed approach for deadlock detection and construction of recovery model for ppnrs.	tvs	contextual proximity
bottom-up approach	combined model	contextual proximity
bottom-up approach	ezpeleta and martinez	contextual proximity
bottom-up approach	fmss	contextual proximity
bottom-up approach	high-level description	contextual proximity
bottom-up approach	huang	contextual proximity
bottom-up approach	huang et al.	contextual proximity
bottom-up approach	kamath and viswanadham	contextual proximity
bottom-up approach	large-scale combined model	contextual proximity
bottom-up approach	lee et al.	contextual proximity
bottom-up approach	net modules	contextual proximity
bottom-up approach	original high-level net	contextual proximity
bottom-up approach	pn model	contextual proximity
bottom-up approach	pn model of a fms	contextual proximity
bottom-up approach	pns	contextual proximity
bottom-up approach	souissi	contextual proximity
bottom-up approach	subnets	contextual proximity

Рисунок 3.13 — Завантажені графи та налаштування

Statistics:	
Common edge count	24
Ethalon unique edge count	233
evaluated unique edge count	272
Common concept count	24
Ethalon unique concept count	55
Evaluated unique concept count	65

Рисунок 3.14 — Статистика по двом графам

Маючи статистику можливо підрахувати різницю між різними графами та патерн, який прослідковується при використанні налаштувань середньої генерації та максимальної. Середні налаштування беруться за еталон, так як не обмежують LLM у мисленні, проте задають стійкість дій, що більш спонукає LLM до слідування тексту, наприклад вершини для графу будуть взяті прямо з тексту, а ребра будуть зачасту повторювати речення. При максимальній генерації LLM створює вершини на основі тексту, але якщо помірна генерація, яка є еталонною робить зачасту вершини за текстом, то

при максимальній кількість вершин може збільшитись, так як в такому випадку, LLM буде бачити більше “об’єктів”, які можна використати, що не є концептуально правильним. Параметрами для налаштування були температура, а також mirostat. Одним із ключових аспектів мовних моделей є передбачення тексту — можливість побудувати висновок на основі існуючої інформації. Mirostat слугує за своєрідний перемикач “звуку” для коригування генерації, який має відповідати за неочікуваність відповідей. Температура є основним параметром для LLM у визначенні рівня свободи генерації , чим вища температура, тим вільніша генерація. на основі цих знань було вирішено зробити еталон налаштувань, та налаштування максимального ступеня, які роблять генерацію несподіваною.

Для початку було вирішено провести експеримент для перевірки генерації на основі параметру температури(t), для перевірки того, як буде змінюватись генерація графу в межах однієї статті, для цього було взято статтю CureUs (стаття [12]).

Таблиця 3.1 — Таблиця статистики експерименту 1

	кількість вершин	кількість ребер
t=0,1	178	648
t=0,3	203	613
t=0,5	196	862
t=0,7	156	370
t=1	175	684

Цей експеримент дозволяє нам побачити рівень генерації, який надається різними температурами, та на основі цього зробити висновок, який

варіант є більш консистентним. Для кращої демонстрації було вирішено побудувати додаткову таблицю, яка продемонструє всі спільні вершини та ребра у форматі (x,y) відповідно.

Таблиця 3.2 — Таблиця порівняння статистики при всіх температурах

	t=0,1	t=0,3	t=0,5	t=0,7	t=1
t=0,1	(178, 648)	(119, 233)	(108, 239)	(91, 116)	(101, 136)
t=0,3	(119, 233)	(203, 613)	(114, 233)	(93, 96)	(85, 145)
t=0,5	(108, 239)	(114, 233)	(196, 862)	(117, 78)	(103, 220)
t=0,7	(91, 116)	(93, 96)	(117, 78)	(156, 370)	(66, 63)
t=1	(101, 136)	(85, 145)	(103, 220)	(66, 63)	(175, 684)

У результаті можемо побачити найчастіші співпадіння по вершинам при температурі 0.5, аналогічно по ребрам. Тому будемо вважати температуру 0.5 як саму стабільну для задачі параметру генерації, через загально найбільшу кількість генерації.

Ще одним аспектом, який потрібно перевірити є різниця між максимально можливою генерацією та помірною. Було вирішено провести експеримент на основі 5 статей, у нижченаведеній таблиці можна побачити остаточну статистику.

Таблиця 3.3 — Таблиця статистики експерименту 2

	Еталонний граф		Спільні значення		Перевірений граф	
	ребра	вершини	ребра	вершини	ребра	вершини
Biology[13]	84	30	1	7	111	41
CureUs[12]	464	115	73	67	507	99
Phone[14]	1227	328	41	44	294	59
Climate[15]	539	117	75	58	308	78
Petri[16]	233	55	24	24	272	65

Стаття Biology була найменшою за розмірами і по ній можна було побачити тотальну різницю між графами. У наступних більших статтях можна побачити кращі результати у спільному секторі, бо при більшому наборі інформації, з більшим контекстом, отримується більше зв’язків та вершин. на прикладі статті CureUs можемо побачити покращення у спільному спекторі на прикладі ребер, де з 1% для еталонного графу з статті Biology, іде перехід до 13% для статті CureUs.

Стаття Phone була найбільшою, та розповідала про обширну історію створення мобільного зв’язку у тому вигляді у якому ми зараз його бачимо, тобто не було опису одного явища, а було плавне перетікання з одного в інше, що як можна побачити по результатам заплутало модель максимальної генерації, та вона зробила менше вершин та ребер ніж модель помірної генерації, яка зробила в три рази більше ребер, а унікальних вершин майже в 6 разів більше. Це пояснюється фактом того, що модель максимальної

генерації, видає більше результатів у випадку роботи з описом одного явища, наприклад стаття Biology описує життя видр, а стаття CureUs ситуацію в медичному секторі Індії, обидві статті розповідають в загальному про одну концепцію, а стаття Phone розповідає послідовний набір, через що можна зробити висновок, що модель максимальної генерації, створить трохи більше вершин та ребер, якщо стаття буде загальним описом одного концепту, а не почерговою подачею концептів. Якщо ж дивитись на результати з математичної точки зору, то можна побачити кореляцію зазначеного вище, де для еталонного графу кількість ребер у процентному співвідношенні вища, ніж у статті Phone, через бінально більший обсяг інформації(3% проти 1%).

Стаття Climate є одним з експериментів, де був не типовий формат статті для моделі максимальної генерації — стаття, яка є набором фактів. У цій статті можемо побачити ситуацію схожу на попередню, проте з меншим співвідношенням кількості ребер та вершин, в еталонному графі їх в два рази більше, проте через те що факти, хоч і були окремими концептами, зводились до однієї теми, що призвело до загально непоганих результатів, у вигляді 12% подібності ребер у еталонного графа з оцінюваним.

Стаття Petri є схожою за структурою до Biology, проте є в два рази більшою. Її статистичні дані є доведенням факту важливості розміру тексту, через загальну статистику відносно еталонного графу у 9%.

3.7 Висновки до розділу

У цьому розділі було продемонстровано фінальну версію програми, показано приклади роботи та пояснено всі нюанси реалізації програмного продукту

Загалом основною проблемою у реалізації стала відсутність достатньо технічної апаратури, так як не було досягнено гарної швидкодії, та доводиться чекати, поки програма відпрацює. В процесі реалізації, від другого варіанту побудови графу було вирішено відмовитись, через відсутність надання достатньо інформативності. В результаті алгоритм порівняння було реалізовано на основі першого алгоритму побудови, через більшу інформативність. Для експерименту для температури було взято першу статтю з текстового блоку. Для другого експерименту було взято всі статті з текстового блоку. На основі експериментів проведених на основі статистики, яку надає алгоритм порівняння графу, було висунуто припущення, про найкращу температуру, яка є головним параметром генерації та про кореляцію тексту та контексту набору інформації, при побудові графу на різних налаштуваннях, а саме:

1) при більшій кількості тексту відсоток співпадінь буде більший;
2) при генерації, з параметрами максимальної свободи можуть бути проблеми, при широкому спектрі контексту у тексті.

Найкращою виявилась температура 0.5, через гарну побудову тексту для зв’язків, а також гарну відповідність до тексту, окрім того саме на цій температурі можна отримати більше всього вершин та ребер.

В другому експерименті було перевірено другий параметр mirostat, в якому можна побачити дві конфігурації налаштування — на максимальну генерацію та помірну. Параметром перевірки було відсоткове співпадіння

ребер та вершин між двома графами. Хоч загально не було високих відсотків співпадіння, проте цей результат є очікуваним, через рівень непередбачуваності при максимальній генерації, проте навіть мала різниця є показником.

Алгоритм пошуку було створено з ухилом допоміжного інструменту, як і алгоритм побудови графу речень, а також у цьому розділі було пояснено архітектуру програми під час роботи.

РОЗДІЛ 4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на побудові графу знань за допомогою штучного інтелекту. Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної реалізації, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) є технологією, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих годин, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

Приклад оптимізації баз знань для студентів демонструє вигоди побудови графу знань за допомогою штучного інтелекту. Граф знань дозволяє створювати персоналізовані навчальні траєкторії, що враховують поточний рівень знань студента та його індивідуальні потреби, підвищуючи ефективність навчання. Також забезпечується швидкий доступ до релевантної інформації, що дозволяє студентам швидко знаходити відповіді на свої

питання та поглиблювати знання у конкретних областях. Крім того,в майбутньому можливо, що LLM автоматично проаналізує прогалини у знаннях студентів та запропонує відповідні матеріали для їх заповнення. Використання графу знань робить навчання більш інтерактивним та цікавим, дозволяючи студентам взаємодіяти з навчальним матеріалом у нових формах.

Застосування функціонально-вартісного аналізу дозволяє не лише оцінити реальну вартість майбутнього програмного продукту, але й виявити можливі резерви для зниження витрат. Це забезпечить кращу економічну ефективність та сприятиме успішній реалізації проекту. В результаті проведеного аналізу можна прийняти обґрунтовані рішення щодо вибору технологій та підходів, що максимально відповідають поставленим цілям та вимогам проекту. Використання графу знань у освітньому процесі значно підвищує якість та ефективність навчання, що є ключовим фактором успіху для студентів.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

— функціонування на персональних комп’ютерах із стандартним набором компонентів;

— зручність та зрозумілість для користувача;

— швидкість обробки даних та доступ до інформації в реальному часі;

— можливість зручного масштабування та обслуговування;

— мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 — розробка можливого програмного продукту, яка дозволяє аналізувати різні набори даних, та побудувати графи знань, які потім можна проаналізувати. Беручи за основу цю функцію, можна виділити наступні:

F_1 — вибір мови програмування;

F_2 — вибір середовища розробки;

F_3 — вибір моделі штучного інтелекту.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Python;

б) R.

Функція F_2:

а) використання Microsoft VS Code;

б) використання PyCharm.

Функція F_3:

а) використання моделі Ollama;

б) використання моделі ChatGPT.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

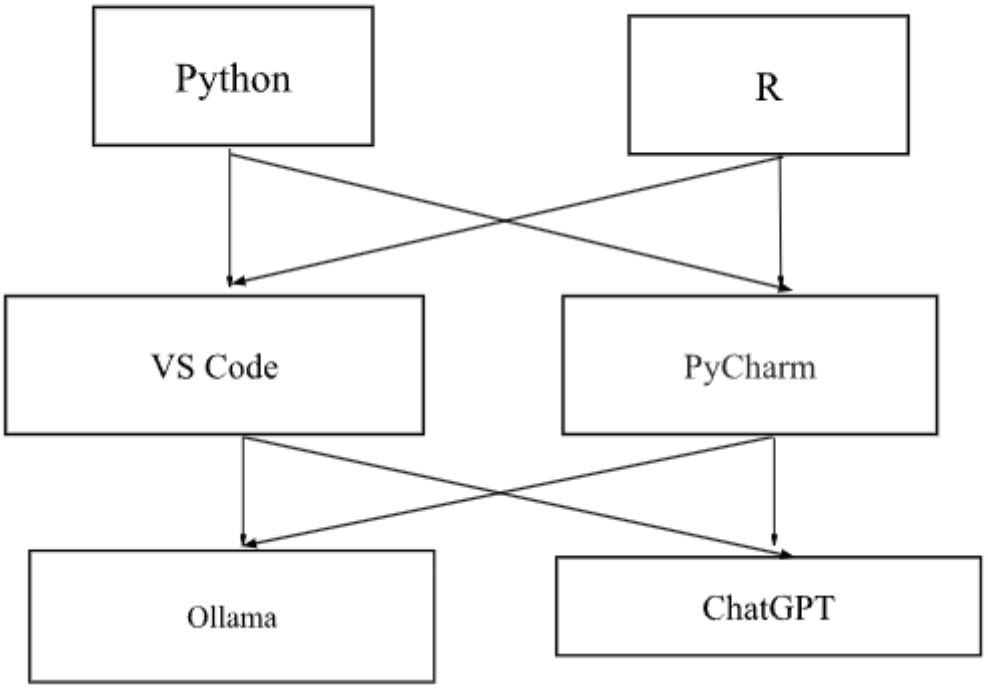


Рисунок 4.1 — Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 — Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Загальнодоступна швидка мова програмування з простим синтаксисом, доступність багатьох бібліотек для роботи зі LLM	Витрачає багато внутрішньох програми комп'ютера, потребує додаткових зусиль для виводу графічної інформації
	Б	Доступна в реалізації програма для різних обчислень	На написання коду необхідно мати базові навички та вміння
F_2	*А*	Безкоштовна платформа, яка вміщує в собі всі потрібні ресурси розробки	Не така швидкодійна як, інші платформи, зокрема не має в собі автоматичного встановлення бібліотек, та потребує додаткових зовнішніх процесів для встановлення пайтону
	Б	Швидкодійна платформа, яка має автоматичні налаштування пайтона	Платна платформа, яка займає більше простору, ніж інші редактори, що може уповільнити дію

Продовження таблиці 4.1

F_3	*А*	Безкоштовна модель, в якій можна змінювати налаштування, та яка встановлюється на комп'ютер для персонального використання	Трохи менша кількість функціоналу у порівнянні з іншими моделями
	Б	Платна модель, у якої є хороші опції інтерфейсу, а також є однією моделей з найкращим аналізом тексту	Не безкоштовна, зокрема потребує взаємодії зі сторонніми інструментами, для вивантаження персональної моделі

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1 — Перевагу даємо простоті та доступності бібліотек. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F_2 — Варіант А є доступнішим з фінансової точки зору, тому перевагу надаємо йому, а також мінуси для обох варіантів є збіжними.

Функція F_3 — Варіант А знову є доступнішим, а також витрачає менше часу на реалізацію. Перевага варіанту Б з більшим функціоналом не знадобиться.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

1) F_1а — F_2а — F_3а;

2) F_1а — F_2б — F_3а.

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

1) *X1* — швидкодія мови програмування;
2) *X2* — об’єм пам’яті для обчислень та збереження даних;
3) *X3* — час навчання даних;
4) *X4* — потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 — Основні параметри програмного продукту

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	50	70	100
Об’єм пам’яті	X2	Мб	1200	800	400
Час попередньої обробки даних	X3	хв	20	10	1
Потенційний об’єм програмного коду	X4	кількість рядків коду	2500	2000	1200

За даними таблиці 4.3 будуються графічні характеристики параметрів (рис. 4.2 — рис. 4.5).

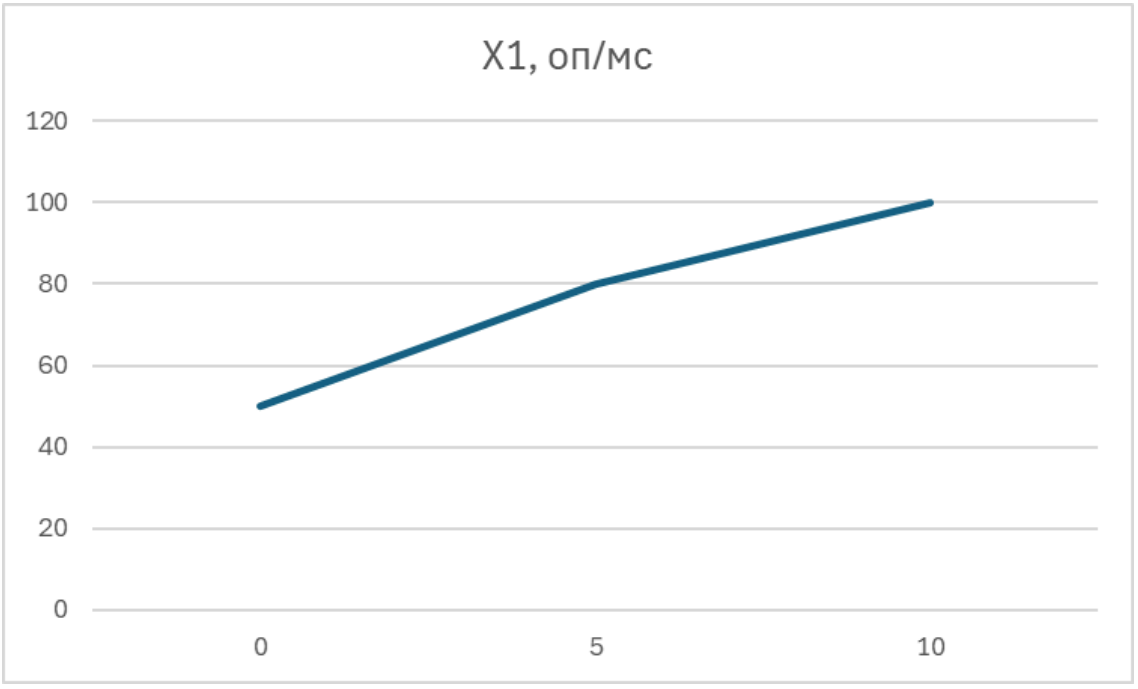


Рисунок 4.2 — X1, швидкодія мови програмування

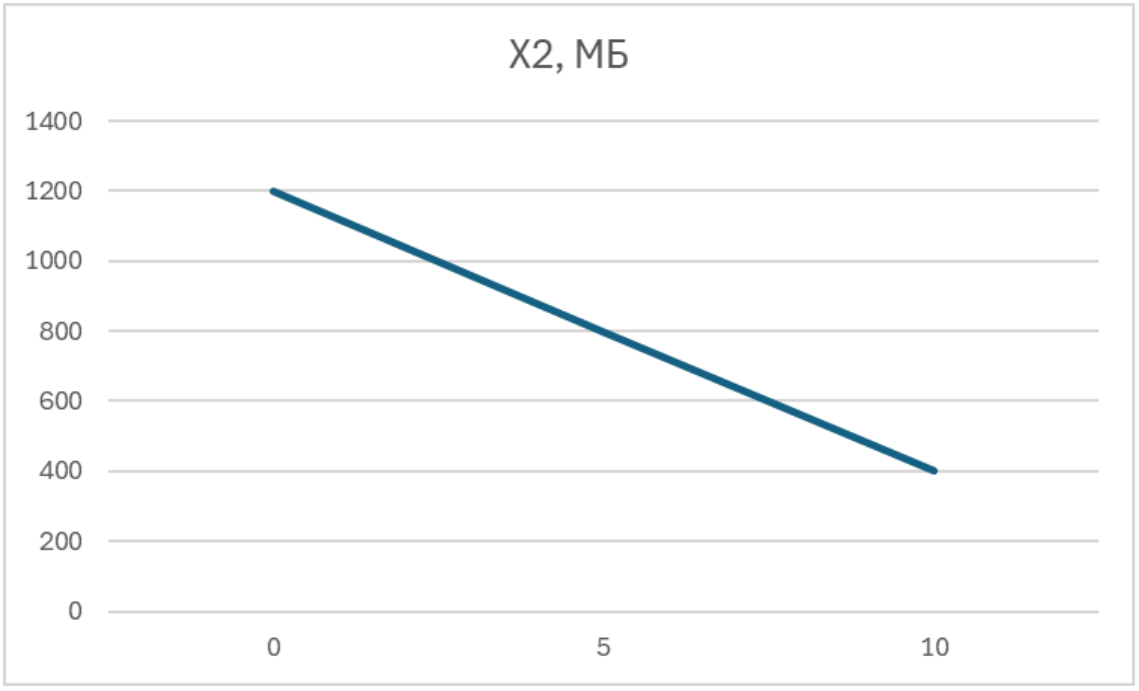


Рисунок 4.3 — X2, об’єм пам’яті

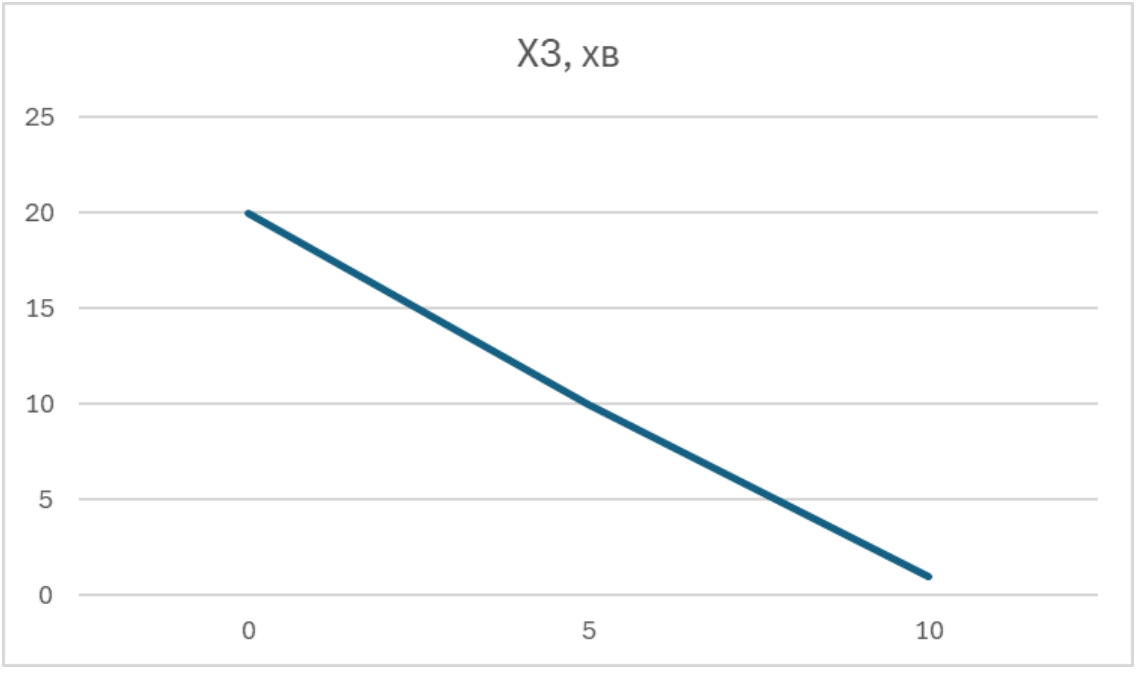


Рисунок 4.4 — X3, час попередньої обробки даних

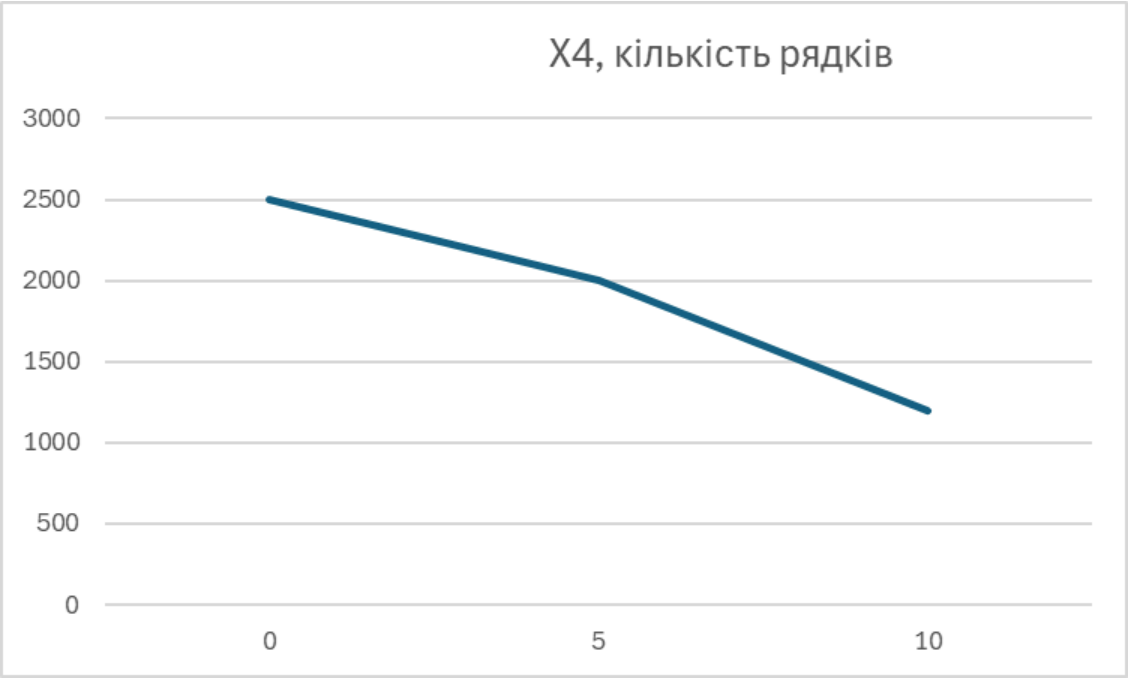


Рисунок 4.5 — X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значущості передбачає:

— визначення рівня значущості параметра шляхом присвоєння різних рангів;

— перевірку придатності експертних оцінок для подальшого використання;

— визначення оцінки попарного пріоритету параметрів;

— обробку результатів та визначення коефіцієнту значущості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 — Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	1	2	1	2	1	1	1	9	-8,5	72,25
X2	Об’єм пам’яті	Mb	2	1	2	1	2	2	2	12	-5,5	30,25
X3	Час попередньої обробки даних	мс	3	3	3	4	4	4	4	25	7,5	56,25
X4	Потенційний об’єм програмного коду	Кількість рядків коду	4	4	4	3	3	3	3	24	6,5	42,25
	Разом		10	10	10	10	10	10	10	70	0	201

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \qquad (4.1)$$

де N — число експертів,

n — кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5 \qquad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T \qquad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 201 \qquad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 201}{7^2(4^3 - 4)} = 0,82 > W_k = 0,67 \qquad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати заносимо у таблицю 4.4.

Таблиця 4.4 — Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	>	<	>	<	<	<	<	0,5
X1 і X3	<	<	<	<	<	<	<	<	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	<	<	<	>	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \{1.5 \text{ при } X_i > X_j \; 1.0 \text{ при } X_i = X_j \; 0.5 \text{ при } X_i < X_j \qquad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \qquad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \qquad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i^{'}}{\sum_{i=1}^{n} b_i^{'}} \qquad (4.9)$$

$$b_i^{'} = \sum_{i=1}^{N} a_{ij} b_j \qquad (4.10)$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 — Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	0,5	0,5	0,5	2,5	0,16	9,25	0,16	34,125	0,16
X2	1,5	1	0,5	0,5	3,5	0,22	12,25	0,21	77,875	0,2
X3	1,5	1,5	1	1,5	5,5	0,34	21,25	0,35	59,125	0,36
X4	1,5	1,5	0,5	1	4,5	0,28	16,25	0,28	41,875	0,28
Всього:					16	1	59	1	213	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об'єм пам'яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j}, \qquad (4.11)$$

де n — кількість параметрів;

$K_{\text{в}i}$ — коефіцієнт вагомості i–го параметра;

B_i — оцінка i–го параметра в балах.

Таблиця 4.6 — Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	A	X1	80	5	0,16	0,8

Продовження таблиці 4.6

F2	А	Х2	612	6	0,2	1,2
	Б	Х3	5	7	0,36	2,52
F4	А	Х4	1596	7	0,28	1,96

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}\left[F_{1k}\right] + K_{\text{ТУ}}\left[F_{2k}\right] + \ldots + K_{\text{ТУ}}\left[F_{zk}\right], \qquad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 0{,}8 + 1{,}2 + 1{,}96 = 3{,}96;$$
$$K_{K2} = 0{,}8 + 2{,}52 + 1{,}96 = 5{,}28.$$

Як видно з розрахунків, кращим є 2 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1) розробка проекту програмного продукту;

2) розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 — до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 — до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_О = T_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М}, \qquad (4.13)$$

де $T_Р$ — трудомісткість розробки ПП;

$К_П$ — поправочний коефіцієнт;

$К_{СК}$ — коефіцієнт на складність вхідної інформації;

$К_М$ — коефіцієнт рівня мови програмування;

$К_{СТ}$ — коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ — коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_Р$ = 37 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1,8. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0,9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 37 \cdot 1{,}8 \cdot 0{,}9 = 59{,}94 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р = 29$ людино-днів, $К_П = 0{,}9$, $К_{СК} = 1$, $К_{СТ} = 0{,}8$:

$$T_2 = 29 \cdot \ 0{,}9 \cdot 0{,}8 = 20.88 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (59{,}94 \ + 20{,}88 + 4{,}8 + 20{,}88) \cdot 8 = 852 \text{ людино-годин.}$$
$$T_{II} = (59{,}94 \ + 20{,}88 + 6{,}91 + 20{,}88) \cdot 8 = 868{,}88 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 40000 грн. Визначаємо середню зарплату за годину за формулою:

$$СЧ = \frac{М}{T_m \cdot t} \text{грн.,} \qquad (4.14)$$

де М – місячний оклад працівників;

T_m — кількість робочих днів тиждень;

t — кількість робочих годин в день.

$$СЧ = \frac{40000+40000}{2 \cdot 21 \cdot 8} = 238 \ \text{грн.} \qquad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$\text{ЗП} = \text{С}_{\text{ч}} \cdot \text{Т}_{i} \cdot \text{КД}, \qquad (4.16)$$

де $\text{С}_{\text{ч}}$ — величина погодинної оплати праці програміста;

Т_{i} — трудомісткість відповідного завдання;

$\text{К}_{\text{Д}}$ — норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $\text{С}_{\text{ЗП}} = 238 \cdot 852 \cdot 1.2 = 243331{,}2$ грн.

II. $\text{С}_{\text{ЗП}} = 238 \cdot 868.88 \cdot 1.2 = 248152{,}13$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $\text{С}_{\text{ВІД}} = \text{С}_{\text{ЗП}} \cdot 0.22 = 243331{,}2 \cdot 0.22 = 53532{,}86$ грн.

II. $\text{С}_{\text{ВІД}} = \text{С}_{\text{ЗП}} \cdot 0.22 = 248152{,}13 \cdot 0.22 = 54593{,}46$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($\text{С}_{\text{М}}$)

Так як одна ЕОМ обслуговує одного програміста з окладом 40000 грн, з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$\text{С}_{\text{Г}} = 12 \cdot \text{М} \cdot \text{К}_{\text{З}} = 12 \cdot 40000 \cdot 0{,}2 = 96000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$\text{С}_{\text{ЗП}} = \text{С}_{\text{Г}} \cdot (1 + \text{К}_{\text{З}}) = 96000 \cdot (1 + 0.2) = 115200 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 115200 \cdot 0,22 = \quad 25344 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.4 \cdot 0,25 \cdot 20000 = 7000 \text{ грн.},$$

де $К_{ТМ}$ — коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$ — річна норма амортизації;

$Ц_{ПР}$ — договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1,4 \cdot 20000 \cdot 0,08 = 2240 \text{ грн.},$$

де $К_Р$ — відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0,35 = \quad = \quad 627,2 \text{ години},$$

де $Д_К$ — календарна кількість днів у році;

$Д_В$, $Д_С$ — відповідно кількість вихідних та святкових днів;

$Д_Р$ — кількість днів планових ремонтів устаткування;

t — кількість робочих годин в день;

$К_В$ — коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 627,2 \cdot 0,2 \cdot 0,3 \cdot 5,23 = 196,8 \quad грн.,$$

де $N_С$ — середньо-споживча потужність приладу;

$К_З$ — коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ — тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0,67 = 20000 \cdot 0,67 = 13400 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \qquad (4.17)$$

$$С_{ЕКС} = 115200 + 25344 + 7000 + 2240 + 196,8 + 13400 = 163380,8 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 163380,8 / 627,2 = 260,49 \quad грн/год.$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot T \quad (4.18)$$

I. $С_М = 260,49 \cdot 852 = 221937,48$ грн.

II. $С_М = 260,49 \cdot 868,88 = 226334,55$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67 \quad (4.19)$$

I. $С_Н = 243331,2 \cdot 0,67 = 163031,9$ грн.

II. $С_Н = 248152,13 \cdot 0,67 = 166261,9$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н \quad (4.20)$$

I. $С_{ПП} = 243331,2 + 53532,86 + 221937,48 + 163031,9 = 681833,44$ грн.

II. $С_{ПП} = 248152,13 + 54593,46 + 226334,55 + 166261,9 = 695342$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{К_j} / С_{Ф_j} \quad (4.21)$$

$К_{ТЕР1} = 3,96 / 681833,44 = 0,581 \cdot 10^{-5}$

$К_{ТЕР2} = 5,28 / 695342 = 0,759 \cdot 10^{-5}$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}= 0,759 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розробляється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 0,759 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

— вибір мови програмного продукту — Python;

— реалізація за допомогою Microsoft VS Code;

— використання Ollama як моделі штучного інтелекту.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

4.8 Висновки до розділу

У даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

У результаті виконання функціонально-вартісного аналізу програмного комплексу що розробляється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У роботі було розглянуто необхідні інструменти, пояснено їхню природу та взаємодію, а також обрано напрямок розвитку реалізації графів знань, як відображення мережі поняття. Було вирішено, що пошук є оптимальним варіантом покращення, оскільки він дозволяє швидко знаходити всю необхідну інформацію в базі даних, що значно ефективніше, ніж ієрархічна структура. Пошук забезпечує швидкий доступ до даних без необхідності переглядати велику кількість файлів, тоді як ієрархія може ускладнити процес пошуку, особливо за складної структури даних.

Також були визначені параметри для перевірки графу на точність, що дозволяє оцінити дані після виконання коду. Було продемонстровано основні алгоритми, необхідні для розв’язання задачі: пошук, методи побудови графу, графи речень та оцінювання графів. Всі алгоритми розділено на три категорії: основні, перевіркові та допоміжні. Алгоритми побудови графу є ключовими, оскільки граф виступає головною частиною, для якої створюються надбудови. Реалізовано два підходи до побудови графу:

1) комплексний граф з моделлю Ollama для складних зв’язків;
2) простий граф на основі синтаксичного аналізу речення.

Перевіркові алгоритми включають побудову графів речень та оцінку графів для перевірки точності та порівняння. Допоміжним є алгоритм пошуку, який використовує векторний пошук для оптимізації процесу підготовки тексту та отримання відповідей.

У фінальній версії програми було продемонстровано приклади роботи та пояснено нюанси реалізації. Від другого варіанту побудови графу було відмовлено через недостатню інформативність, натомість алгоритм порівняння реалізовано на основі першого підходу до побудови графу. Було

проведено експерименти, які виявили найкращу температуру генерації — 0.5, яка забезпечує гарну відповідність тексту та максимальну кількість вершин і ребер. Також перевірено параметр mirostat з різними конфігураціями, що показало хоч і незначні, але хороші результати. На основі експериментів проведених на основі статистики, яку надає алгоритм порівняння графу, було висунуто припущення, про найкращу температуру, яка є головним параметром генерації та про кореляцію тексту та контексту набору інформації, при побудові графу на різних налаштуваннях, а саме:

1) при більшій кількості тексту відсоток співпадінь буде більший;
2) при генерації, з параметрами максимальної свободи можуть бути проблеми, при широкому спектрі контексту у тексті.

Можливим покращенням є впровадження синтезу керування, який автоматично підлаштовуватиме всі необхідні опції. Це дозволить зробити процес налаштування більш інтуїтивним і зручним для користувачів, зменшуючи потребу в ручній конфігурації.

Окремою проблемою стала відсутність прогресивної техніки. Без належного технічного обладнання виникають труднощі з часом виконання великих задач. Це може значно впливати на продуктивність системи, особливо при обробці великих обсягів даних. Вирішення цієї проблеми потребує інвестицій у сучасне обладнання, що дозволить значно прискорити процеси обробки та підвищити ефективність роботи системи в цілому.

Також ця робота є хорошим прикладом демонстрації роботи моделі штучного інтелекту, який встановлено на персональний комп’ютер, та з яким може працювати кожен, що в свою чергу каже, що майбутнє не за горами, та розвиток штучного інтелекту принесе користь усьому людству.

ПЕРЕЛІК ПОСИЛАНЬ

1. Lassila O. Swick R. R. Resource Description Framework(RDF) Model and Syntax Specification, 5 January 1999. URL: https://www.w3.org/TR/PR-rdf-syntax/Overview.html (date of the application: 06.06.2024).
2. Nayak R. The Research Agent: Addressing the Challenge of Answering Questions Based on a Large Text Corpus. 29 серп. 2023. URL: https://medium.com/towards-data-science/the-research-agent-4ef8e6f1b741
3. Nayak R. How to Convert Any Text Into a Graph of Concepts. 10 лист. 2023. URL: https://towardsdatascience.com/how-to-convert-any-text-into-a-graph-of-concepts-110844f22a1a (date of the application: 06.06.2024).
4. Prompting Techniques. URL: https://www.promptingguide.ai/techniques
5. Bratanic T. Enhancing the Accuracy of RAG Applications With Knowledge Graphs. Mar 30, 2024. URL: https://medium.com/neo4j/enhancing-the-accuracy-of-rag-applications-with-knowledge-graphs-ad5e2ffab663 (date of the application: 06.06.2024).
6. Takyar A. How to build a private LLM? URL: https://www.leewayhertz.com/build-private-llm/ (date of the application: 07.06.2024).
7. Vaid S. Information Retrieval: Power of Vector Search. URL: https://crucialbits.com/blog/information-retrieval-power-of-vector-search/ (date of the application: 06.06.2024).
8. Nayak P. Semantic Chunking for RAG. 21 квіт. 2024. URL: https://medium.com/the-ai-forum/semantic-chunking-for-rag-f4733025d5f5 (date of the application: 06.06.2024).

9. Upadhyay A. Implementing RAG with Langchain and Hugging Face. 16 жов. 2023. URL: https://medium.com/international-school-of-ai-data-science/implementing-rag-with-langchain-and-hugging-face-28e3ea66c5f7 (date of the application: 06.06.2024).
10. Bangqing P. English Semantic Feature Processing and Sentence Structure Analysis Based on Hierarchical Network of Concepts. *International Journal of Multimedia and Ubiquitous Engineering*. 2014. Vol.9, No.6. URL: https://gvpress.com/journals/IJMUE/vol9_no6/21.pdf (date of the application: 06.06.2024).
11. Hogan A. Blomqvist E. Cochez M. d'Amato C. de Melo G. Gutierrez C. Kirrane S. Labra Gayo J. E. Navigli R. Neumaier S. Ngonga Ngomo A.-C. Polleres A. Rashid S. M. Rula A. Schmelzeisen L. Sequeda J. Staab S. Zimmermann A. Knowledge Graphs [1st edition]. ACM Computing Surveys. Chapter 7. с. 87–99. URL: https://www.emse.fr/~zimmermann/KGBook/Multifile/quality-assessment/ (date of the application: 06.06.2024).
12. Saxena S. G. Godfrey T. India's Opportunity to Address Human Resource Challenges in Healthcare. Cureus. 11 черв. 2023. URL: https://www.cureus.com/articles/158868-indias-opportunity-to-address-human-resource-challenges-in-healthcare?source=post_page-----110844f22a1a----------------------------#!/ (date of the application: 06.06.2024).
13. Vinke C. M. Schoemaker N. J. The welfare of ferrets (Mustela putorius furo T): A review on the housing and management of pet ferrets. Elsevier. *Applied Animal Behaviour Science*. Volume 139, Issues 3–4, July 2012, Pages 155-168. URL:

https://www.sciencedirect.com/science/article/abs/pii/S0168159112000998 (date of the application: 06.06.2024).

14. Ahmad F. Huang H. Wang X-L. Petri net modeling and deadlock analysis of parallel manufacturing processes with shared-resources. Elsevier. *Journal of Systems and Software*. Volume 83, Issue 4, April 2010, Pages 675-688. URL: https://www.sciencedirect.com/science/article/abs/pii/S0164121209003057 (date of the application: 06.06.2024).
15. Borth D. Telephone - Dialing, Rotary, Push-Button. *Encyclopaedia Britannica online*. URL: https://www.britannica.com/technology/telephone/Transmission (date of the application: 06.06.2024).
16. Robinson D. 5 Biggest Environmental Problems of 2024. EARTH.ORG. *Climate Change*. 3 січ. 2024. URL: https://earth.org/the-biggest-environmental-problems-of-our-lifetime/ (date of the application: 06.06.2024).

ДОДАТОК А

Вибір середовища розробки є важливим фактором, так як визначає, який функціонал та яку мову можна обрати для програми. Було вирішено обрати Microsoft Visual Studio Code, через доступність для будь-кого, бо є середовищем розробки на безплатній основі, а також можливість використання Python в комбінації з JavaScript, що дозволяє створити веб-застосунок, та в майбутньому розвивати його, в цьому ж напрямку.

Переваги Microsoft Visual Studio Code
Ефективність використання ресурсів

Однією з найбільших переваг Microsoft Visual Studio Code (VS Code) є його ефективність у використанні системних ресурсів. Цей редактор коду спроектовано таким чином, щоб бути легким і швидким, забезпечуючи при цьому багатий функціонал для розробників. VS Code оптимізований для швидкого запуску і роботи навіть на системах з обмеженими ресурсами, що дозволяє розробникам зосередитися на написанні коду без затримок і повільного завантаження.

VS Code підтримує розширення і плагіни, які можна вмикати або вимикати за потребою, що дозволяє оптимізувати використання ресурсів залежно від поточних завдань розробника.

Кросплатформеність

Microsoft Visual Studio Code є кросплатформеним редактором коду, що означає, що він працює на різних операційних системах без жодних проблем. VS Code підтримує Windows, macOS, і Linux, забезпечуючи однаковий користувацький досвід на всіх цих платформах. Це дозволяє розробникам використовувати один і той самий інструмент незалежно від операційної системи, яку вони використовують.

Багатофункціональність і гнучкість

VS Code підтримує широкий спектр мов програмування і фреймворків, надаючи розробникам гнучкість у виборі технологій для своїх проєктів. Вбудована

підтримка для популярних мов програмування, таких як Python, JavaScript, TypeScript, C++, і багатьох інших, робить VS Code універсальним інструментом для розробки програмного забезпечення.

Редактор також підтримує об'єктно-орієнтоване програмування (ООП), процедурне програмування та інші парадигми, що дозволяє розробникам створювати добре структуровані та підтримувані програми.

Однією з ключових особливостей VS Code є його розширюваність. Завдяки великій кількості доступних розширень, розробники можуть налаштовувати редактор під свої потреби, додаючи підтримку для нових мов, інструментів розробки, тем оформлення та інших функцій.

Виклики використання Microsoft Visual Studio Code
Ускладнення процесу налаштування

Попри всі переваги, використання VS Code може вимагати значного часу на початкове налаштування. Для того щоб повністю використати потенціал редактора, розробникам часто доводиться встановлювати і налаштовувати численні розширення і плагіни. Це може бути складним завданням для новачків, які тільки починають працювати з цим інструментом.

Потреба в детальному тестуванні конфігурацій

Через широкий спектр доступних розширень і налаштувань, програми, створені у VS Code, можуть потребувати додаткового тестування для забезпечення стабільності та сумісності всіх використаних компонентів. Різноманіття можливих конфігурацій може призводити до неочікуваних проблем, які потрібно ретельно виявляти і виправляти, що додає додаткових витрат часу і зусиль на стадії налаштування і тестування редактора.

Переваги мови програмування Python
Ефективність використання ресурсів

Однією з найбільших переваг Python є його здатність до швидкої розробки та простоти у використанні. Хоча Python не надає такого ж рівня контролю над апаратним забезпеченням, як мови низького рівня, він забезпечує високу продуктивність розробки за рахунок своєї простої та читабельної синтаксичної структури. Це особливо важливо для розробників, які працюють над проєктами у сфері штучного інтелекту, де швидкість написання коду і можливість швидкої ітерації є критичними.

Python має автоматичне управління пам'яттю, що дозволяє розробникам зосередитися на розробці алгоритмів і логіки, не витрачаючи час на ручне керування ресурсами. Це є суттєвою перевагою у порівнянні з мовами низького рівня, де управління пам'яттю вимагає додаткових зусиль і знань.

Кросплатформеність

Python є кросплатформеною мовою програмування, що означає, що програми, написані на Python, можуть бути виконані на різних операційних системах без значних змін у коді. Це дозволяє розробникам створювати додатки, які можуть працювати на різних платформах, таких як Windows, macOS, Linux, і навіть мобільних пристроях.

Багатофункціональність і гнучкість

Python підтримує об'єктно-орієнтоване програмування (ООШІП), що дозволяє розробникам створювати складні, але добре структуровані програми. Крім того, Python підтримує інші парадигми програмування, такі як процедурне, функціональне та генерічне програмування, що забезпечує високу гнучкість при розробці програмного забезпечення.

Python також має багатий набір бібліотек і фреймворків, таких як TensorFlow, Keras, PyTorch та інші, які роблять його ідеальним вибором для розробки рішень у сфері штучного інтелекту та машинного навчання. Ці інструменти спрощують процес створення, тренування і розгортання моделей штучного інтелекту.

Виклики використання Python

Ускладнення процесу розробки

Попри всі переваги, використання Python має свої виклики. Python є інтерпретованою мовою, що може призводити до дещо нижчої швидкодії порівняно з компільованими мовами, такими як C++. Це може бути критичним для деяких додатків, що вимагають максимальної продуктивності.

Також, простота синтаксису Python може приховувати складність виконуваних операцій, що іноді ускладнює оптимізацію коду.

Вибрані бібліотеки для Python

На основі зазначених критеріїв було обрано кілька ключових бібліотек для застосування в розробці програм з LLM, створення RAG (retrieval-augmented generation) та побудови графів. Ці бібліотеки відповідають вимогам щодо швидкодії, популярності та оновлюваності, що робить їх оптимальним вибором для реалізації функціоналу застосунку.

Потреба в детальному тестуванні

Через динамічну природу мови, програми на Python потребують детального тестування для виявлення і виправлення помилок. Хоча автоматичне управління пам'яттю зменшує ризик витоків пам'яті, інші типи помилок, такі як типові помилки та логічні помилки, потребують ретельного тестування. Це додає додаткових витрат часу і зусиль на стадії розробки та тестування, особливо для складних систем штучного інтелекту.

Критерії вибору бібліотек

1. Швидкодія

- Вимога: Бібліотека повинна забезпечувати високу швидкодію.

-Обґрунтування: Висока швидкодія бібліотеки дозволяє розробникам створювати ефективні додатки, які швидко обробляють дані та виконують задачі. Це особливо важливо для додатків з LLM, де затримки можуть значно вплинути на продуктивність і користувацький досвід.

2. Популярність

-Вимога: Бібліотека має бути популярною.

-Обґрунтування: Популярність бібліотеки зазвичай свідчить про її якість, надійність та активну спільноту користувачів. Основним критерієм популярності обрано кількість зірок на GitHub, що є індикатором довіри та поширеності бібліотеки серед розробників.

3. Оновлюваність

-Вимога: Бібліотека має регулярно оновлюватися.

-Обґрунтування: Регулярні оновлення свідчать про активну підтримку бібліотеки, усунення вразливостей, додавання нових функцій та забезпечення сумісності з новими версіями інших інструментів. Бібліотеки, що не підтримуються, можуть стати непридатними для довгострокового використання.

Вибрані бібліотеки для Python

На основі зазначених критеріїв було обрано кілька ключових бібліотек для застосування в розробці програм з LLM, створення RAG (retrieval-augmented generation) та побудови графів. Ці бібліотеки відповідають вимогам щодо

швидкодії, популярності та оновлюваності, що робить їх оптимальним вибором для реалізації функціоналу застосунку.

1. Langchain==0.0.314

-Призначення: Бібліотека для роботи з ланцюжками обробки мовних моделей.

-Швидкодія: Забезпечує високу продуктивність при обробці мовних даних.

-Популярність: Активно використовується в спільноті розробників AI.

-Оновлюваність: Регулярно оновлюється, що забезпечує підтримку нових можливостей та виправлення помилок.

-Переваги: Легка інтеграція, гнучкість у налаштуванні та використанні.

2. Torch

-Призначення: Фреймворк для машинного навчання.

-Швидкодія: Висока продуктивність, підтримка GPU.

-Популярність: Дуже популярний серед розробників машинного навчання.

-Оновлюваність: Активно підтримується та регулярно оновлюється.

-Переваги: Висока продуктивність, підтримка глибокого навчання, велика спільнота користувачів.

3. Transformers

-Призначення: Бібліотека для роботи з трансформерними моделями.

-Швидкодія: Ефективна обробка великих обсягів даних.

-Популярність: Одна з найпопулярніших бібліотек для NLP.

-Оновлюваність: Регулярно оновлюється.

-Переваги: Підтримка багатьох моделей, зручність використання, активна спільнота.

4. Sentence-transformers

-Призначення: Бібліотека для побудови векторних подань речень.

-Швидкодія: Висока ефективність у перетворенні тексту в вектори.

-Популярність: Дуже популярна серед розробників NLP.

-Оновлюваність: Регулярно оновлюється.

-Переваги: Висока точність, легкість інтеграції, велика кількість попередньо навчених моделей.

5. Datasets

-Призначення: Бібліотека для роботи з наборами даних.

-Швидкодія: Швидке завантаження та обробка даних.

-Популярність: Широко використовується в проєктах з машинного навчання.

-Оновлюваність: Регулярно оновлюється.

-Переваги: Зручність у використанні, підтримка багатьох форматів даних, ефективність.

6. Faiss-cpu

-Призначення: Бібліотека для ефективного пошуку за схожістю.

-Швидкодія: Висока продуктивність у пошуку векторів.
-Популярність: Популярна серед розробників AI.
-Оновлюваність: Активно підтримується.
-Переваги: Висока продуктивність, ефективний пошук векторів.

7. Accelerate
-Призначення: Бібліотека для прискорення тренування моделей.
-Швидкодія: Підвищує швидкість тренування моделей.
-Популярність: Широко використовується в спільноті AI.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Легкість інтеграції, висока ефективність.

8. Bitsandbytes
-Призначення: Бібліотека для оптимізації використання пам'яті.
-Швидкодія: Оптимізація використання пам'яті для швидшої обробки.
-Популярність: Зростає серед розробників AI.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Оптимізація пам'яті, зручність у використанні.

9. FastAPI
-Призначення: Фреймворк для створення API.
-Швидкодія: Висока продуктивність завдяки асинхронній підтримці.
-Популярність: Дуже популярний серед веб-розробників.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Висока продуктивність, простота у використанні, асинхронна підтримка.

10. Pyvis==0.3.2
-Призначення: Бібліотека для візуалізації графів.
-Швидкодія: Швидка візуалізація графів.
-Популярність: Використовується в спільноті розробників для побудови графів.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Зручність візуалізації, інтерактивність.

11. Seaborn==0.13.0
-Призначення: Бібліотека для візуалізації даних.
-Швидкодія: Ефективна побудова графіків.
-Популярність: Дуже популярна серед аналітиків даних.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Висока якість графіків, легкість у використанні.

12. Thinc==8.2.3
-Призначення: Бібліотека для створення нейронних мереж.
-Швидкодія: Висока продуктивність у тренуванні моделей.
-Популярність: Використовується в проєктах з LLM.

-Оновлюваність: Регулярно оновлюється.
-Переваги: Зручність у використанні, висока продуктивність.

13. Spacy==3.7.4
-Призначення: Бібліотека для обробки природної мови (NLP).
-Швидкодія: Висока продуктивність при обробці тексту.
-Популярність: Дуже популярна в спільноті NLP.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Вис

ока продуктивність, підтримка багатьох мов, зручність у використанні.

14. NLTK==3.8.1
-Призначення: Бібліотека для обробки природної мови.
-Швидкодія: Добра швидкодія для аналізу тексту.
-Популярність: Широко використовується в академічних дослідженнях та проєктах NLP.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Великий набір інструментів для обробки тексту, підтримка багатьох мов.

15. FS==2.4.16
-Призначення: Бібліотека для роботи з файловими системами.
-Швидкодія: Висока ефективність роботи з файлами.
-Популярність: Використовується в різних проєктах для роботи з файлами.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Зручність використання, підтримка різних файлових систем.

16. Pandas
-Призначення: Бібліотека для аналізу даних.
-Швидкодія: Висока ефективність обробки великих наборів даних.
-Популярність: Дуже популярна серед аналітиків даних.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Зручність у використанні, широкий набір функцій для аналізу даних.

17. Pathlib
-Призначення: Бібліотека для роботи з шляхами до файлів.
-Швидкодія: Ефективна обробка шляхів до файлів.
-Популярність: Широко використовується в Python-проєктах.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Зручність використання, інтуїтивний інтерфейс.

18. Numpy
-Призначення: Бібліотека для наукових обчислень.
-Швидкодія: Висока продуктивність обчислень.

-Популярність: Дуже популярна серед науковців і інженерів.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Широкий набір математичних функцій, ефективність.

19. Uuid
-Призначення: Бібліотека для генерації унікальних ідентифікаторів.
-Швидкодія: Висока швидкість генерації UUID.
-Популярність: Широко використовується в Python-проєктах.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Надійність, простота використання.

20. Os
-Призначення: Бібліотека для взаємодії з операційною системою.
-Швидкодія: Висока продуктивність в операціях з ОС.
-Популярність: Широко використовується в Python-проєктах.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Зручність використання, широкий набір функцій.

21. Json
-Призначення: Бібліотека для роботи з JSON.
-Швидкодія: Висока ефективність при обробці JSON даних.
-Популярність: Дуже популярна серед розробників.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Простота використання, гнучкість, висока продуктивність.

22. Networkx
-Призначення: Бібліотека для роботи з графами.
-Швидкодія: Висока продуктивність в обробці графів.
-Популярність: Широко використовується в наукових дослідженнях та проєктах.
-Оновлюваність: Регулярно оновлюється.
-Переваги: Зручність використання, багатий функціонал для роботи з графами.

Огляд моделі штучного інтелекту

Модель Ollama - це одна з останніх інновацій у світі штучного інтелекту, яка використовує глибоке навчання та нейронні мережі для досягнення вражаючих результатів у різних областях. Ось деякі з переваг цієї моделі:

1. Висока точність: Однією з ключових переваг моделі Ollama є її висока точність. Вона може розпізнавати та аналізувати складні зв'язки в даних з вражаючою точністю, що робить її корисною для рішення різноманітних завдань.

2. Гнучкість: Модель Ollama може бути адаптована до різноманітних завдань інтелектуального аналізу даних, від розпізнавання образів до мовного

розуміння. Це дає змогу використовувати її в різних галузях, таких як медицина, фінанси, технології та інші.

3. Швидкість обробки: Модель Ollama забезпечує швидку обробку великих обсягів даних, що дозволяє швидше приймати рішення на основі отриманих результатів. Це особливо важливо в областях, де швидкість реакції має ключове значення.

4. Автоматизація : Використання моделі Ollama дозволяє автоматизувати багато процесів, що раніше вимагали значної кількості людських ресурсів. Це дозволяє підвищити ефективність та знизити витрати в бізнесі.

5. Постійне вдосконалення: Оскільки модель Ollama базується на штучних нейронних мережах, вона постійно вдосконалюється через навчання на нових даних. Це дозволяє підтримувати її актуальність та адаптованість до змін у навколишньому середовищі.

Загалом, модель Ollama представляє собою потужний інструмент для аналізу даних, який може допомогти підвищити продуктивність, знизити витрати і забезпечити конкурентні переваги в різних сферах діяльності.

ДОДАТОК Б

my_API/main.py:

```
import torch
import networkx as nx
import pandas as pd
import numpy as np
import uuid
import spacy
import nltk
import os
import json


from fastapi import FastAPI
from pydantic import BaseModel
from fastapi.staticfiles import StaticFiles
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.vectorstores import FAISS
from transformers import  pipeline
from pyvis.network import Network
from spacy import displacy
from spacy.matcher import Matcher
from fs.opener import manage_fs

from util.df_helpers import documents2Dataframe, df2Graph, graph2Df, contextual_proximity, colors2Community

modelPath = "sentence-transformers/all-MiniLM-l6-v2"
model_kwargs = {'device':'cpu'}
encode_kwargs = {'normalize_embeddings': False}
embeddings = HuggingFaceEmbeddings(
     model_name=modelPath,      # Provide the pre-trained model's path
      model_kwargs=model_kwargs, # Pass the model configuration options
     encode_kwargs=encode_kwargs # Pass the encoding options
)
db = FAISS.load_local("faiss_index", embeddings)

retriever = db.as_retriever(search_kwargs={"k": 4})

pipe = pipeline("text-generation",
                    model="TinyLlama/TinyLlama-1.1B-Chat-v1.0",
                    torch_dtype=torch.bfloat16,
                    device_map="auto")

app = FastAPI()
```

```
app.mount("/static", StaticFiles(directory="static_data"), name="static")

class QuestionRequest(BaseModel):
    question: str | None = None

class QuestionResponse(BaseModel):
    answer: str | None = None



@app.post("/get/answer", response_model=QuestionResponse)
async def get_answer(question_request: QuestionRequest):
    answer = ''
    try:
                                                        docs = retriever.get_relevant_documents(question_request.question)


            context_text = '\n\n'.join([doc.page_content for doc in docs])
        messages = [
            {
                "role": "system",
                     "content": f'''Use the following pieces of context to answer the question at the end. If you don't know the answer, just say that you don't know, don't try to make up an answer.
                context: {context_text}'''
            },
                                    {"role": "user", "content": f'{question_request.question}'}
        ]
            prompt = pipe.tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
        print("Im here")
                    outputs = pipe(prompt, max_new_tokens=256, do_sample=True, temperature=0.7, top_k=50, top_p=0.95)
        print("Im hehere")
        answer = outputs[0]["generated_text"][len(prompt):]
    except:
        answer = 'Internal Server Error'
    ret = QuestionResponse()
    ret.answer = answer
    return ret

#####################################################################################
#Graph Functionality
#####################################################################################

class GraphRequest(BaseModel):
    document: str | None = None
    options: dict | None = None
    graphName: str | None = None
```

```
        showContextProximity: bool | None = None

    class GraphResponse(BaseModel):
        answer: str | None = None

    splitter = RecursiveCharacterTextSplitter(
        chunk_size=1500,
        chunk_overlap=150,
        length_function=len,
        is_separator_regex=False,
    )

    @app.post("/get/graph", response_model=GraphResponse)
    async def get_graph(graph_request: GraphRequest):
        print(graph_request)
        pages = splitter.split_text(graph_request.document)
        print("pages count " + str(len(pages)))
        df = documents2Dataframe(pages)
        print(df.head(10))
         concepts_list = df2Graph(df, model='zephyr:latest', options =
graph_request.options)
        dfg1 = graph2Df(concepts_list)
        print(dfg1.head(10))
        dfg1.replace("", np.nan, inplace=True)
                 dfg1.dropna(subset=["node_1",   "node_2",   'edge'],
inplace=True)
        dfg1['count'] = 4

        if graph_request.showContextProximity is True:
            dfg2 = contextual_proximity(dfg1)
            dfg2.tail()
            dfg = pd.concat([dfg1, dfg2], axis=0)
        else:
            dfg = dfg1
        dfg = (
            dfg.groupby(["node_1", "node_2"])
               .agg({"chunk_id": ",".join, "edge": ','.join, 'count':
'sum'})
            .reset_index()
        )
                        if    graph_request.graphName!=    None    and
graph_request.graphName!='':
                                            graph_output_directory    =
f'graphs/{graph_request.graphName}'
            if not os.path.exists(graph_output_directory):
                os.makedirs(graph_output_directory)
                    dfg.to_csv(f'{graph_output_directory}/graph.csv',
sep='\t', encoding='utf-8', header='true')
            with open(f'{graph_output_directory}/model_options.json',
'w', encoding='utf-8') as f:
                                    json.dump(graph_request.options,  f,
ensure_ascii=False, indent=4)

                nodes   =   pd.concat([dfg['node_1'],   dfg['node_2']],
axis=0).unique()
```

```
            G = nx.Graph()

            # Add nodes to the graph
            for node in nodes:
                G.add_node(
                    str(node)
                )

            # Add edges to the graph
            for index, row in dfg.iterrows():
                G.add_edge(
                    str(row["node_1"]),
                    str(row["node_2"]),
                    title=row["edge"],
                    weight=row['count'] / 4
                )

            communities_generator = nx.community.girvan_newman(G)
            top_level_communities = next(communities_generator)
            next_level_communities = next(communities_generator)
            communities = sorted(map(sorted, next_level_communities))
            print("Number of Communities = ", len(communities))
            print(communities)

              # Now add these colors to communities and make another
dataframe
            colors = colors2Community(communities)

            for index, row in colors.iterrows():
                G.nodes[row['node']]['group'] = row['group']
                G.nodes[row['node']]['color'] = row['color']
                G.nodes[row['node']]['size'] = G.degree[row['node']]

            net = Network(
                notebook=False,
                # bgcolor="#1a1a1a",
                cdn_resources="remote",
                height="900px",
                width="100%",
                select_menu=True,
                # font_color="#cccccc",
                filter_menu=False,
            )

            net.from_nx(G)
            # net.repulsion(node_distance=150, spring_length=400)
            net.force_atlas_2based(central_gravity=0.015, gravity=-31)
                 #  net.barnes_hut(gravity=-18100,  central_gravity=5.05,
spring_length=380)
            net.show_buttons(filter_=["physics"])

            file_name = uuid.uuid4().hex
            tmp_file = f"static_data/tmp/{file_name}.html"
```

```
        net.show(tmp_file, notebook=False)

        ret = GraphResponse()
        ret.answer = f"/static/tmp/{file_name}.html"
        return ret

    ######################################################################
##################################
    #NLP Functionality
    ######################################################################
##################################

    nlp = spacy.load("en_core_web_sm")

    @app.post("/get/nlp/graph", response_model=GraphResponse)
    async def get_nlp_graph(graph_request: GraphRequest):

        dfg = process_file_content(graph_request.document)

            nodes = pd.concat([dfg['node_1'], dfg['node_2']],
axis=0).unique()

        G = nx.Graph()

        # Add nodes to the graph
        for node in nodes:
            G.add_node(
                str(node)
            )

        # Add edges to the graph
        for index, row in dfg.iterrows():
            G.add_edge(
                str(row["node_1"]),
                str(row["node_2"]),
                title=row["edge"],
                weight=row['count'] / 4
            )

        communities_generator = nx.community.girvan_newman(G)
        top_level_communities = next(communities_generator)
        next_level_communities = next(communities_generator)
        communities = sorted(map(sorted, next_level_communities))
        print("Number of Communities = ", len(communities))
        print(communities)

          # Now add these colors to communities and make another
dataframe
        colors = colors2Community(communities)

        for index, row in colors.iterrows():
            G.nodes[row['node']]['group'] = row['group']
            G.nodes[row['node']]['color'] = row['color']
            G.nodes[row['node']]['size'] = G.degree[row['node']]
```

```
        net = Network(
            notebook=False,
            # bgcolor="#1a1a1a",
            cdn_resources="remote",
            height="900px",
            width="100%",
            select_menu=True,
            # font_color="#cccccc",
            filter_menu=False,
        )

        net.from_nx(G)
        # net.repulsion(node_distance=150, spring_length=400)
        net.force_atlas_2based(central_gravity=0.015, gravity=-31)
             #  net.barnes_hut(gravity=-18100,  central_gravity=5.05,
spring_length=380)
        net.show_buttons(filter_=["physics"])

        file_name = uuid.uuid4().hex
        tmp_file = f"static_data/tmp/{file_name}.html"
        net.show(tmp_file, notebook=False)

        ret = GraphResponse()
        ret.answer = f"/static/tmp/{file_name}.html"
        return ret

    def process_file_content(content):
                                                sent_detector          =
nltk.data.load('tokenizers/punkt/english.pickle')
        sentences = sent_detector.tokenize(content.strip())

        entity_pairs = [get_entities(s) for s in sentences]
        print(entity_pairs)

        relations = [get_relation(s) for s in sentences]
        print(relations)

        source = [i[0] for i in entity_pairs]
        target = [i[1] for i in entity_pairs]
        count = [0 for i in entity_pairs]

          kg_df  =  pd.DataFrame({'node_1':  source,  'node_2':  target,
'edge': relations, 'count': count})
        print(kg_df)
        return kg_df


    def get_entities(sent):
        ent1 = ""
        ent2 = ""

        prv_tok_dep = ""
        prv_tok_text = ""

        prefix = ""
```

```
        modifier = ""

        for tok in nlp(sent):
            print(tok)
            # if token is a punctuation mark then move on to the next
token
            if tok.dep_ == "punct":
                continue

            # check: token is a compound word or not
            if tok.dep_ == "compound":
                prefix = tok.text
                # if the previous word was also a 'compound' then add
the current word to it
                if prv_tok_dep == "compound":
                    prefix = prv_tok_text + " " + tok.text

            # check: token is a modifier or not
            if tok.dep_.endswith("mod"):
                modifier = tok.text
                # if the previous word was also a 'compound' then add
the current word to it
                if prv_tok_dep == "compound":
                    modifier = prv_tok_text + " " + tok.text

            if tok.dep_.find("subj") != -1:
                ent1 = modifier + " " + prefix + " " + tok.text
                prefix = ""
                modifier = ""
                prv_tok_dep = ""
                prv_tok_text = ""

            if tok.dep_.find("obj") != -1:
                ent2 = modifier + " " + prefix + " " + tok.text

            # update variables
            prv_tok_dep = tok.dep_
            prv_tok_text = tok.text

        return [ent1.strip(), ent2.strip()]


    def get_relation(sent):

        doc = nlp(sent)

        # Matcher class object
        matcher = Matcher(nlp.vocab)

        # define the pattern
        pattern = [[
            {'DEP': 'ROOT'},
            {'DEP': 'prep', 'OP': "?"},
            {'DEP': 'agent', 'OP': "?"},
            {'POS': ' ADJ', 'OP': "?"}
```

```
        ]]

        matcher.add("matching_1", pattern)

        matches = matcher(doc)
        k = len(matches) - 1

        span = doc[matches[k][1]:matches[k][2]]

        return span.text


    ##################################################################
################################
    #Sentence Functionality
    ##################################################################
################################

    @app.post("/get/nlp/struct", response_model=GraphResponse)
    async def get_nlp_struct(graph_request: GraphRequest):

        doc = nlp(graph_request.document)

        file_name = uuid.uuid4().hex
        tmp_file = f"static_data/tmp/{file_name}.html"

        with manage_fs('./') as filesystem:
            with filesystem.open(tmp_file, "w") as file:
                file.write('''<!DOCTYPE html>
    <html>
    <head>
        <title>NLTK</title>

    </head>
    <body>''')

                for sent in doc.sents:
                    svg = displacy.render(sent, style="dep")
                    file.write(svg)
                    file.write('<br>')

                file.write('''</body>
    </html>''')

                file.flush()

        ret = GraphResponse()
        ret.answer = f"/static/tmp/{file_name}.html"
        return ret


    ##################################################################
################################
    #Evaluation Functionality
```

```
    ##################################################################
################################

    class GraphNameResponse(BaseModel):
        names: list | None = None

    class GraphEdge(BaseModel):
        subject: str | None = None
        object: str | None = None
        relation: str | None = None

    class GraphDetailsResponse(BaseModel):
        options: dict | None = None
        graph_edges: list[GraphEdge] | None = None

    class GraphEvaluationResponse(BaseModel):
        options: dict | None = None
        graph_edges: list[GraphEdge] | None = None
        common_edge_count: int | None = None
        ethalon_unique_edge_count: int | None = None
        evaluated_unique_edge_count: int | None = None
        common_concept_count: int | None = None
        ethalon_unique_concept_count: int | None = None
        evaluated_unique_concept_count: int | None = None


    @app.get("/get/graph/names", response_model=GraphNameResponse)
    async def get_graph_names():
          subfolders = [ f.name for f in os.scandir('graphs') if
f.is_dir() ]
        ret = GraphNameResponse()
        ret.names = subfolders
        return ret

    @app.get("/get/graph/details",
response_model=GraphDetailsResponse)
    async def get_graph_details(name: str):
        ret = GraphDetailsResponse()
        with open(f'graphs/{name}/model_options.json') as f:
            options = json.load(f)
            print(options)
            ret.options = options

        edges: list[GraphEdge] = []
            df = pd.read_csv(f'graphs/{name}/graph.csv', sep='\t',
encoding='utf-8', header=0)
        for index, row in df.iterrows():
            edge=GraphEdge()
            edge.subject=row["node_1"]
            edge.object=row["node_2"]
            edge.relation = row["edge"]
            edges.append(edge)

        ret.graph_edges = edges
        return ret
```

```
    @app.get("/get/graph/evaluation",
response_model=GraphEvaluationResponse)
    async def get_graph_evaluation(ethalon: str, name: str):
        ret = GraphEvaluationResponse()
        with open(f'graphs/{name}/model_options.json') as f:
            options = json.load(f)
            print(options)
            ret.options = options

        edges: list[GraphEdge] = []
            df  =  pd.read_csv(f'graphs/{name}/graph.csv',  sep='\t',
encoding='utf-8', header=0)
        for index, row in df.iterrows():
            edge=GraphEdge()
            edge.subject=row["node_1"]
            edge.object=row["node_2"]
            edge.relation = row["edge"]
            edges.append(edge)

        ret.graph_edges = edges

        ethalon_edges: list[GraphEdge] = []
          edf  =  pd.read_csv(f'graphs/{ethalon}/graph.csv',  sep='\t',
encoding='utf-8', header=0)
        for index, row in edf.iterrows():
            edge=GraphEdge()
            edge.subject=row["node_1"]
            edge.object=row["node_2"]
            edge.relation = row["edge"]
            ethalon_edges.append(edge)

        common_edge_count = 0
        ethalon_unique_edge_count = 0
        for ethalon in ethalon_edges:
            found = False
            for evaluated in edges:
                        if  ethalon.subject  ==  evaluated.subject  and
ethalon.object == evaluated.object:
                    common_edge_count = common_edge_count + 1
                    found = True
                    break
            if found is False:
                ethalon_unique_edge_count = ethalon_unique_edge_count
+ 1

        evaluated_unique_edge_count = 0
        for evaluated in edges:
            found = False
            for ethalon in ethalon_edges:
                        if  ethalon.subject  ==  evaluated.subject  and
ethalon.object == evaluated.object:
                    found = True
                    break
            if found is False:
```

```
                                        evaluated_unique_edge_count   =
evaluated_unique_edge_count + 1

        ret.evaluated_unique_edge_count = evaluated_unique_edge_count
        ret.common_edge_count = common_edge_count
        ret.ethalon_unique_edge_count = ethalon_unique_edge_count


        evaluated_concepts = []
        for evaluated in edges:
            if evaluated.subject not in evaluated_concepts:
                evaluated_concepts.append(evaluated.subject)
            if evaluated.object not in evaluated_concepts:
                evaluated_concepts.append(evaluated.object)
        print (evaluated_concepts)

        ethalon_concepts = []
        for ethalon in ethalon_edges:
            if ethalon.subject not in ethalon_concepts:
                ethalon_concepts.append(ethalon.subject)
            if ethalon.object not in ethalon_concepts:
                ethalon_concepts.append(ethalon.object)
        print (ethalon_concepts)

        common_concept_count = 0
        ethalon_unique_concept_count = 0
        for ethalon in ethalon_concepts:
            if ethalon in evaluated_concepts:
                common_concept_count = common_concept_count + 1
            else:
                                      ethalon_unique_concept_count   =
ethalon_unique_concept_count + 1

        evaluated_unique_concept_count = 0
        for evaluated in evaluated_concepts:
            if evaluated not in ethalon_concepts:
                                    evaluated_unique_concept_count   =
evaluated_unique_concept_count + 1

                                ret.evaluated_unique_concept_count      =
evaluated_unique_concept_count
        ret.common_concept_count = common_concept_count
                                  ret.ethalon_unique_concept_count      =
ethalon_unique_concept_count

        return ret
    ollama/client.py:
    import os
    import json
    import requests

    BASE_URL             =             os.environ.get('OLLAMA_HOST',
'http://localhost:11434')
```

```
    # Generate a response for a given prompt with a provided model.
    # This is a streaming endpoint, so will be a series of responses.
    #  The  final  response  object  will  include  statistics  and
additional data from the request.
    # Use the callback function to override the default handler.
    def  generate(model_name,  prompt,  system=None,  template=None,
context=None, options=None, callback=None):
        try:
            url = f"{BASE_URL}/api/generate"
            payload = {
                "model": model_name,
                "prompt": prompt,
                "system": system,
                "template": template,
                "context": context,
                "options": options
            }

            # Remove keys with None values
             payload = {k: v for k, v in payload.items() if v is not
None}

               with requests.post(url, json=payload, stream=True) as
response:
                response.raise_for_status()

                  # Creating a variable to hold the context history of
the final chunk
                final_context = None

                  # Variable to hold concatenated response strings if
no callback is provided
                full_response = ""

                     # Iterating over the response line by line and
displaying the details
                for line in response.iter_lines():
                    if line:
                                # Parsing each line (JSON chunk) and
extracting the details
                        chunk = json.loads(line)

                        # If a callback function is provided, call it
with the chunk
                        if callback:
                            callback(chunk)
                        else:
                             # If this is not the last chunk, add the
"response" field value to full_response and print it
                            if not chunk.get("done"):
                                                    response_piece  =
chunk.get("response", "")
                                full_response += response_piece
                                        print(response_piece, end="",
flush=True)
```

```
                        # Check if it's the last chunk (done is true)
                        if chunk.get("done"):
                            final_context = chunk.get("context")

                # Return the full response and the final context
                return full_response, final_context
        except requests.exceptions.RequestException as e:
            print(f"An error occurred: {e}")
            return None, None


    # Create a model from a Modelfile. Use the callback function to
override the default handler.
    def create(model_name, model_path, callback=None):
        try:
            url = f"{BASE_URL}/api/create"
            payload = {"name": model_name, "path": model_path}

             # Making a POST request with the stream parameter set to
True to handle streaming responses
              with requests.post(url, json=payload, stream=True) as
response:
                response.raise_for_status()

                     # Iterating over the response line by line and
displaying the status
                for line in response.iter_lines():
                    if line:
                                # Parsing each line (JSON chunk) and
extracting the status
                        chunk = json.loads(line)

                        if callback:
                            callback(chunk)
                        else:
                            print(f"Status: {chunk.get('status')}")
        except requests.exceptions.RequestException as e:
            print(f"An error occurred: {e}")


    # Pull a model from a the model registry. Cancelled pulls are
resumed from where they left off, and multiple
    # calls to will share the same download progress. Use the
callback function to override the default handler.
    def pull(model_name, insecure=False, callback=None):
        try:
            url = f"{BASE_URL}/api/pull"
            payload = {
                "name": model_name,
                "insecure": insecure
            }

             # Making a POST request with the stream parameter set to
True to handle streaming responses
```

```
                with requests.post(url, json=payload, stream=True) as
response:
                 response.raise_for_status()

                      # Iterating over the response line by line and
displaying the details
                 for line in response.iter_lines():
                     if line:
                                # Parsing each line (JSON chunk) and
extracting the details
                         chunk = json.loads(line)

                         # If a callback function is provided, call it
with the chunk
                         if callback:
                             callback(chunk)
                         else:
                                # Print the status message directly to
the console
                                print(chunk.get('status', ''), end='',
flush=True)

                          # If there's layer data, you might also want
to print that (adjust as necessary)
                         if 'digest' in chunk:
                                print(f" - Digest: {chunk['digest']}",
end='', flush=True)
                                  print(f" - Total: {chunk['total']}",
end='', flush=True)
                                                print(f" - Completed:
{chunk['completed']}", end='\n', flush=True)
                         else:
                             print()
         except requests.exceptions.RequestException as e:
             print(f"An error occurred: {e}")


     # Push a model to the model registry. Use the callback function
to override the default handler.
     def push(model_name, insecure=False, callback=None):
         try:
             url = f"{BASE_URL}/api/push"
             payload = {
                 "name": model_name,
                 "insecure": insecure
             }

              # Making a POST request with the stream parameter set to
True to handle streaming responses
                with requests.post(url, json=payload, stream=True) as
response:
                 response.raise_for_status()

                      # Iterating over the response line by line and
displaying the details
```

```
                for line in response.iter_lines():
                    if line:
                                # Parsing each line (JSON chunk) and
extracting the details
                        chunk = json.loads(line)

                        # If a callback function is provided, call it
with the chunk
                        if callback:
                            callback(chunk)
                        else:
                               # Print the status message directly to
the console
                               print(chunk.get('status', ''), end='',
flush=True)

                         # If there's layer data, you might also want
to print that (adjust as necessary)
                        if 'digest' in chunk:
                               print(f" - Digest: {chunk['digest']}",
end='', flush=True)
                                 print(f" - Total: {chunk['total']}",
end='', flush=True)
                                                print(f" - Completed:
{chunk['completed']}", end='\n', flush=True)
                        else:
                            print()
        except requests.exceptions.RequestException as e:
            print(f"An error occurred: {e}")


    # List models that are available locally.
    def list():
        try:
            response = requests.get(f"{BASE_URL}/api/tags")
            response.raise_for_status()
            data = response.json()
            models = data.get('models', [])
            return models

        except requests.exceptions.RequestException as e:
            print(f"An error occurred: {e}")
            return None


    # Copy a model. Creates a model with another name from an
existing model.
    def copy(source, destination):
        try:
            # Create the JSON payload
            payload = {
                "source": source,
                "destination": destination
            }
```

```
                    response = requests.post(f"{BASE_URL}/api/copy",
json=payload)
            response.raise_for_status()

                # If the request was successful, return a message
indicating that the copy was successful
            return "Copy successful"

        except requests.exceptions.RequestException as e:
            print(f"An error occurred: {e}")
            return None


    # Delete a model and its data.
    def delete(model_name):
        try:
            url = f"{BASE_URL}/api/delete"
            payload = {"name": model_name}
            response = requests.delete(url, json=payload)
            response.raise_for_status()
            return "Delete successful"
        except requests.exceptions.RequestException as e:
            print(f"An error occurred: {e}")
            return None


    # Show info about a model.
    def show(model_name):
        try:
            url = f"{BASE_URL}/api/show"
            payload = {"name": model_name}
            response = requests.post(url, json=payload)
            response.raise_for_status()

            # Parse the JSON response and return it
            data = response.json()
            return data
        except requests.exceptions.RequestException as e:
            print(f"An error occurred: {e}")
            return None


    def heartbeat():
        try:
            url = f"{BASE_URL}/"
            response = requests.head(url)
            response.raise_for_status()
            return "Ollama is running"
        except requests.exceptions.RequestException as e:
            print(f"An error occurred: {e}")
            return "Ollama is not running"
    util/df_helpers:
    import uuid
    import pandas as pd
    import numpy as np
```

```
import random
import seaborn as sns

from util.prompts import extract_relations


def documents2Dataframe(documents) -> pd.DataFrame:
    rows = []
    for chunk in documents:
        row = {
            "text": chunk,
            "chunk_id": uuid.uuid4().hex,
        }
        rows = rows + [row]

    df = pd.DataFrame(rows)
    return df


def df2Graph(dataframe: pd.DataFrame, model=None, options = None)
-> list:
    # dataframe.reset_index(inplace=True)
    results = dataframe.apply(
            lambda row: extract_relations(row.text, {"chunk_id":
row.chunk_id}, model, options), axis=1
    )
    # invalid json results in NaN
    results = results.dropna()
    results = results.reset_index(drop=True)

    # Flatten the list of lists to one single list of entities.
    concept_list = np.concatenate(results).ravel().tolist()
    return concept_list


def graph2Df(nodes_list) -> pd.DataFrame:
    # Remove all NaN entities
        graph_dataframe = pd.DataFrame(nodes_list).replace("  ",
np.nan)
      graph_dataframe = graph_dataframe.dropna(subset=["node_1",
"node_2"])
                                    graph_dataframe["node_1"]
graph_dataframe["node_1"].apply(lambda x: x.lower())
                                    graph_dataframe["node_2"]
graph_dataframe["node_2"].apply(lambda x: x.lower())

    return graph_dataframe

def contextual_proximity(df: pd.DataFrame) -> pd.DataFrame:
    # Melt the dataframe into a list of nodes
    dfg_long = pd.melt(
                df, id_vars=["chunk_id"], value_vars=["node_1",
"node_2"], value_name="node"
    )
    dfg_long.drop(columns=["variable"], inplace=True)
```

```
          # Self join with chunk id as the key will create a link
between terms occuring in the same text chunk.
            dfg_wide = pd.merge(dfg_long, dfg_long, on="chunk_id",
suffixes=("_1", "_2"))
         # drop self loops
                 self_loops_drop = dfg_wide[dfg_wide["node_1"] ==
dfg_wide["node_2"]].index
                                                        dfg2 =
dfg_wide.drop(index=self_loops_drop).reset_index(drop=True)
         # Group and count edges.
         dfg2 = (
             dfg2.groupby(["node_1", "node_2"])
             .agg({"chunk_id": [",".join, "count"]})
             .reset_index()
         )
         dfg2.columns = ["node_1", "node_2", "chunk_id", "count"]
         dfg2.replace("", np.nan, inplace=True)
         dfg2.dropna(subset=["node_1", "node_2"], inplace=True)
         # Drop edges with 1 count
         dfg2 = dfg2[dfg2["count"] != 1]
         dfg2["edge"] = "contextual proximity"
         return dfg2

     def colors2Community(communities) -> pd.DataFrame:
         palette = "hls"
         # Define a color palette
         p = sns.color_palette(palette, len(communities)).as_hex()
         random.shuffle(p)
         rows = []
         group = 0
         for community in communities:
             color = p.pop()
             group += 1
             for node in community:
                     rows += [{"node": node, "color": color, "group":
group}]
         df_colors = pd.DataFrame(rows)
         return df_colors
     util/prompts.py:
     import json
     import ollama.client as client


     def extract_relations(prompt: str, metadata: dict, model: str,
options: dict):

         sys_prompt = (
               "You are a network graph maker who extracts terms and
their relations from a given context. "
              "You are provided with a context chunk (delimited by )
Your task is to extract the ontology "
               "of terms mentioned in the given context. These terms
should represent the key concepts as per the context. \n"
             "Thought 1: While traversing through each sentence, Think
about the key terms mentioned in it.\n"
```

```
                    "\tTerms may include object, entity, location,
organization, person, \n"
                  "\tcondition, acronym, documents, service, concept,
etc.\n"
                "\tTerms should be as atomistic as possible\n\n"
              "Thought 2: Think about how these terms can have one on
one relation with other terms.\n"
                "\tTerms that are mentioned in the same sentence or
the same paragraph are typically related to each other.\n"
                "\tTerms can be related to many other terms\n\n"
               "Thought 3: Find out the relation between each such
related pair of terms. \n\n"
             "Format your output as a list of json. Each element of
the list contains a pair of terms"
            "and the relation between them, like the following: \n"
            "[\n"
            "   {\n"
            '       "node_1": "A concept from extracted ontology",\n'
               '         "node_2": "A related concept from extracted
ontology",\n'
             '        "edge": "relationship between the two concepts,
node_1 and node_2 in one or two sentences"\n'
            "   }, {...}\n"
            "]"
        )

        user_prompt = f"context: {prompt}``` \n\n output: "
                response,   _   =   client.generate(model_name=model,
system=sys_prompt,
                                                    prompt=user_prompt,
options=options)
        try:
            result = json.loads(response)
            result = [dict(item, **metadata) for item in result]
        except:
                print("\n\nERROR ### Here is the buggy response: ",
response, "\n\n")
            result = None
        return result
    vector_embedings/main.py:
    import torch

    from pathlib import Path
    from langchain.document_loaders import HuggingFaceDatasetLoader
    from               langchain.text_splitter               import
RecursiveCharacterTextSplitter
    from langchain.embeddings import HuggingFaceEmbeddings
    from langchain.vectorstores import FAISS
    from     transformers     import     AutoTokenizer,     pipeline,
AutoModelForQuestionAnswering
    from langchain.llms import HuggingFacePipeline
    from langchain.chains import RetrievalQA
    from langchain.document_loaders import DirectoryLoader
```

```
    def main():
        # Define the path to the pre-trained model you want to use
        modelPath = "sentence-transformers/all-MiniLM-l6-v2"

          # Create a dictionary with model configuration options, specifying to use the CPU for computations
        model_kwargs = {'device': 'cpu'}

          # Create a dictionary with encoding options, specifically setting 'normalize_embeddings' to False
        encode_kwargs = {'normalize_embeddings': False}

         # Initialize an instance of HuggingFaceEmbeddings with the specified parameters
        embeddings = HuggingFaceEmbeddings(
            model_name=modelPath,  # Provide the pre-trained model's path
                    model_kwargs=model_kwargs,    # Pass the model configuration options
            encode_kwargs=encode_kwargs  # Pass the encoding options
        )

        a = False
        if a == True:

            folder = "cureus"
            # Input data directory
            input_directory = Path(f"data_input/{folder}")


            # Create a loader instance
                        loader = DirectoryLoader(input_directory, show_progress=True)

            # Load the data
            data = loader.load()

                                # Create an instance of the RecursiveCharacterTextSplitter class with specific parameters.
            # It splits text into chunks of 1000 characters each with a 150-character overlap.
                                                    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=150)

            # 'data' holds the text you want to split, split the text into documents using the text splitter.
            docs = text_splitter.split_documents(data)
            db = FAISS.from_documents(docs, embeddings)
            db.save_local("faiss_index")
        else:
            db = FAISS.load_local("faiss_index", embeddings)

          # Create a retriever object from the 'db' with a search configuration where it retrieves up to 4 relevant splits/documents.
        retriever = db.as_retriever(search_kwargs={"k": 4})
```

```
        pipe = pipeline("text-generation",
                        model="TinyLlama/TinyLlama-1.1B-Chat-v1.0",
                        torch_dtype=torch.bfloat16,
                        device_map="auto")
        print("To leave, type - exit")
        print('\n [green] USER> [/green]')
        question = input()

        while question.lower() != 'exit':
            docs = retriever.get_relevant_documents(question)

              context_text = '\n\n'.join([doc.page_content for doc in
docs])
            messages = [
                {
                    "role": "system",
                         "content": f'''Use the following pieces of
context to answer the question at the end. If you don't know the
answer, just say that you don't know, don't try to make up an answer.
                    context: {context_text}'''
                },
                {"role": "user", "content": f'{question}'}
            ]
                prompt = pipe.tokenizer.apply_chat_template(messages,
tokenize=False, add_generation_prompt=True)
                        outputs = pipe(prompt, max_new_tokens=256,
do_sample=True, temperature=0.7, top_k=50, top_p=0.95)
                                    print('[red]    AI:    [/red]',
outputs[0]["generated_text"][len(prompt):])
            print('\n [green] USER> [/green]')
            question = input()


    if __name__ == '__main__':
        main()
    index.html:
    <!DOCTYPE html>
    <html>
    <head>
        <title>Page Title</title>
        <script src="/static/js/jquery-1.10.2.js"></script>
                                                                <script
src="/static/js/jquery-ui-1.10.4.custom.min.js"></script>
    </head>
    <body>
        <div>
                  <input id="question-input" placeholder="Ask your
question"/>
              <input id="ask-question-btn" type="button" value="Send"
class="btn"/>
            <div id="answer"></div>
        </div>
        <script>
```

```
            $(document).ready(function () {
                $('#ask-question-btn').click(onAskQuestion);
            })

            function onAskQuestion(){
                var question = $('#question-input').val();
                var json = {"question": question};

                $.ajax({
                    url: '/get/answer',
                    data: JSON.stringify(json),
                    type: "POST",

                    beforeSend: function (xhr) {
                                        xhr.setRequestHeader("Accept",
"application/json");
                                    xhr.setRequestHeader("Content-Type",
"application/json");
                    },
                    success: function (response) {
                        $("#answer").html(response.answer);
                    }
                });
            }
        </script>
    </body>
    </html>
    index2.html:
    <!DOCTYPE html>
    <html>
    <head>
        <title>Page Title</title>
        <script src="/static/js/jquery-1.10.2.js"></script>
                                                                    <script
src="/static/js/jquery-ui-1.10.4.custom.min.js"></script>
    </head>
    <body>
        <div>
             <textarea id="document-input" placeholder="Enter text of
the document">
            </textarea>
            <table>
                <tr>
                                                                <td><label
for="temperature">temperature</label></td>
                        <td><input id="temperature" name="temperature"
type="checkbox" ></td>
                          <td><input name="temperature" type="number"
style="display:none"></td>
                </tr>
                <tr>
                    <td><label for="mirostat">mirostat</label></td>
                            <td><input id="mirostat" name="mirostat"
type="checkbox" ></td>
```

```
                                <td> <input name="mirostat" type="number"
style="display:none"></td>
                    </tr>
                    <tr>
                                                                    <td><label
for="mirostat-eta">mirostat-eta</label></td>
                            <td><input id="mirostat-eta" name="mirostat-eta"
type="checkbox" ></td>
                                <td><input name="mirostat-eta" type="number"
style="display:none" ></td>
                    </tr>
                    <tr>
                                                                    <td><label
for="mirostat-tau">mirostat-tau</label></td>
                            <td><input id="mirostat-tau" name="mirostat-tau"
type="checkbox" ></td>
                                <td><input name="mirostat-tau" type="number"
style="display:none"></td>
                    </tr>
                    <tr>
                        <td><label for="num_ctx">num_ctx</label></td>
                                <td><input  id="num_ctx"  name="num_ctx"
type="checkbox" ></td>
                                <td><input  name="num_ctx"  type="number"
style="display:none"></td>
                    </tr>
                    <tr>
                        <td><label for="num_keep">num_keep</label></td>
                                <td><input  id="num_keep"  name="num_keep"
type="checkbox" ></td>
                                <td><input  name="num_keep"  type="number"
style="display:none"></td>
                    </tr>
                    <tr>
                        <td><label for="seed">seed</label></td>
                            <td><input id="seed" name="seed" type="checkbox"
></td>
                                    <td><input  name="seed"  type="number"
style="display:none"></td>
                    </tr>
                    <tr>
                                                                    <td><label
for="num_predict">num_predict</label></td>
                            <td><input id="num_predict" name="num_predict"
type="checkbox" ></td>
                                <td><input name="num_predict" type="number"
style="display:none"></td>
                    </tr>
                    <tr>
                        <td><label for="top_k">top_k</label></td>
                                    <td><input  id="top_k"  name="top_k"
type="checkbox" ></td>
                                <td><input  name="top_k"  type="number"
style="display:none"></td>
                    </tr>
```

```
                <tr>
                    <td><label for="top_p">top_p</label></td>
                                <td><input  id="top_p"  name="top_p"
type="checkbox" ></td>
                                <td><input  name="top_p"  type="number"
style="display:none"></td>
                </tr>
                <tr>
                    <td><label for="tfs_z">tfs_z</label></td>
                                <td><input  id="tfs_z"  name="tfs_z"
type="checkbox" ></td>
                                <td><input  name="tfs_z"  type="number"
style="display:none"></td>
                </tr>
            </table>
            <br>
            <table>
                <tr>
                      <td><label for="context-proximity">Show context
proximity edges</label></td>
                     <td><input id="context-proximity" type="checkbox"
></td>
                </tr>
            </table>
            <br>
            <table>
                <tr>
                                <td><label  for="save-graph">Save  graph
as</label></td>
                    <td><input id="save-graph" type="checkbox" ></td>
                                <td><input  name="save-graph"  type="text"
style="display:none"></td>
                </tr>
            </table>
                      <input  id="upload-document-btn"  type="button"
value="Upload" class="btn"/>
            <div id="content"></div>
        </div>
        <script>

            $(document).ready(function () {
                $('#upload-document-btn').click(onUploadDocument);
                $('input[type="checkbox"]').click(onCheckboxClick);
                $('#save-graph').click(onSaveGraphCheckbox);
            })

            function onCheckboxClick(){
               let value = $(this).is(':checked');
               let name = $(this).attr('name');
               if (value == true)
                {
                    $('input[type="number"][name='+name+']').show();
                }
                else
                {
```

```
                $('input[type="number"][name='+name+']').hide();
            }
        }

        function onSaveGraphCheckbox(){
           let value = $(this).is(':checked');
           if (value == true)
            {

$('input[type="text"][name="save-graph"]').show();
            }
            else
            {

$('input[type="text"][name="save-graph"]').hide();
            }
        }

        function onUploadDocument(){
            var document = $('#document-input').val();
            var inputs = $('input[type="number"][name]:visible')
            var options = {};
            for(var i = 0; i<inputs.length; i++)
            {
                var inp=inputs[i];

options[$(inp).attr('name')]=parseFloat(inp.value);
            }

                                                var    graphName    =
$('input[type="text"][name="save-graph"]').val();

                                            var   showContextProximity   =
$('#context-proximity').is(':checked');

            var json = {"document": document, "options": options,
"showContextProximity": showContextProximity };

            if(graphName!=null && graphName!='')
            {
                json['graphName'] = graphName;
            }

            $.ajax({
                url: '/get/graph',
                data: JSON.stringify(json),
                type: "POST",

                beforeSend: function (xhr) {
                                        xhr.setRequestHeader("Accept",
"application/json");
                                    xhr.setRequestHeader("Content-Type",
"application/json");
                },
                success: function (response) {
```

```
                        window.location.href =response.answer;
                    }
                });
            }
        </script>
    </body>
    </html>
    index3.html:
    <!DOCTYPE html>
    <html>
    <head>
        <title>Page Title</title>
        <script src="/static/js/jquery-1.10.2.js"></script>
                                                                <script
src="/static/js/jquery-ui-1.10.4.custom.min.js"></script>
    </head>
    <body>
        <div>
             <textarea id="document-input" placeholder="Enter text of
the document">
            </textarea>
                     <input  id="upload-document-btn"  type="button"
value="Upload" class="btn"/>
            <div id="content"></div>
        </div>
        <script>

            $(document).ready(function () {
                $('#upload-document-btn').click(onUploadDocument);
            })

            function onUploadDocument(){
                var document = $('#document-input').val();
                var json = {"document": document};

                $.ajax({
                    url: '/get/nlp/graph',
                    data: JSON.stringify(json),
                    type: "POST",

                    beforeSend: function (xhr) {
                                        xhr.setRequestHeader("Accept",
"application/json");
                                    xhr.setRequestHeader("Content-Type",
"application/json");
                    },
                    success: function (response) {
                        window.location.href =response.answer;
                    }
                });
            }
        </script>
    </body>
    </html>
    graph-comparator.html:
```

```
<!DOCTYPE html>
<html>
<head>
    <title>Page Title</title>
    <script src="/static/js/jquery-1.10.2.js"></script>
                                                        <script
src="/static/js/jquery-ui-1.10.4.custom.min.js"></script>
    <style>
        .big-table
        {
            width: 100%;
        }
        .main-col
        {
            width: 50%;
        }
        td
        {
            vertical-align: top;
            padding-left: 10px;
        }
        .td-padding
        {
            padding-top: 20px;
        }
        tr
        {
            border-bottom: 1px solid black;
        }
        .info-table
        {
            border-collapse: collapse;
        }
    </style>
</head>
<body>
    <table class="big-table">
        <tr>
            <td class="main-col">
                <label>Ethalon Graph</label>
                <select id="ethalon-graph" ></select>
            </td>
            <td class="main-col">
                <label>Evaluated Graph</label>
                <select id="evaluated-graph" ></select>
            </td>
        </tr>
        <tr>
            <td id="ethalon-options-holder" class="td-padding">

            </td>
            <td id="evaluated-options-holder" class="td-padding">

            </td>
        </tr>
```

```
            <tr>
                <td id="ethalon-graph-holder" class="td-padding">

                </td>
                <td id="evaluated-graph-holder" class="td-padding">

                </td>
            </tr>
            <tr>
                <td class="td-padding">

                </td>
                <td id="evaluated-stats-holder" class="td-padding">

                </td>
            </tr>

        </table>

        <script>

            $(document).ready(function () {
               onUploadDocument();
            })


            function onUploadDocument(){
                $.ajax({
                    url: '/get/graph/names',
                    type: "GET",

                    beforeSend: function (xhr) {
                                        xhr.setRequestHeader("Accept",
"application/json");
                                    xhr.setRequestHeader("Content-Type",
"application/json");
                    },
                    success: initUI
                });
            }

            function initUI(response)
            {

                var ethalonGraphCombo = $('#ethalon-graph');
                var evaluatedGraphCombo = $('#evaluated-graph');
                ethalonGraphCombo.append($('<option>', {
                        value: '',
                        text : 'Select Graph'
                    }));
                    evaluatedGraphCombo.append($('<option>', {
                        value: '',
                        text : 'Select Graph'
                    }));
                for(var i=0; i<response.names.length; i++)
```

```
                {
                    var name = response.names[i];
                    ethalonGraphCombo.append($('<option>', {
                        value: name,
                        text : name
                    }));
                    evaluatedGraphCombo.append($('<option>', {
                        value: name,
                        text : name
                    }));
                }

ethalonGraphCombo.change(onEthalonGraphComboSelChange);

evaluatedGraphCombo.change(onEvaluatedGraphComboSelChange);
            }
            function onEthalonGraphComboSelChange()
            {
                var val = this.value;
                if (val==null || val=='')
                {
                    $("#ethalon-graph-holder").html('');
                    return;
                }

                $.ajax({
                    url: '/get/graph/details?name='+val,
                    type: "GET",

                    beforeSend: function (xhr) {
                                        xhr.setRequestHeader("Accept",
"application/json");
                                xhr.setRequestHeader("Content-Type",
"application/json");
                    },
                    success: renderEthalonGraph
                });
            }

            function renderEthalonGraph(response)
            {
                var holder = $("#ethalon-options-holder");
                holder.html('');
                var table = $('<table>', {class: 'info-table'});
                holder.append(table);
                var row = $('<tr>');
                table.append(row);
                var col = $('<td>');
                row.append(col);
                col.html('Options:');
                var names = Object.keys(response.options)
                for(var i=0; i<names.length; i++)
                {
                    row = $('<tr>');
                    table.append(row);
```

```
                col = $('<td>');
                row.append(col);
                col.html(names[i]);
                col = $('<td>');
                row.append(col);
                col.html(response.options[names[i]]);
            }

            holder = $("#ethalon-graph-holder");
            holder.html('');
            table = $('<table>', {class: 'info-table'});
            holder.append(table);
            row = $('<tr>');
            table.append(row);
            col = $('<td>');
            row.append(col);
            col.html('Graph:');
            for(var i = 0; i<response.graph_edges.length; i++)
            {
                row = $('<tr>');
                table.append(row);

                edge = response.graph_edges[i];
                col = $('<td>');
                row.append(col);
                col.html(edge.subject);
                col = $('<td>');
                row.append(col);
                col.html(edge.object);
                col = $('<td>');
                row.append(col);
                col.html(edge.relation);
            }
        }

        function onEvaluatedGraphComboSelChange()
        {
            var ethalon = $('#ethalon-graph').val();
            if (ethalon==null || ethalon=='')
            {
                $("#evaluated-graph-holder").html('');
                return;
            }

            var val = this.value;
            if (val==null || val=='')
            {
                $("#evaluated-graph-holder").html('');
                return;
            }

            $.ajax({
                                                                    url:
'/get/graph/evaluation?ethalon='+ethalon+'&name='+val,
                type: "GET",
```

```
                    beforeSend: function (xhr) {
                                        xhr.setRequestHeader("Accept",
"application/json");
                                    xhr.setRequestHeader("Content-Type",
"application/json");
                    },
                    success: renderEvaluatedGraph
                });
            }

            function renderEvaluatedGraph(response)
            {
                var holder = $("#evaluated-options-holder");
                holder.html('');
                var table = $('<table>', {class: 'info-table'});
                holder.append(table);
                var row = $('<tr>');
                table.append(row);
                var col = $('<td>');
                row.append(col);
                col.html('Options:');
                var names = Object.keys(response.options)
                for(var i=0; i<names.length; i++)
                {
                    row = $('<tr>');
                    table.append(row);
                    col = $('<td>');
                    row.append(col);
                    col.html(names[i]);
                    col = $('<td>');
                    row.append(col);
                    col.html(response.options[names[i]]);
                }

                holder = $("#evaluated-graph-holder");
                holder.html('');
                table = $('<table>', {class: 'info-table'});
                holder.append(table);
                row = $('<tr>');
                table.append(row);
                col = $('<td>');
                row.append(col);
                col.html('Graph:');
                for(var i = 0; i<response.graph_edges.length; i++)
                {
                    var row = $('<tr>');
                    table.append(row);

                    edge = response.graph_edges[i];
                    var col = $('<td>');
                    row.append(col);
                    col.html(edge.subject);
                    col = $('<td>');
                    row.append(col);
```

```
        col.html(edge.object);
        col = $('<td>');
        row.append(col);
        col.html(edge.relation);
    }


    holder = $("#evaluated-stats-holder");
    holder.html('');
    table = $('<table>', {class: 'info-table'});
    holder.append(table);
    row = $('<tr>');
    table.append(row);
    col = $('<td>');
    row.append(col);
    col.html('Statistics:');
    row = $('<tr>');
    table.append(row);
    col = $('<td>');
    row.append(col);
    col.html('Common edge count');
    col = $('<td>');
    row.append(col);
    col.html(response.common_edge_count);

    row = $('<tr>');
    table.append(row);
    col = $('<td>');
    row.append(col);
    col.html('Ethalon unique edge count');
    col = $('<td>');
    row.append(col);
    col.html(response.ethalon_unique_edge_count);

    row = $('<tr>');
    table.append(row);
    col = $('<td>');
    row.append(col);
    col.html('evaluated unique edge count');
    col = $('<td>');
    row.append(col);
    col.html(response.evaluated_unique_edge_count);

    row = $('<tr>');
    table.append(row);
    col = $('<td>');
    row.append(col);
    col.html('Common concept count');
    col = $('<td>');
    row.append(col);
    col.html(response.common_concept_count);

    row = $('<tr>');
    table.append(row);
    col = $('<td>');
```

```
                row.append(col);
                col.html('Ethalon unique concept count');
                col = $('<td>');
                row.append(col);
                col.html(response.ethalon_unique_concept_count);

                row = $('<tr>');
                table.append(row);
                col = $('<td>');
                row.append(col);
                col.html('Evaluated unique concept count');
                col = $('<td>');
                row.append(col);
                col.html(response.evaluated_unique_concept_count);
            }

        </script>
    </body>
    </html>
    nltk.html:
    <!DOCTYPE html>
    <html>
    <head>
        <title>Page Title</title>
        <script src="/static/js/jquery-1.10.2.js"></script>
                                                                <script
src="/static/js/jquery-ui-1.10.4.custom.min.js"></script>
    </head>
    <body>
        <div>
             <textarea id="document-input" placeholder="Enter text of
the document"></textarea>
                      <input  id="upload-document-btn"  type="button"
value="Upload" class="btn"/>
            <div id="content"></div>
        </div>
        <script>

            $(document).ready(function () {
                $('#upload-document-btn').click(onUploadDocument);
            })

            function onUploadDocument(){
                var document = $('#document-input').val();
                var json = {"document": document};

                $.ajax({
                    url: '/get/nlp/struct',
                    data: JSON.stringify(json),
                    type: "POST",

                    beforeSend: function (xhr) {
                                        xhr.setRequestHeader("Accept",
"application/json");
```

```
                                        xhr.setRequestHeader("Content-Type",
"application/json");
                        },
                        success: function (response) {
                            window.location.href =response.answer;
                        }
                    });
                }
            </script>
        </body>
        </html>
```

ДОДАТОК В

"Методи побудови мереж термінів на основі текстового аналізу та штучного інтелекту"

Виконав:Мірошниченко Михайло Андрійович, група КА-04

Керівник: канд. техн. наук, ст. викл. каф. ММСА, Савастьянов В.В.

Актуальність:

З розвитком інформаційних технологій обсяг даних, що зберігаються і обробляються, стрімко зростає. Векторний пошук та побудова графів знань є новими технологіями для ефективного керування цими даними. Принцип використання LLM, забезпечує покращену точність та ефективність в аналізі великих обсягів інформації. Дослідження даних методів є актуальним для підвищення якості інформаційних систем.

Дослідження

Об’єкт дослідження: Об’єктом дослідження є опрацювання набору текстових даних та приведення їх у зручний для розуміння формат.

Предмет дослідження: На основі LLM створити графи знань, які зможуть працювати з будь-яким текстом та оптимізуватимуть його в зручні для розуміння графи, які можна буде перевірити на точність.

Мета дослідження: Метою роботи є створення алгоритмів для побудови графів, та їх перевірки, зокрема реалізувати можливість для користувача самостійно вибирати налаштування моделі та робити перевірки з наданням статистики з різницею графів, та з можливістю подивитись принцип за яким відбувається синтаксичний аналіз речень.

Постановка задачі

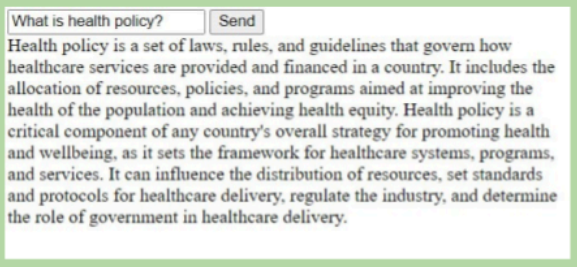
Задачі дослідження:

1. Аналіз існуючих методів векторного пошуку та побудови графів знань.
2. Розробка методів на основі моделі LLM.
3. Реалізація запропонованих методів.
4. Оцінка ефективності реалізованих методів шляхом аналізу різниці вершин та ребер графу.

Основні результати дипломної роботи

Реалізація векторного пошуку:

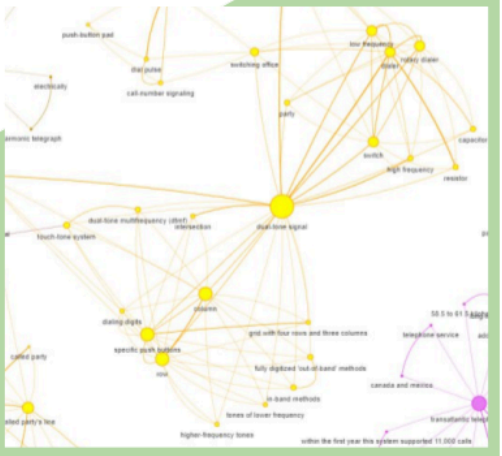
- Векторний пошук було реалізовано з використанням побудованого пайплайну та бази даних FAISS
- Розроблено алгоритм векторного пошуку, що забезпечує високу точність пошуку інформації.
- Було побудовано інтерфейс для векторного пошуку



Основні результати дипломної роботи

Побудова графів знань:

- На основі моделі Ollama реалізовано один з двох методів побудови графів знань.
- Графи знань дозволяють структуровано представити взаємозв'язки між різними елементами даних.
- Для цього метода було реалізовано підбір налаштувань, за допомогою якого можна коригувати рівень генерації LLM
- Було реалізовано їх збереження, для подальшого вивчення



Основні результати дипломної роботи

Побудова графів знань:

- На основі моделі NLP було реалізовано один другий метод побудови графів знань.
- Цей метод демонструє альтернативний метод, який є швидшим, проте не є таким же структурованим та наповненим

Основні результати дипломної роботи

Оцінка графів знань:

- Проведено аналіз побудованих графів знань шляхом порівняння кількості вершин та ребер до і після реалізації.
- Було проведено аналіз моделей з двома різними наборами налаштувань, проте для чіткості експерименту було проведено перевірку генерацій за головним параметром - температурою, на його основі було виявлено, що найкращим вибором для температури буде 0,5, через найбільшу кількість співпадінь з іншими налаштуваннями

	кількість вершин	кількість ребер
t=0,1	178	648
t=0,3	203	613
t=0,5	196	862
t=0,7	156	370
t=1	175	684

	t=0,1	t=0,3	t=0,5	t=0,7	t=1
t=0,1	(178, 648)	(119, 233)	(108, 239)	(91, 116)	(101, 136)
t=0,3	(119, 233)	(203, 613)	(114, 233)	(93, 96)	(85, 145)
t=0,5	(108, 239)	(114, 233)	(196, 862)	(117, 78)	(103, 220)
t=0,7	(91, 116)	(93, 96)	(117, 78)	(156, 370)	(66, 63)
t=1	(101, 136)	(85, 145)	(103, 220)	(66, 63)	(175, 684)

Основні результати дипломної роботи

Оцінка графів знань:

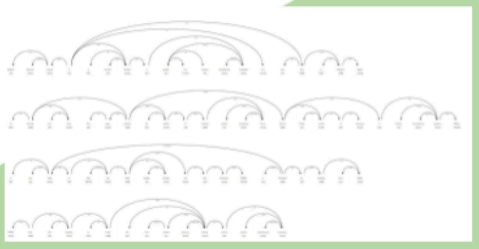
- На основі перевірки температури було побудовано експеримент з додатковим використанням параметру mirostat, який робить відповіді менш передбачуваними
- Експеримент показав, що кращий збіг результатів генерації буде в тексті, який розповідає про одне поняття та є обширним за своїми розмірами.

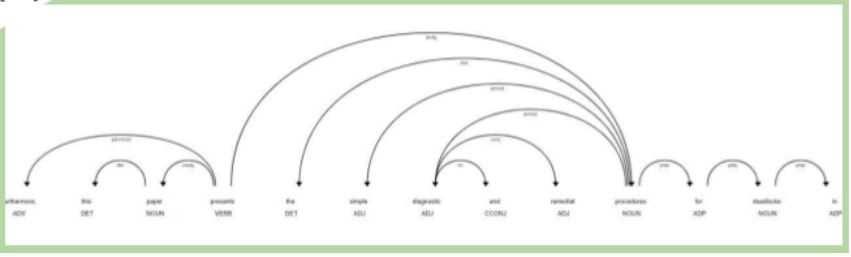
	Еталонний граф		Спільні значення		Перевірений граф	
	ребра	вершини	ребра	вершини	ребра	вершини
Biology[12]	84	30	1	7	111	41
CureUs[13]	464	115	73	67	507	99
Phone[14]	1227	328	41	44	294	59
Climate[15]	539	117	75	58	308	78
Petri[16]	233	55	24	24	272	65

Основні результати дипломної роботи

Оцінка побудовою графу речень:

- Одним з можливих етапів перевірки було реалізовано побудову графів речення
- У цьому пункті перевіряється те, як відбувається побудова речень і завдяки цьому можна отримати приблизне уявлення про розуміння тексту LLM





Наукова новизна

Запропоновано систему побудови графів та їх перевірку на точність, що дозволить оптимізувати використану модель Ollama та задавати їй налаштування для кожної ситуації.

Висновки

Висновки:

- Реалізовані методи векторного пошуку та побудови графів знань показали гарні результати.
- Аналіз графів знань підтвердив доцільність використання запропонованих методів для більш детального представлення інформації.

Подальші дослідження

Розробка методів для автоматичного оновлення графів знань у реальному часі.

Впровадження запропонованих методів у реальні інформаційні системи для подальшого тестування та вдосконалення.

Покращення точності та створення за допомогою більшої кількості тестів, для можливого переходу до синтезу конфігурації

Дякую за увагу