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: «Формування ієрархічних стратегій, заснованих на фактах, з використанням LLM»

Виконав:

студент IV курсу, групи КА-05 _____

Клепачевський Дмитро Русланович

Керівник:

ст. викладач, к.т.н. _____

Савастьянов Володимир Володимирович

Консультант з економічного розділу: _____

к.е.н. Рощина Надія Василівна

Консультант з нормоконтролю: _____

к.ф.-м.н. Статкевич Віталій Михайлович

Рецензент: професор, д.т.н., завідувач кафедри ІБ _____

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

Клепачевському Дмитру Руслановичу

1. Тема роботи «Формування ієрархічних стратегій, заснованих на фактах, з використанням LLM», керівник роботи Савастьянов Володимир Володимирович, старший викладач, кандидат технічних наук, затверджені наказом по університету від «___»_____ 2024 р. №_____

2. Термін подання студентом роботи: 11.06.2024.

3. Вихідні дані до роботи: 9787 повідомлень з різних каналів мережі Telegram, що містили інформацію про дрони та їх технології з 24 лютого 2023 року по 15 травня 2024 року; текстова інформація з 33 сайтів, що містили різну інформацію про технології дронів.

4. Зміст роботи: дослідження предметної області, теоретичний опис використаних технологій, програмна реалізація та аналіз результатів, функціонально-вартісний аналіз програмного продукту.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
економічний	Рощина Н.В., доц., к.е.н.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формування тематики (напряму) дослідження	05.02.2024 – 27.02.2024	виконано
2	Аналіз актуальності задач стосовно тематики дослідження	28.02.2024 – 07.03.2024	виконано
3	Аналіз відомих результатів стосовно тематики дослідження	08.03.2024 – 14.03.2024	виконано
4	Формулювання задач дослідження	15.03.2024 – 25.03.2024	виконано
5	Уточнення теми дипломної роботи	26.03.2024 – 04.04.2024	виконано
6	Збір та парсинг даних	05.04.2024 – 10.04.2024	виконано
7	Розробка та дослідження LLM моделей, виконання запитів	11.04.2024 – 25.04.2024	виконано
8	Створення ієрархічних структур із отриманих результатів	25.04.2024 – 09.05.2024	виконано
9	Оформлення пояснювальної записки	10.05.2024 – 24.05.2024	виконано
10	Підготовка презентації для захисту	25.05.2024 – 03.06.2024	виконано
11	Попередній захист дипломної роботи	04.06.2024 – 05.06.2024	виконано

Студент _____ Дмитро КЛЕПАЧЕВСЬКИЙ

Керівник _____ Володимир САВАСТЬЯНОВ

РЕФЕРАТ

Дипломна робота: 125 с., 44 рис., 8 табл., 2 дод., 24 джерела.

ВЕЛИКІ МОВНІ МОДЕЛІ, ІЄРАРХІЧНІ СТРУКТУРИ, ТЕКСТОВА АНАЛІТИКА, ТРАНСФОРМЕРИ, ВІРТУАЛЬНИЙ АСИСТЕНТ

Темою роботи є створення ієрархічних структур, які засновані на фактах, і зроблені за допомогою використання Великих мовних моделей (LLM). Отримані ієрархічні структури є ключовими складовими для синтезу стратегій, та відображають їх структуру.

Об’єктом дослідження є генерація ієрархічних структур з векторизованого корпусу текстів заданої тематики (на прикладі бази знань про дрони) із застосуванням LLM.

Предметом дослідження є різні Великі мовні моделі, такі як ChatGPT та Llama.

Метою даної роботи є представлення великих баз знань у ієрархічному вигляді задля спрощеного відображення великих даних за допомогою LLM. Завдяки цьому, буде створено віртуального асистента, який буде натренований на великому обсязі даних, і зможе відображати комплексну інформацію у зручному вигляді ієрархічних структур.

Актуальність роботи пов’язана з великою кількістю релевантної інформації, яку можна швидко та якісно обробити та представити у зручному вигляді за допомогою LLM.

В результаті роботи на мові Python були побудовані та навчені різні моделі, такі як ChatGPT та Llama, моделі були навчені на зібраній базі знань про дрони та були порівняні між собою за різними метриками оцінювання ієрархічних структур. За допомогою засобу Gephi, результати, отримані за допомогою LLM, були відображені у зручному вигляді з метою візуалізації.

ABSTRACT

Bachelor‘s Thesis: 125 p., 44 fig., 8 tab., 2 app., 24 references.

LARGE LANGUGE MODELS, HIERARCHICAL STRUCTURES, TEXT ANALYTICS, TRANSFORMERS, VIRTUAL ASSISTANT

The topic of the work is the creation of hierarchical structures based on facts and made by using Large Language Models (LLM). The resulting hierarchical structures are key components for synthesizing strategies and reflect their structure.

The object of the study is the generation of hierarchical structures from a vectorized corpus of texts on a given topic (for example, a knowledge base about drones) using LLM.

The subject of the research is various Large Language Models, such as ChatGPT and Llama.

The purpose of this work is to represent large knowledge bases in a hierarchical form for a simplified display of big data using LLM. Therefore, a virtual assistant will be created that will be trained on a large amount of data and will be able to display complex information in a convenient form of hierarchical structures.

The relevance of the work is related to the large amount of relevant information that can be quickly and efficiently processed and presented in a convenient way using LLM.

As a result of the work, various models, such as ChatGPT and Llama, were built and trained in Python, trained on the collected drone knowledge base, and compared with each other using various hierarchical structure evaluation metrics. Using the Gephi tool, the results obtained by LLM were displayed in a convenient way for visualization purposes.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI	Artificial Intelligence, Штучний Інтелект
API	Application Programming Interface
CoT	Chain of Thoughts, Ланцюжок думок
CSV	Comma Separated Values
LLM	Large Language Models, Велика Мовна Модель
NLP	Natural Language Processing, Обробка природньої мови
NN	Neural Network, Нейронна Мережа

ВСТУП

В останні роки великі штучні нейронні мережі почали ставати все популярніше, і вони широко використовуються в абсолютно різних галузях. Це не обійшло стороною і галузь текстової обробки і аналітики, де великі масиви даних потребують швидкого та інтелектуального процесу обробки. Великі мовні моделі (Large Language Models) – це моделі штучних нейронних мереж, які навчаються на великих масивах даних з метою бути здатними розуміти та генерувати природну мову. Великі мовні моделі (LLM) можуть бути навчанні на великих даних, і після того як вони натреновані на цих даних, LLM можуть узагальнити великий текст у бажаній формі, і в цій роботі буде розглянуто узагальнення у ієрархічно-структурованій формі. Для людини прочитати величезні тексти може зайняти декілька днів і більше, в той час як LLM може бути натренована на цих даних за декілька годин, або навіть хвилин, і після цього у зручному вигляді може узагальнити всі дані. Саме завдяки цьому, LLM використовуються в академії та індустрії, щоб економити години або дні, та збільшувати продуктивність команди.

Тим не менше, однією з найбільших проблем Великих мовних моделей є галюцинації. LLM можуть генерувати текст, який ніяк не збігається із базою знань, на яких LLM була натренована. Для вирішення цієї проблеми існує наука, інженерія промптів (Prompt Engineering). Промпти – це текстові запити до моделі, після яких модель починає генерувати текст відповідно до запиту. Для того, щоб модель генерувала розумний текст, треба щоб і промпт був правильно складений. З цієї причини, в цій роботі розглядаються дві технології роботи з промптами з метою отримання ієрархічних структур – Chain of Thoughts (Ланцюжок думок) та звичайний промптинг.

Метою даної роботи є Ієрархічне формування розумних стратегій з використанням Великих мовних моделей. Завдяки цьому, можна створити

віртуального експерта, який буде натренований на великих об’ємах даних, та завдяки правильно створеним запитам, буде генерувати необхідну інформацію у структуровано-ієрархічному вигляді. Спочатку, буде зібрано великі масиви даних із публічних джерел про використання ворожих дронів. Далі, різні LLM будуть натреновані на цих даних, і після цього за допомогою Ланцюжка думок будуть створені ієрархічно-структуровані результати, аби отримати узагальнену інформацію про базу знань у структурованому вигляді. Отримані ієрархічні структури будуть ключовими компонентами для створення стратегій та будуть відображати їх структуру.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Ієрархічні структури

Створення ієрархічних структур значно допомагає та зберігає час, оскільки великі масиви даних можуть бути швидко оброблені за допомогою Large Language Models (LLM), і після створення ієрархічної структури, людина може отримати інформацію, представлену у зрозумілому структурованому вигляді.

Онтологія – це формалізоване представлення знань у вигляді набору понять та відносин між ними. Саме в цьому розумінні ієрархічні структури і можуть бути представлені. Онтології активно використовуються в абсолютно різних галузях: онтології генів, в біомедицині, та в системах добування інформації (Information retrieval systems) [1].

Оскільки можна будувати абсолютно різні онтології та ієрархічні структури за допомогою LLM, то відповідно треба мати й метрики оцінювання отриманих результатів з метою порівняння та оцінки моделей [1].

Структуризація великих масивів даних, які доступні в публічному доступі в Інтернеті, є дуже важливою задачею, оскільки може зберегти багато часу та представити дані у зручному форматі. Важливим етапом у комплексних дослідженнях є представлення бази знань з обраної тематики у ієрархічному вигляді, з метою надати можливість для автоматизованої обробки [2].

За допомогою аналізу текстового корпусу (зібраної бази знань), інформацію можна представити у ієрархічному вигляді за допомогою техніки відображення графів. Завдяки такому представленню даних з текстового корпусу, можна отримати можливість сформувати та охопити цілу галузь знань, яку можна взяти за основу для подальшого будування онтологій [3].

За допомогою активного промптингу (ітеративне використання LLM декілька разів) можна створювати причинно-наслідкові мережі та візуалізувати їх за допомогою засобу Gephi [4]. Згенерувавши CSV (comma separated values) файл, його з легкістю можна завантажити у Gephi та зробити відповідну візуалізацію. Перевага засобу й у тому, що це може бути використано не лише для створення причинно-наслідкових мереж, а й для ієрархічних структур, які розглядаються в даній роботі.

1.2 Текстова аналітика з використанням LLM

Підходи text-to-text (текст-до-тексту) навчання моделей активно використовуються для вирішення основних типових завдань Natural Language Processing (NLP), де ці підходи базуються на Generative Artificial Intelligence models (генеративний штучний інтелект) з використанням Large Language Models (LLM) з технікою налаштування промптів (prompt tuning) [5].

Генеративні моделі з використанням LLM досягають неймовірних результатів, і відповідно до нещодавнього досліду, 5 з 7 ключових завдань NLP отримали ще кращі результати з технологією Generative LLM [5]. Ці факти показують те, що використання віртуальних асистентів на основі LLM із коректним використанням технік з запитами (промптами) мають дати ефективні результати у задача з формуванням ієрархічних структур за допомогою віртуальних асистентів LLM.

Впровадження та інтеграція Large Language Models показує ефективні результати у домені розумної обробки документів (Intelligent Document Processing) [6]. Саме такий підхід може бути використаний і у створенні ієрархічних структур по заздалегідь підготовленій бази даних. Спершу, дані збираються з багатьох ресурсів у вигляді документів, наприклад, PDF

документів. Саме після цього, за допомогою технік розумної обробки текстових документів, які доступні для інтеграцій з LLM, можна обробити дані і створити базу знань на основі відповідних документів. Однак, подібне тренування на базі даних може призвести до перенавчання або до зсуву (bias), чи збільшення обчислювальних можливостей. Тому, під час тренуванні моделі на базі даних, яка може бути представлена у вигляді набору документів, треба приділяти особливу увагу проблемі перенавчання.

1.2.1 Віртуальні асистенти

Віртуальні асистенти відіграють найважливішу роль у допомозі з вирішенням різноманітних завдань, включно із завданням створення ієрархічних структур [7]. Відповідно, віртуальні асистенти можуть значно пришвидшити будь-яку роботу: віртуальний асистент на базі LLM (LLM-powered virtual assistant) здатний узагальнити та підсумувати велику базу знань у вигляді ієрархічної структури дуже швидко, навіть якщо база знань має десятки і сотні текстових файлів.

Generative AI моделі демонструють та підтверджують свою ефективність і у завданнях підтримки рішень (decision support), основуючись на методах аналізу ієрархічних структур [8]. А саме, LLM-асистенти відкрили широкі можливості навіть у використані для систем кібербезпеки завдяки генеративним моделям штучного інтелекту [8].

1.3 Large Language Models

LLM – це великі мовні моделі, які являють собою великі складні нейронні мережі, які можуть бути навчання на вузькій базі знань конкретної галузі, і після цього використані у цілях отримання бажаної інформації з великого масиву даних. У розрізі тематики цієї роботи, LLM будуть використовувати у форматі text-to-text, що означає, що на вхід будуть подані текстові дані, і в цей же час на виході модель також подає текстові дані.

Зовсім свіже дослідження [5] показало, що Великі мовні моделі показали неймовірні результати у 7 ключових завдання обробки природньої мови (Natural Language Processing). Всі сім завдань були вирішені за допомогою однієї самої мовної моделі, GatorTronGPT, яка налічує 20 мільярдів параметрів. У цій роботі активно було використано Prompt tuning, аби налаштувати промпт таким чином, щоб він давав найкращий можливий результат для кожного завдання. Врешті-решт, 5 з 7 ключових завдань отримали видатні результати, що доказує ефективність LLM у цілій низці завдань типу text-to-text.

1.4 Chain of Thoughts

Chain of Thoughts (Ланцюжок думок) – це технологія промптингу, яка використовує набір проміжних кроків міркування, завдяки чому значно покращує можливість Великих мовних моделей для комплексних завдань [9]. Відповідні дослідження показали, що технологія Ланцюжка думок значно покращує результати промптів завдяки логічним міркуванням, які використовує мовна модель та прописує у відповіді на запит.

Інший дослід [10] також підкреслює чудову можливість мислення та набору міркувань моделлю за допомогою технології Ланцюжка думок. Однак, дослідники наголошують, що якість запитів дуже залежить від демонстрацій, що були надані моделі, і часто створення таких баз знань вручну займає багато часу, та виявляється неефективним. Дійсно, це може зайняти багато часу і використання цієї технології втратить свій першопочатковий сенс. Саме тому, було запровадження технологію Синтетичного Промптингу (Syntethic Prompting) під архітектурою Ланцюжка думок. Цей метод запроваджує декілька штучних прикладів до моделі, і таким чином змушує модель згенерувати більше штучних прикладів вже самостійно. Врешті-решт, модель вибирає найбільш ефективні демонстрації для того, щоб забезпечити якомога кращі результати. В цьому ж досліді, було відразу перевірено якість синтетичного промптингу (рис. 1.1).

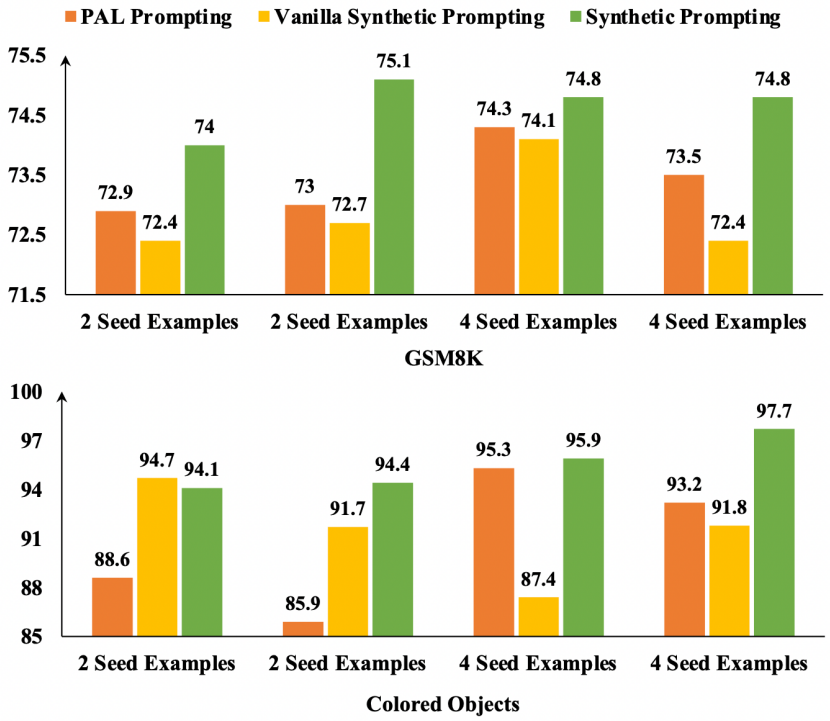


Рисунок 1.1 – Використання CoT з синтетичним промптингом [10]

Як результат, технологія показала найкращі показники на всіх окрім одного прикладах, де були взяти два різні набори даних – GSM8K та Colored

Objects. Зазначимо, що майже всюди, наявній технології вдалось значно перевершити результати.

Інші дослідники [11] також підкреслюють, що Великі мовні моделі виявились дуже ефективними у напрямку комплексних завдань з природною мовою. Також, наголошено, що технологія Ланцюжка думок виявила можливість значно покращити результати роздумів моделі, за допомогою проміжних кроків, які зроблені перед отриманням фінального результату. Також зазначено, що технологія не тільки не дає краще логічні висновки, які зроблені моделлю, а також підвищує інтерпретованість результатів, здатність контролювати систему, та гнучкість системи. Дійсно, саме завдяки цим показникам, технологія Ланцюжка думок надає чудову можливість створенню автономних мовних віртуальних агентів, що і є головним фокусом цієї роботи.

Відповідно до дослідження [11], мовний віртуальний агент працює за ітеративною схемою, в якій присутні план, рішення, та контроль (рис. 1.2).

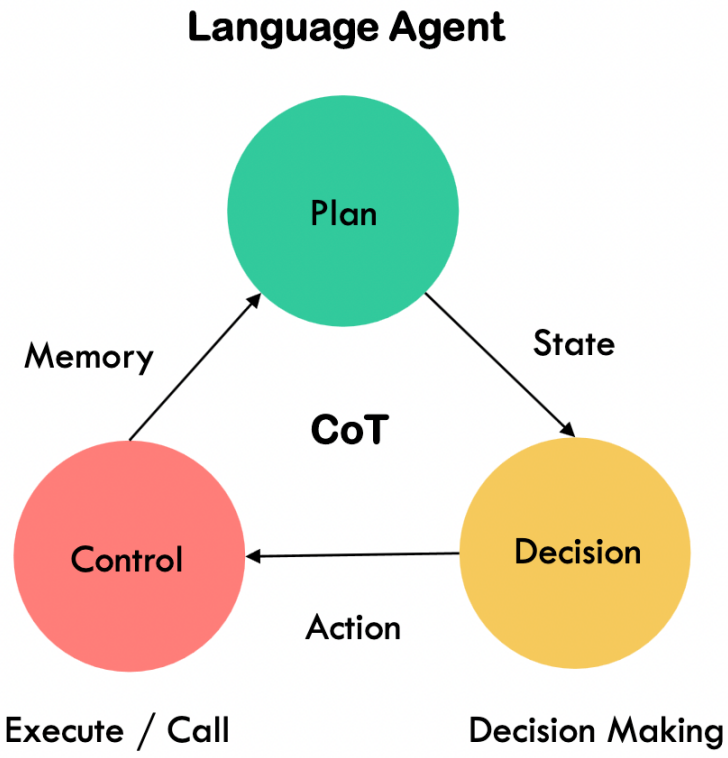


Рисунок 1.2 – Огляд роботи CoT

Саме завдяки цій схемі, технологія Chain of Thoughts (CoT, Ланцюжок думок) і виконує свою роботу.

Також, було проведене дослідження з розглядом активного промптингу за допомогою Великих мовних моделей з технологією Ланцюжка думок [12]. У додаток до інших досліджень, також зазначено, що Ланцюжок думок є дуже ефективною технологією для покращення якості моделі для комплексних завдань із важкими етапами мислення моделі. Ця наукова робота пропонує поєднати технологію Chain of Thoughts разом з Active-Prompt (Активний Промптинг), щоби адаптувати Великі мовні моделі до ряду різних завдань з штучними запитами.

Для цього, за допомогою Few-Shot Prompting, де надається декілька прикладів із заздалегідь прописаними відповідями, пишеться довгий запит до моделі, аби вона могла вивчити патерни та підлаштуватись під завдання (рис. 1.3).

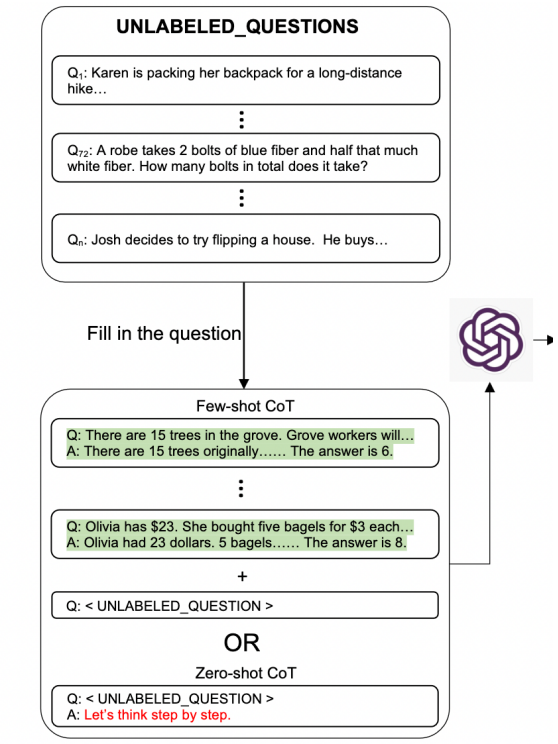


Рисунок 1.3 – Використання Few-shot CoT

Спочатку, є лише запитання та завдання без заздалегідь відомих відповідей. Але, декілька питань заповнюються із відповідями, і відразу подаються до моделі й з іншими питаннями, які вже не мають відповідей. Після всього цього, велика мовна модель, вчиться на цих даних, і налаштовується саме на такий тип завдань. Дійсно, це показує ефективність, і результати стають значно кращими аніж за ті, що були без даної технології.

Також, було вивчено взаємодію технології Ланцюжка думок з такими великими мовними моделями, як GPT-3.5 та GPT-4 [13]. Дослід було проведено на даних з автоматичним оцінюванням написання студентських робіт. Було розглянуто ряд можливих проблем, наприклад такі, як брак інтерпретованості результатів моделі. Разом з технологією Ланцюжка думок, було розглянуто і Zero-Shot Learning та Few-Shot Learning.

Дослідження демонструє, що запити зроблені за допомогою Ланцюжка думок значно підвищують точність оцінювання, збільшуючи її на 13,44% у сценаріях з нульовою кількістю спроб (Zero Shot Prompting) і на 3,7% (Few Shot Prompting) у сценаріях з кількома спробами [13]. Підказки CoT, забезпечили більш збалансовану точність у різних категоріях навичок, підкреслюючи важливість міркувань, що стосуються конкретної предметної області. GPT-4 перевершив GPT-3.5 у різних завданнях з підрахунку балів, особливо в жадібній вибірці з одним викликом, продемонструвавши покращення на 8,64%. Дослідження підкреслює потенціал Великих мовних моделей (LLM) у створенні зрозумілого та інтерпретованого автоматичного скорингу, що підвищує точність і прозорість (рис. 1.4).

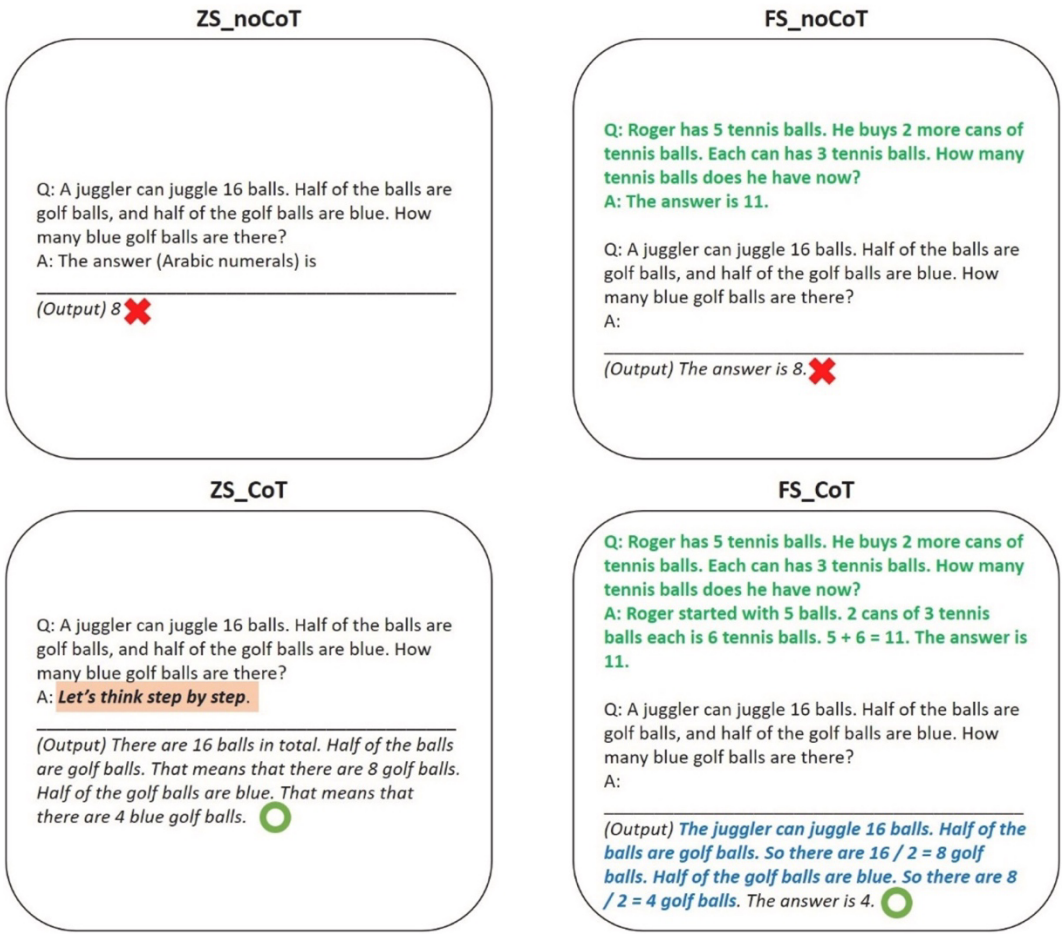


Рисунок 1.4 – Приклад кращої роботи LLM з технологією Ланцюжка думок [13]

Дійсно, як можна помітити, результати без технології Ланцюжка думок дали взагалі неправильні результати. В той час, технологія Ланцюжка думок з Zero Shot Prompting та Few Shot Prompting, обидві дали правильні результати. У випадку з Few Shot Prompting, модель можна вважати більш зрозумілою та більш інтерпретованою.

Відповідно до тематики даної роботи, метою якою є створення віртуального асистента з можливістю створення ієрархічної структури, було знайдено дуже схожий дослід [14]. Було описано використання Великих мовних моделей, щоб забезпечити впровадження віртуальних асистентів з пам'яттю та можливістю вивчати неструктуровані дані, та інші патерни. Також, було використано технологію Chain of thought (Ланцюжок думок) для того, щоб покращити дії, зроблені віртуальним асистентом.

У цій роботі, віртуальний асистент на базі LLM, адаптується до вхідної інформації, відповідної до користувача (рис. 1.5).

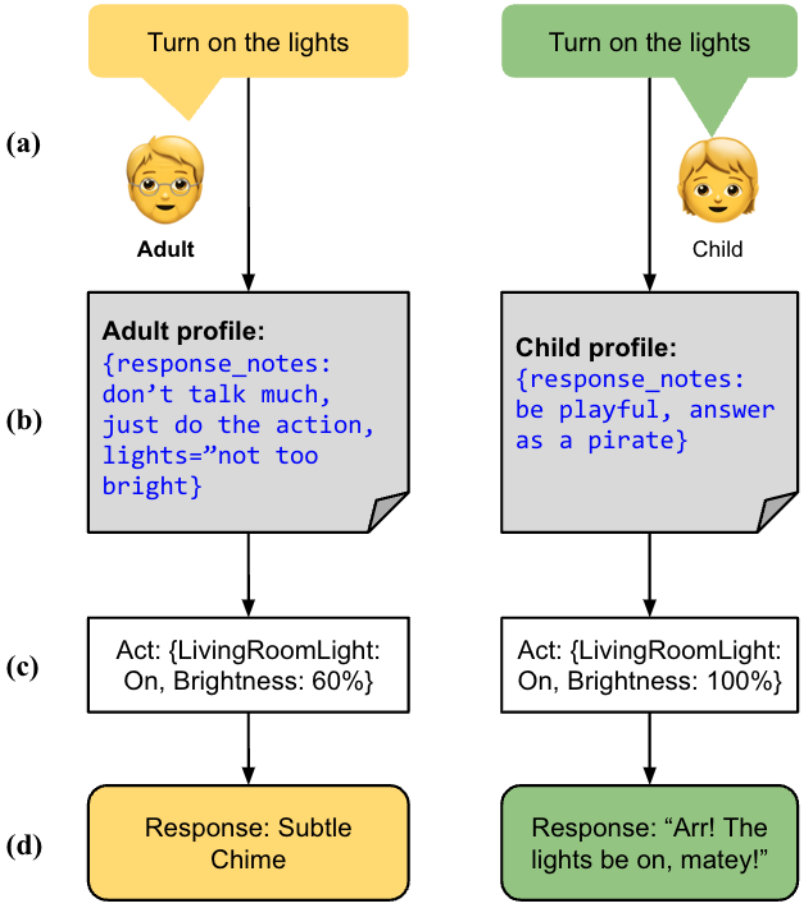


Рисунок 1.5 – Віртуальний асистент адаптується до поведінки користувача [14]

Ланцюжок думок використано на базі контексту даних користувача, і після цього визначається відповідна дія Великої мовної моделі. Завдяки Ланцюжку думок, модель здатна вирішити комплексне завдання і зрозуміти, чи вхідна інформація поступає від дорослої людини, чи ні.

1.5 Висновки до розділу 1

Створення віртуальних асистентів на основі відповідного до спеціальної тематики текстового корпусу, стало дуже популярною технікою, що здатна зберігати час та підвищувати ефективність у різних галузях. Коректне тренування моделі, та використання її з різними техніками промптингу, призводить до успішного створення ієрархічних структур по зібраній базі знань.

Мета цієї роботи – розглянути різні LLM моделі та натренувати їх на текстовому корпусі даних про дрони. Після тренування, LLM здатні вивчити досвід, отриманий з текстового корпусу, та внаслідок цього, буде отримано віртуального асистента, який за допомогою такої техніки як Chain of Thoughts зможе створювати ієрархічні структури з інформації про дрони.

2 РОЗРОБКА ПІДХОДУ СТВОРЕННЯ ПРОТОТИПІВ СТРАТЕГІЙ У ВИГЛЯДІ ІЄРАРХІЧНИХ СТРУКТУР З ВИКОРИСТАННЯМ LLM

2.1 Постановка задачі

Мета дипломної роботи – створення системного підходу до створення прототипів стратегій у вигляді ієрархічних структур, заснованих на фактах, і з використанням LLM.

Об’єктом дослідження є створення ієрархічних структур з векторизованого текстового корпусу заданої тематики (на прикладі бази знань про дрони) із використанням LLM.

Задано:

- ресурси із даними для парсингу бази знань;
- проблема, що потребує покрового рішення.

Потрібно:

- проаналізувати існуючі підходи для створення ієрархічних стратегій з використанням LLM;
- виконати збір текстового корпусу з наявних ресурсів;
- натренувати різні LLM моделі на зібраному текстовому корпусі;
- використати різні техніки запитів для створення ієрархічних структур за допомогою LLM;
- візуалізувати отримані ієрархічні структури;
- порівняти результати моделей між собою за допомогою різних метрик.

Загальну реалізацію даної роботи представлено у вигляді діаграми (рис. 2.1), що описує системний підхід щодо реалізації ієрархічних стратегій за допомогою LLM, які дозволять реалізувати ієрархічні структури (рис. 2.1).

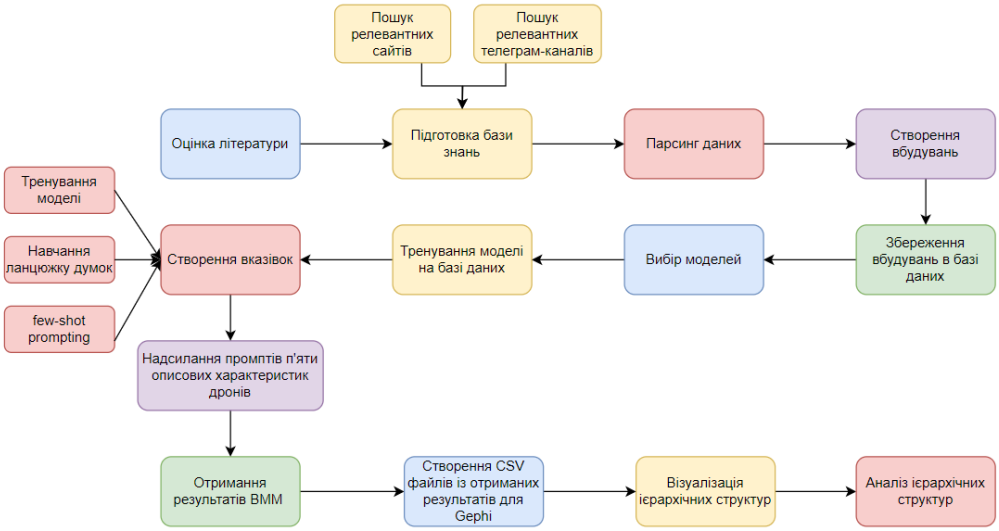


Рисунок 2.1 Системний підхід до реалізації ієрархічних стратегій за допомогою LLM

Спочатку, реалізовується пошук релевантних сайтів та телеграм каналів з метою підготовки бази знань, що засновано на досконалій оцінці літератури та схожих досліджень. Після цього, виконується парсинг даних з обраних ресурсів, та створюються числові репрезентації для зібраної бази знань. Числові репрезентації зберігаються в базі даних, після чого виконується тренування обраних LLM моделей, з метою вивчити загальні характеристики та отримати досвід з зібраної текстової інформації.

Для того, щоб LLM моделі генерували якісні результати, виконується створення відповідних вказівок для моделей, а саме: Ланцюжок думок (Chain of Thoughts) та Few Shot Prompting. За допомогою цих технологій, надсилаються запити до моделей, з метою створити описові характеристики дронів по зібраним текстовим корпусам. Коли LLM моделі відповіли на запит, то за допомогою застосунку Gephi створюється візуалізація ієрархічних структур, та порівняння моделей між собою.

2.2 Ієрархічні структури та стратегії

Ієрархічні структури є важливим інструментом для організації та управління складними системами. Вони забезпечують розуміння та контроль за різними рівнями системи, дозволяючи ефективно переходити від загальних стратегій до конкретних дій.

2.2.1 Представлення основи стратегій у вигляді ієрархічних структур

Від глобального рівня стратегій у вигляді ієрархічних структур можна перейти до локального рівня, за допомогою декомпозиції та виділення основних об’єктів, суб’єктів та систем, які у свою чергу теж можна декомпозувати, доки не залишаться лише листи з діями над цими об’єктами, суб’єктами та системами.

Процес декомпозиції дозволяє розбити більш глобальні стратегії на дрібніші компоненти, що стає основою для локальних стратегій. Це надає можливість отримати детальніше розуміння стратегічних рішень у складних системах.

Декомпозиція являє собою виділення основних об’єктів, суб’єктів та систем, які є ключовими елементами у виконанні стратегічних завдань.

Таким чином, використання ієрархічних структур для представлення стратегій надає можливість складним системам ефективно переходити від глобальних до локальних рівнів управління, забезпечуючи детальний контроль та управління на кожному етапі. Ця декомпозиція значно підвищує ефективність реалізації стратегічних цілей у складних системах.

Оскільки метою цієї роботи є побудова ієрархічних структур за допомогою LLM, то треба впровадити метрики, за допомогою яких можна оцінити та порівняти різні побудовані ієрархічні структури. Нижчеописані метрики використовуються для оцінювання складності побудованої

ієрархічної структури, що дає змогу оцінити наскільки комплексні результати дає кожна з використаних LLM моделей. Розрахунок всіх метрик надає можливість порівняти моделі між собою, та з'ясувати, які з них дають більш складні ієрархічні структури.

2.2.2 Метрики оцінювання ієрархічних структур

Загальна кількість вузлів ієрархії (Total Number of Nodes in Hierarchy) – це метрика оцінювання ієрархічних структур, яка використовується з метою оцінки розміру та складності ієрархічної структури, такої як онтологія. Метрика визначає загальну кількість вузлів (елементів), що входять до складу структури. Загальна кількість вузлів ієрархії розраховується за формулою (2.1):

$$N = \sum_{i=1}^{L} n_i \qquad (2.1)$$

, де N – загальна кількість вузлів; L – кількість рівнів в ієрархії; n_i – кількість вузлів на кожному рівні i.

Завдяки цій метриці можна оцінити складність даної ієрархічної структури, що є важливою оцінкою, оскільки це дає змогу зрозуміти наскільки складні структури може створити задана LLM модель.

Загальна кількість вузлів, що співпадають (Total Number of Matching Nodes) – це інша метрика оцінки ієрархічних структур, яка визначає кількість вузлів, які є однаковими або еквівалентними в обох ієрархіях. Ця метрика може бути корисною для аналізу подібності між різними ієрархічними структурами. Метрика рахується за формулою (2.2):

$$M = \sum_{i=1}^{N} \delta(a_i, b_i) \qquad (2.2)$$

, де M – загальна кількість вузлів, що співпадають; N – загальна кількість вузлів у меншій з двох структур; a_i, b_i – вузли з першої та другої структур відповідно; $\delta(a_i, b_i)$ – функція, яка дорівнює 1, якщо вузли з двох ієрархій співпадають, та дорівнює 0 інакше.

Ця метрика корисна тим, що дозволяє порівняти попарно різні ієрархічні структури, та може допомогти зрозуміти які моделі дають схожі структури в кінцевому результаті.

Кількість вузлів на останньому рівні (Number of Nodes at the Last Level) – це ще одна метрика оцінки, яка використовується для оцінювання розподілу вузлів в ієрархічній структурі. Ця метрика визначає кількість вузлів, що знаходяться на найнижчому рівні ієрархії. Обчислюється метрика за формулою (2.3):

$$L = |\{v \in V \mid level(v) = max_i(level_i)\}| \qquad (2.3)$$

Глибина вузлів гілок (Depth of Branch Nodes) – це метрика, що використовується для оцінки глибини вузлів у ієрархічній структурі. Метрика визначає максимальну глибину від кореневого вузла до найглибшого вузла в гілці. Метрика обчислюється за формулою (2.4):

$$D = \max(depth(v) \mid v \in V) \qquad (2.4)$$

, де D – максимальна глибина вузлів гілок; $depth(v)$ – глибина вузла v, тобто кількість рівнів від кореневого вузла до вузла v; V – множина всіх вузлів у ієрархічній структурі.

Метрики, перелічені вище, будуть використані з метою оцінки отримання різних ієрархічний структур з метою аналізу результатів від різних LLM моделей за допомогою різних CoT та ToT технологій промптів.

2.3 Визначення Великої мовної моделі

Велика мовна модель (Large Language Model, LLM) - це алгоритм глибокого навчання (Deep learning), який може виконувати різноманітні завдання з обробки природної мови (NLP). Великі мовні моделі використовують моделі-трансформери і навчаються на великих наборах даних, а отже, є великими. Це дозволяє їм розпізнавати, перекладати, прогнозувати або генерувати текст чи інший контент.

Великі мовні моделі також називають нейронними мережами (neural networks, NNs), які є обчислювальними системами, натхненними людським мозком. Ці нейронні мережі працюють за допомогою мережі вузлів, які є багаторівневими, подібно до нейронів. Великі мовні моделі обробляють текстовий запит за допомогою різноманітних технологій для отримання кращого кінцевого результату (рис. 2.2)

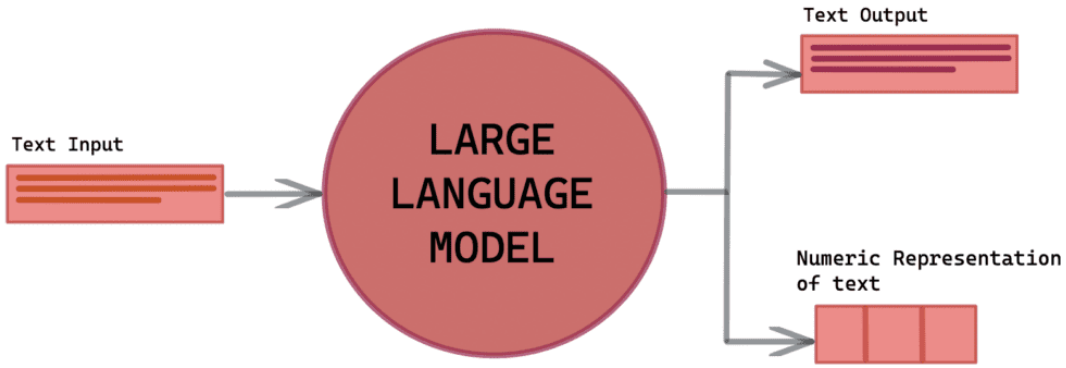


Рисунок 2.2 – Велика мовна модель

Окрім навчання людської мови для програм штучного інтелекту (Artificial Intelligence, AI), Великі мовні моделі також можна навчити виконувати різноманітні завдання, такі як написання програмного коду тощо. Як і людський мозок, Великі мовні моделі необхідно попередньо навчити, а потім доопрацювати, щоб вони могли вирішувати завдання класифікації текстів, відповідати на запитання, узагальнювати документи і генерувати тексти. Їхні можливості розв'язання проблем можуть бути застосовані в таких галузях, як охорона здоров'я, фінанси та розваги, де Великі мовні моделі слугують для різноманітних застосувань обробки природної мови, таких як переклад, чат-боти, асистенти штучного інтелекту тощо.

Великі мовні моделі також мають велику кількість параметрів, які схожі на спогади, які модель збирає в процесі навчання. Ці параметри уявляють як базу знань моделі.

2.3.1 Модель трансформер

Модель трансформер – це найпоширеніша архітектура великої мовної моделі. Вона складається з енкодера та декодера. Ця модель обробляє дані, розбиваючи вхідні дані на токени, а потім одночасно виконує математичні рівняння, щоб виявити взаємозв'язки між токенами. Це дозволяє комп'ютеру побачити закономірності, які побачила б людина, якби отримала такий самий запит.

Трансформери працюють з механізмами самоуважності, що дозволяє моделі навчатися швидше, ніж традиційні моделі, такі як моделі з довгою короткочасною пам'яттю. Саме самоуважність дозволяє моделі

трансформер розглядати різні частини послідовності або весь контекст речення, щоб генерувати прогнози.

Енкодер у трансформері переводить вхідну послідовність відображення символів $(x_1, \dots, x_n)$ до послідовності неперервних відображень $z = (z_1, \dots, z_n)$. Отримавши z, декодер генерує вихідну послідовність символів $(y_1, \dots, y_n)$ [18]. На кожному кроці, модель трансформер є авторегресивною, тобто кожен попередній згенерований символ є додатковою вхідною інформацією при генерації тексту (рис. 2.3).

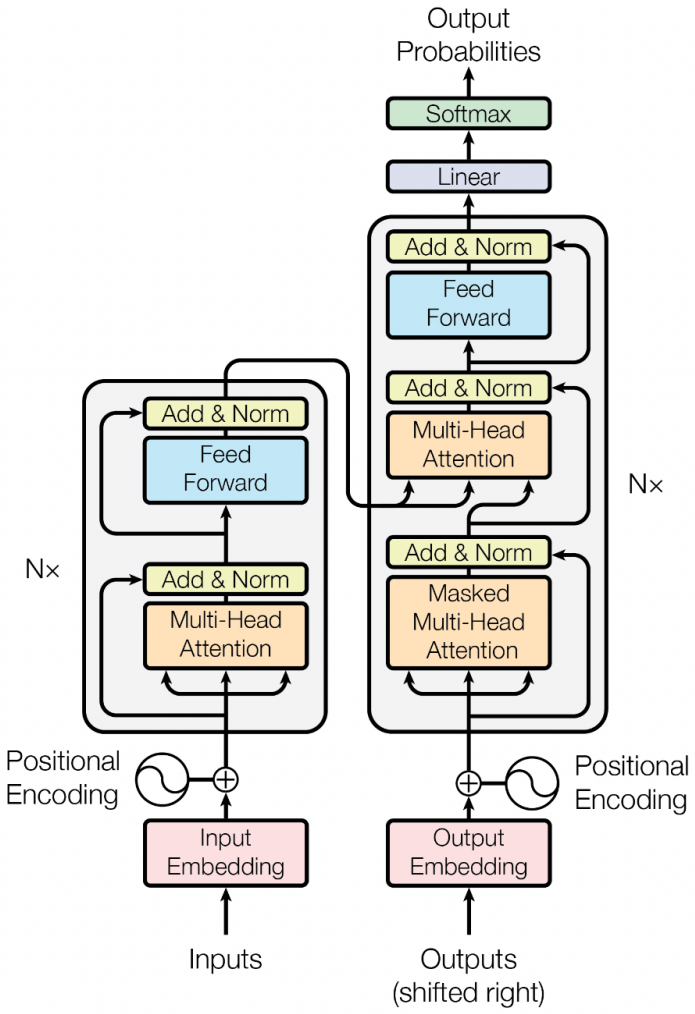


Рисунок 2.3 – Архітектура моделі трансформера [18]

2.4 Ключові компоненти LLM

LLM складаються з декількох шарів нейронних мереж (рис. 2.4). Рекурентні шари, шари прямого поширення, шари вбудовування та шари уваги працюють разом, щоб обробити вхідний текст і генерувати вихідний контент.

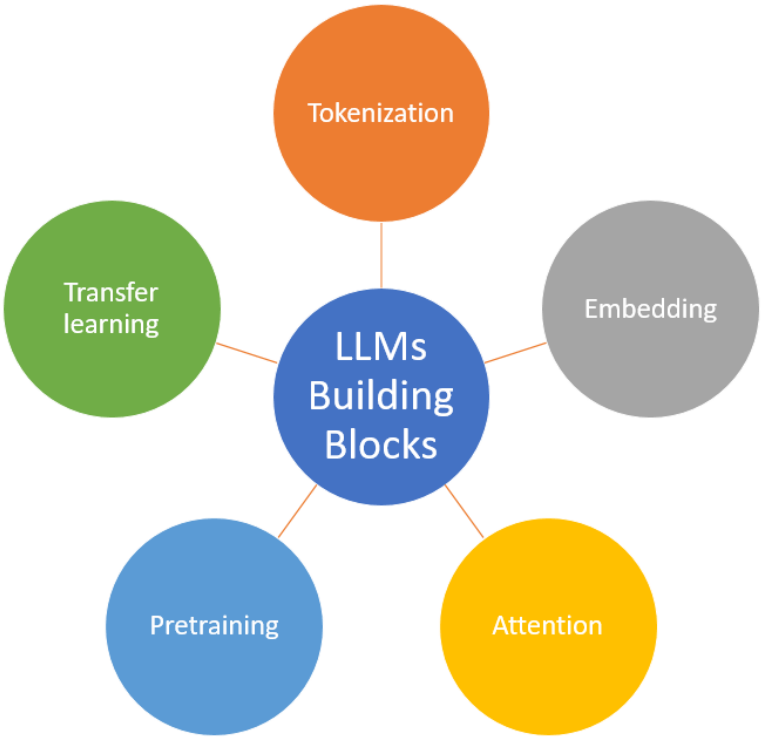


Рисунок 2.4 – Складові великої мовної моделі [19]

2.4.1 Шар токенізації

Токенізація це найперший етап підготовки моделі - перетворення послідовності тексту в окремі слова, підслова або токени, які модель може зрозуміти. У LLM токенізація зазвичай виконується за допомогою алгоритмів підслів, таких як Byte Pair Encoding (BPE) або WordPiece, які

розбивають текст на менші одиниці, що охоплюють як часті, так і рідкісні слова.

2.4.2 Шар вбудування (Embedding Layer)

Шар вбудовування (embedding layer) створює репрезентації з вхідного тексту, фіксуючи семантичне і синтаксичне значення тексту, щоб модель могла зрозуміти контекст. Шар вбудування складається з двох компонент: Positional Encoding та Input Embedding, після чого дві компоненти додаються.

Є багато різних варіантів обчислення позиційного енкодингу (positional encoding), але найпоширенішим є варіант з використанням синусів та косинусів [18]:

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right)$$

, де pos – це позиція слова, i – розмірність, d_{model} – розмірність шару вбудування (Embedding layer).

Також, модель трансформер використовує pre-trained (попередньо треновані) embeddings слів, завдяки чому кожне слово отримує свою репрезентацію з розмірністю d_{model}.

Word embeddings (вбудування слів) – це низькорозмірне векторне представлення набору слів із бази знань (vocabulary), метою якого є відобразити семантичну схожість між словами. Ці векторні представлення є

векторами з дійсними значеннями, і слова, які схожі за значенням, будуть близько у векторному просторі.

2.4.3 Шар прямого зв’язку

Шар прямого зв'язку (feedforward layer) великої мовної моделі складається з декількох повністю з'єднаних шарів, які перетворюють вхідні вставки. Таким чином, ці шари дозволяють моделі отримувати абстракції більш високого рівня - тобто розуміти наміри користувача за допомогою введеного тексту.

2.4.4 Шар нормалізації

Шар нормалізації (Layer Normalization) напряму оцінює нормалізаційну статистику від суми вхідних даних до нейронів в схованих шарах моделі, таким чином, що нормалізація не впроваджує ніяких нових залежностей [20].

Ця техніка гарно працює з моделями трансформерів, що використовуються LLM, та завдяки цьому підвищується якість моделі та час тренування.

Шар нормалізації рахує значення по всім прихованим нейронам за формулами (2.5) та (2.6):

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad (2.5)$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \qquad (2.6)$$

, де H означає кількість нейронів у прихованому шарі.

2.4.5 Попереднє навчання

Попереднє навчання (pre-training) це процес тренування великої мовної моделі на великому наборі даних (бази знань), перед тим, як почати використовувати модель для конкретного завдання. Під час попереднього навчання модель вивчає загальні мовні закономірності, зв'язки між словами та інші фундаментальні знання. Результатом цього процесу є попередньо навчена модель, яку можна точно налаштувати, використовуючи менший набір даних для конкретного завдання, що значно зменшує кількість маркованих даних і час навчання, необхідний для досягнення високої продуктивності при виконанні різних завдань для обробки природньої мови.

2.4.6 Механізм уваги

Механізм уваги (attention mechanism) дозволяє мовній моделі зосередитися на конкретних частинах вхідного тексту, які мають відношення до поставленого завдання. Цей шар дозволяє моделі генерувати найточніший результат.

Multi-Head Attention є ключовим компонентом моделі трансформера, яка використовує декілька одиничних механізмів уваги. Механізм уваги може бути описаний як перетворення трьох компонент: Query (запит), Key

(ключ), Value (значення), до вихідного значення. Матриця вихідних значень обчислюється за формулою (2.7):

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad (2.7)$$

, де d_k – це розмірність векторів Q (Query) та V (value) [18].

2.4.7 Трансферне навчання

Трансферне навчання це техніка використання знань, отриманих під час попередньої підготовки, і застосування їх до нової, пов'язаної з ними задачі. У контексті LLM, трансферне навчання передбачає точне налаштування попередньо навченої моделі на меншому, звуженому для завдання наборі даних, щоб досягти високої продуктивності при виконанні цього завдання. Перевага трансферного навчання полягає в тому, що воно дозволяє моделі скористатися величезною кількістю загальних мовних знань, отриманих під час попереднього навчання, зменшуючи потребу у великих наборах даних і довготривалому навчанні для кожної нової задачі.

2.4.8 Енкодер та Декодер

Енкодер складається з набору шести ідентичних шарів. Кожен шар має свої підшари. Перший шар – це Multi-Head Self-Attention механізм, та другий – це простий шар feed-forward нейронної мережі. Також, реалізовується skip connection навколо кожного з двох підшарів, який завершується із шаром нормалізації. [18]

Відповідно, виходом кожного з підшарів є:

$$LayerNorm(x + Sublayer(x))$$

, де $Sublayer(x)$ – це функція, яка реалізована підшаром.

Декодер, в свою чергу, також складається з набору шести ідентичних шарів. У додаток до двох підшарів, декодер також має третій підшар, який виконує Multi-Head Attention на вихідні дані з шарів енкодеру. Аналогічно до енкодеру, реалізовані skip зв'язки навколо кожного з підшарів, які потім подаються у шар нормалізації. [18]

2.4.9 Параметр temperature

Оскільки часто Великі мовні моделі використовуються у задачах, які вимагають креативність, був створений параметр температури моделі (model's temperature). Параметр температури регулює кількість випадковості (randomness) моделі, яка призводить до більш різносторонніх результатів [21]. Відповідно, параметр температури часто зветься параметром креативності моделі.

Температура – це гіперпараметр моделі, який ми знаходимо в стохастичних моделях, щоб регулювати випадковості у вибірковій процедурі [22]. Трансформація софтмакс (softmax function) використовується для нелінійних трансформацій, перетворюючи вхідні дані на розподіл ймовірностей, які в сумі дають 1.

Вищі значення температури моделі збільшують ентропію, що призводить до більшої кількості випадковості та невизначеності в генеративних процесах [22]. Зазвичай, значення для параметру t в інтервалі

[0, 2], та в ситуації, коли t = 0, це є випадком жадібного відбору (greedy sampling), тобто коли завжди береться токен з найвищою ймовірністю (формула (2.8)).

$$softmax(z)_i = \frac{\exp\left(\frac{z_i}{t}\right)}{\sum_j^n \exp\left(\frac{z_j}{t}\right)} \qquad (2.8)$$

, де $z \in \mathbb{R}^n$.

2.5 Chain of Thoughts

Розглянемо процес мислення під час розв'язування складного логічного завдання, наприклад, багатокрокової математичної задачі. Зазвичай ми розбиваємо задачу на проміжні кроки і вирішуємо кожен з них, перш ніж дати остаточну відповідь.

Ланцюжок думок (Chain Of Thoughts) - це техніка промпт інжинірингу, призначена для того, щоб спрямувати Великі мовні моделі (LLM) на послідовність проміжних кроків, що ведуть до бажаної відповіді. Цей підхід розширює можливості міркувань LLM (Великої мовної моделі), дозволяючи їм вирішувати один крок за раз, замість того, щоб вирішувати всю проблему одночасно. Цей метод особливо корисний для вирішення складних проблем, які важко або неможливо вирішити за один крок. Ланцюжок думок дозволяє моделям декомпонувати багатокрокові задачі на проміжні кроки (рис. 2.5), а це означає, що додаткові обчислення можуть бути виділені для задач, які вимагають більшої кількості кроків міркувань.

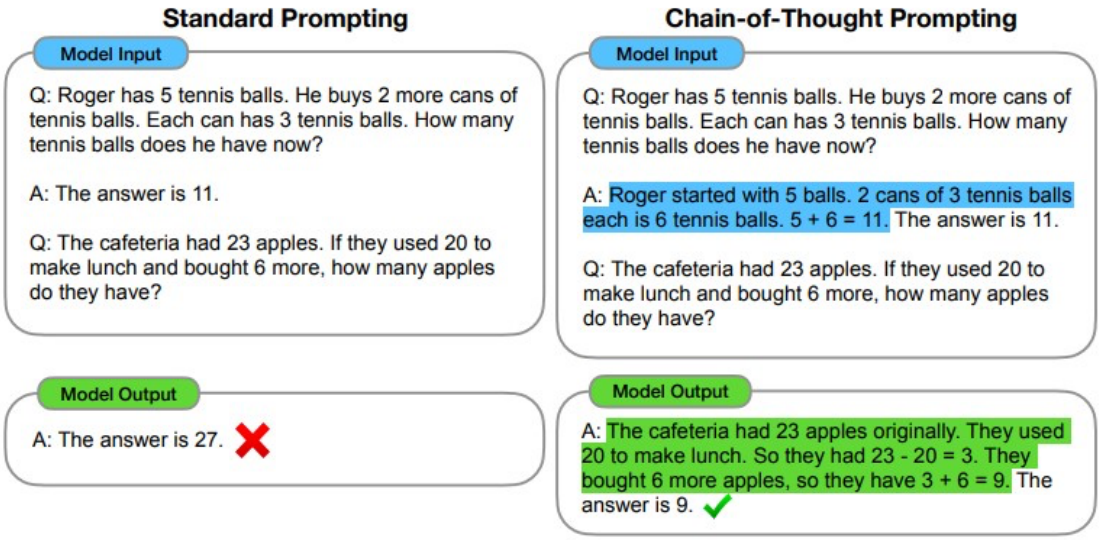


Рисунок 2.5 – Використання промптингу з Ланцюжком думок

Окрім цього, Ланцюжок думок забезпечує інтерпретоване вікно в поведінку моделі, припускаючи, як вона могла дійти до певної відповіді, і надаючи можливості для налагодження там, де шлях міркувань пішов невірно (хоча повна характеристика обчислень моделі, які підтримують відповідь, залишається відкритим питанням).

Ланцюжок думок можна застосовувати для вирішення таких завдань, як текстові математичні задачі, логічні міркування та маніпуляції зі символами, і він потенційно підходить для будь-якої задачі, яку людина здатна розв'язати за допомогою мови.

2.6 LangChain

LangChain - це фреймворк, призначений для розробки додатків, які ефективно та раціонально використовують Великі мовні моделі (LLM). Він надає набір інструментів, що допомагають розробникам створювати додатки, що використовують можливості LLM, таких як GPT-3, у

структурований та модульний спосіб. LangChain фокусується на спрощенні інтеграції LLM з іншими обчислювальними та інформаційними ресурсами для створення складних, інтелектуальних додатків [23].

Вкажемо ключові аспекти та компоненти LangChain.

1. Ланцюжки (Chains): LangChain вводить поняття «ланцюжків», які є послідовністю операцій або кроків, що можуть включати виклики мовних моделей, перетворення даних і взаємодію із зовнішніми системами. Ланцюжки дозволяють розробникам створювати складні робочі процеси, які використовують LLM.
2. Запити (Prompts): керування та створення правильних запитів є важливою частиною роботи з LLM. LangChain надає інструменти для створення запитів, включаючи шаблони та параметризацію, що полегшує створення послідовних та ефективних промптів для різних завдань.
3. Пам'ять: багато додатків вимагають збереження контексту при багатократній взаємодії з LLM. LangChain включає механізми управління пам'яттю, що дозволяє програмам запам'ятовувати попередні взаємодії та підтримувати контекст у часі.
4. Агенти: LangChain підтримує створення агентів, які є автономними сутностями, що можуть виконувати завдання за допомогою LLM. Агенти можуть бути розроблені для обробки певних типів запитів, виконання фонових завдань або взаємодії з користувачами в діалоговому режимі.
5. Інтеграція: фреймворк полегшує інтеграцію Великих мовних моделей із зовнішніми API, базами даних та іншими сервісами, що дозволяє створювати додатки, які можуть отримувати доступ і обробляти широкий спектр джерел даних у різноманітних видах.

2.7 Deep Lake

Deep Lake – векторне сховище, що забезпечує зберігання вбудувань та їхніх метаданих. Воно дозволяє здійснювати пошук за цими вбудуваннями та їхніми атрибутами для ефективного пошуку даних. Deep Lake інтегрується з LangChain, що полегшує розробку додатків.

Deep Lake є мультимодальним, що означає, що його можна використовувати для зберігання об'єктів різної модальності, таких як тексти, зображення, аудіо та відео, разом з їх векторними представленнями, а також безсерверним, що означає, що ми можемо створювати хмарні набори даних і керувати ними без створення та управління екземпляром бази даних. Цей аспект значно прискорює роботу над новими проектами.

Векторне сховище Deep Lake було використано, щоб зберігати вбудування (embeddings), що були отримані з бази знань.

2.8 GPT Моделі

Моделі GPT, розроблені OpenAI, компанією завдяки якій ми маємо ChatGPT, є значним досягненням у галузі обробки природної мови та штучного інтелекту. Ці моделі, засновані на архітектурі моделі-трансформера, з перекладу GPT розшифровується як Generative Pre-training Transformer (генераторний трансформер попереднього навчання), що призначені для розуміння і генерування тексту, схожого на людський, шляхом передбачення наступного слова в реченні.

Починаючи з GPT-1, який запровадив концепцію неконтрольованого попереднього навчання з подальшим контрольованим доопрацюванням, еволюція моделей GPT зазнала значних покращень як у масштабах, так і в можливостях. GPT-2 розширив цей підхід, збільшивши розмір моделі та

навчальні дані, продемонструвавши потенціал великомасштабних мовних моделей у створенні зв'язного та контекстуально релевантного тексту.

Найпомітніший стрибок відбувся з GPT-3, яка може похизуватись 175 мільярдами параметрів, що робить її однією з найбільших і найпотужніших мовних моделей на цей день. Здатність GPT-3 виконувати широкий спектр завдань, від перекладу до генерації коду, з мінімальним налаштуванням, демонструє його універсальність і глибину розуміння. Ґрунтуючись на цьому фундаменті, GPT-4 ще більше розширює ці можливості, значно збільшивши довжину контексту до 64 000 слів, що дає змогу вести складніші та більш нюансовані розмови.

Одне з ключових застосувань моделей GPT - розмовний штучний інтелект, де вони чудово підтримують контекст під час тривалої взаємодії, що робить їх ідеальними для чат-ботів і віртуальних асистентів. GPT-4, зокрема, покращує багатомовне розуміння, безпеку та налаштування, що дозволяє йому працювати з різними мовами та надавати точніші, контекстно-залежні відповіді. Крім того, здатність GPT-4 обробляти візуальні дані, хоча вона ще не повністю інтегрована в ChatGPT, натякає на майбутні досягнення в галузі мультимодального ШІ.

Розробка та інтеграція цих моделей має глибокі наслідки для різних сфер, включаючи обслуговування клієнтів, освіту, створення контенту тощо. Використовуючи API ключ, що можна отримати на офіційному сайті OpenAI, розробники можуть інтегрувати GPT-4 в додатки, покращуючи користувацький досвід за рахунок інтелектуальних, контекстно-залежних взаємодій. Оскільки моделі продовжують розвиватися, їхній потенціал революціонізувати взаємодію між людиною і комп'ютером зростає, прокладаючи шлях до більш інтуїтивних та ефективних інструментів комунікації.

2.9 Технології промптингу

Інжиніринг промптів (Prompt engineering) - це практика написання добре структурованих і ретельно продуманих запитів (або промптів), які можуть бути краще інтерпретовані генеративною моделлю штучного інтелекту. Запит вказує LMM, яке завдання виконати і який результат згенерувати. Вона може містити інструкцію, контекст, вхідні дані та вихідні показники. Використовуючи інжиніринг промптів, ми можемо використовувати LMM для виконання різних завдань, від простих відповідей на запитання до складної генерації креативних текстів. Він базується на емерджентній властивості - навчанні в контексті, що дозволяє великим штучним моделям навчатися на основі промптів. Інжиніринг промптів покращує продуктивність Великих мовних моделей у виконанні поставлених завдань та використовує різні типи промптинг-технік. Далі будуть наведені приклади найпопулярніших з них.

2.9.1 Zero-shot Prompting

Zero-shot Prompting техніка промпт інжинірингу базується на тому, що ми надаємо LLM запит, який описує завдання, але не містить жодних прикладів для завдання. Потім LMM має згенерувати відповідь на запит. Це покращує гнучкість та узагальнення LLM. Його можна використовувати для навчання LLM на декількох завданнях, без необхідності накопичення навчальних даних для кожного із завдань. Наприклад, ChatGPT може написати вірш про теорему Фалеса, без надання попередніх прикладів того, як писати вірш. Однак ця техніка обмежена для складних завдань. На рис.

2.6 зображено приклад Zero-shot Prompting, як ChatGPT без будь-якого пренавчання зміг написати вірш.

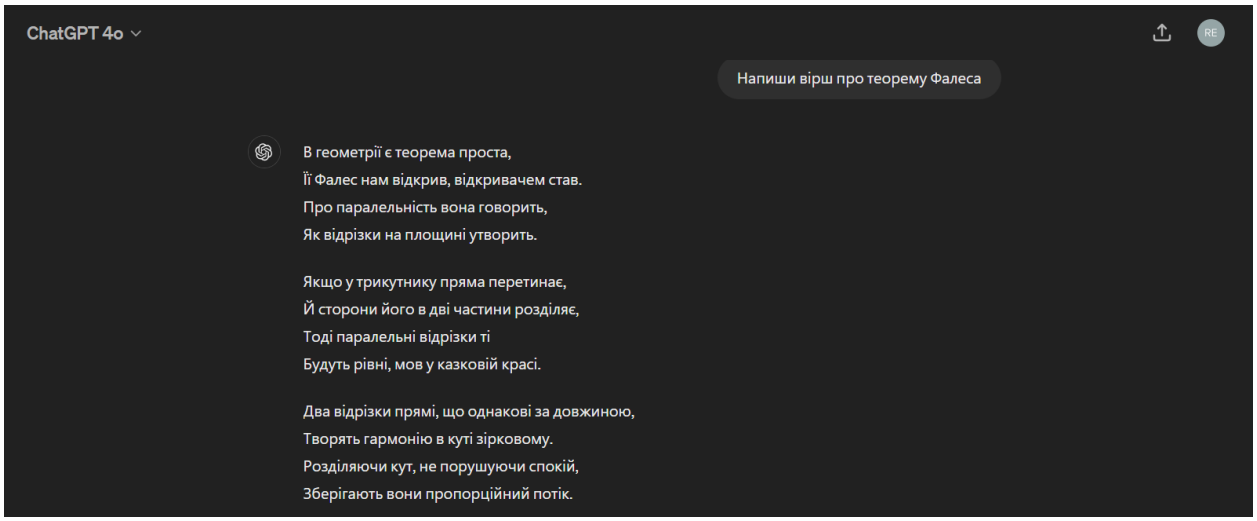


Рисунок 2.6 – Приклад Zero-shot Prompting

2.9.2 Few-shot Prompting

Few-shot Prompting техніка тренує модель на кращу продуктивність. Цей метод промпт інжинірінга полягає в тому, що на додаток до запиту LLM надаються декілька прикладів бажаного результату (рис. 2.6). Приклади допомагають моделі краще зрозуміти завдання і генерувати більш точні та інформативні відповіді. Ми повинні надавати моделі великі та різні приклади, а не кілька схожих прикладів. Це гарантує, що модель дізнається якомога більше про завдання. Цей метод є хорошою технікою для багатьох завдань, але не є надійною для складних завдань на міркування. Тому потрібні більш просунуті методи промптинга, такі як Ланцюжок думок, дерево думок або точне налаштування (fine-tuning) (рис. 2.7).

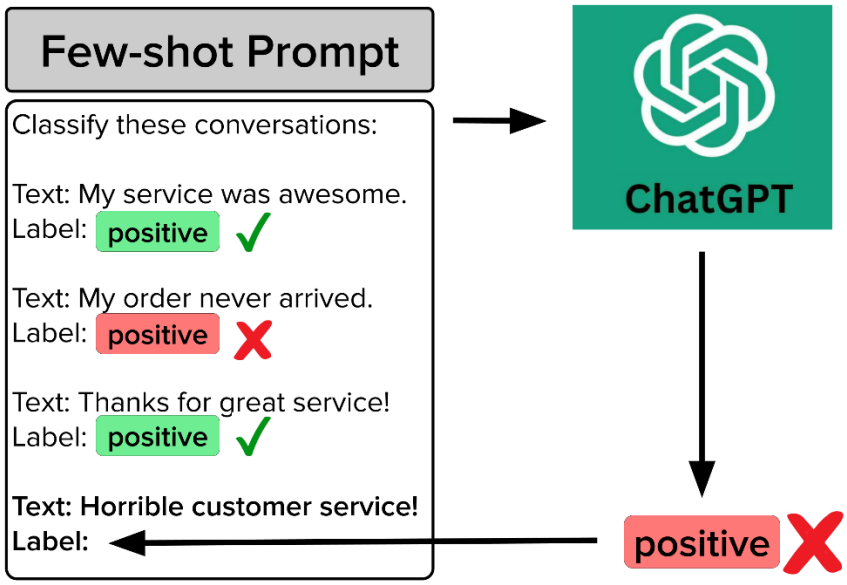


Рисунок 2.7 – Приклад Few-shot Prompting техніки

2.10 Вибір програмних засобів

У цьому розділі будуть розглянуті такі технології як мова програмування, модулі та бібліотеки, які були використані в даній роботі для досягнення результатів отримання ієрархічних структур за допомогою Large Language Models. Дані технології були обрані відповідно до опрацьованих дослідів, що описані у попередніх розділах, з метою отримання бажаних результатів відповідно до тематики даної роботи.

2.10.1 Python

Python – це мова програмування, яка часто обирається завдяки своїм різноманітним характеристикам та можливостям, застосуванню та простоті. Використання мови Python чудово підходить для завдань із машинним навчанням та нейронними мережами, в той час як LLM (Large Language Models) і є великими, комплексними нейронними мережами з різними архітектурами.

Вкажемо головні переваги мови Python для використання з LLM.

1. Незалежність від платформ та операційних систем: скрипти на мові Python не потребують ніяких редагувань для того, щоб бути запущеними на різних операційних систем та у різних середовищах розробки. Також, доступно багато онлайн ресурсів для запуску скриптів в режимі онлайн з потужними графічними картами (наприклад, за допомогою ресурсів Kaggle або Google Colab, можливо використовувати швидкі GPU навіть безкоштовно).
2. Стабільність та простота: код мови Python є дуже чітким та зрозумілим, на відміну від інших мов програмування, що полегшує розуміння коду та процес роботи.
3. Різновид фреймворків та бібліотек: мова Python має дуже великий набір бібліотек та фреймворків, які пропонують надійне середовище та значно зберігає час. Всі основні бібліотеки стабільно оновлюються та мають підтримку, що значно спрощує роботу.

4. Документація: мова Python має дуже багато доступних ресурсів та документацій, що дає можливість легко розібратись з будь-якою функцією, бібліотекою, та іншими об’єктами мови програмування.

Оскільки мова Python є передовою мовою програмування у використані для навчання великих нейронних мереж, вона була обрана для цієї роботи. Python має багато фреймворків для зрозумілого та швидкого навчання Великих мовних моделей, що і є метою даної роботи.

2.10.2 Requests

Requests – це бібліотека мови Python, яка дозволяє відправляти HTTP запити, використовуючи мову Python. Вона дозволяє надсилати HTTP/1.1 запити з такими методами, як GET, POST, PUT, DELETE тощо. Бібліотека автоматично обробляє кодування URL-адрес, параметрів і даних, закодованих у формі. Крім того, запити надають прямий доступ до вмісту відповіді, заголовків і кодів стану, що робить її популярним вибором для веб-скрепінгу.

2.10.3 Gephi

Gephi – це один з провідних засобів для візуалізації та дослідження різного виду графів та мереж. Також, перевагою цього застосунку є те, що він абсолютно безкоштовний та з відкритим вихідним кодом. У застосунку Gephi є багато різного функціоналу, що включає:

1. Exploratory Data Analysis (Дослідницький Аналіз Даних) – аналіз за допомогою мережевих маніпуляцій у реальному часі.

2. Link Analysis (Аналіз зв’язків) – отримання основних структур асоціацій між об’єктами.
3. Social Network Analysis (Аналіз соціальних мереж) – легке створення соціальних зв’язків між даними.
4. Biological Network Analysis (Аналіз біологічних мереж) – відображення характеристик біологічних даних.
5. Poster Creation (Створення постерів) – створення наукових робіт з високоякісними даними.

Оскільки у цій роботі головною метою є створення ієрархічних структур, то з метою їх аналізу, застосунок Gephi буде використовуватись з функцією аналізу зв’язків (link analysis).

2.10.4 Selenium

Selenium – це потужний модуль мови Python, який використовується для контролю веб-браузерів та запровадження автоматизації роботи з браузерами. Він дозволяє програмно керувати браузерами, виконуючи такі дії, як навігація по веб-сторінках, заповнення форм, натискання кнопок, і багато іншого. Selenium підтримує різні браузери, включаючи Chrome, Firefox, Safari та Internet Explorer, забезпечуючи крос-браузерне тестування.

2.10.5 Telethon

Telethon – це ще один асинхронний модуль мови Python, який забезпечує зручне використання з метою взаємодії з API мережі Telegram. Цей модуль дозволяє створювати ботів, автоматизацію завдань, та

взаємодію з каналами Telegram. Перелічимо декілька переваг цього модулю для роботи з мережею Telegram:

1. Асинхронний API – цей модуль використовує модуль asyncio, дозволяючи обробляти декілька завдань паралельно.
2. Управління сесіями – Telethon обробляє управління сесіями, включаючи зберігаючи та завантажуючи сесії.
3. Обробка подій – ця функція запроваджує надійну систему обробки подій, дозволяючи розробникам відповідати до різних видів подій, такі як нові повідомлення, редагування, та дії користувачів.
4. Customization – бібліотека дуже адаптована до нових змін, дозволяючи розширювати функціональність до спеціальних потреб.
5. Повне покриття API – цей модуль повністю покриває API мережі Telegram, дозволяючи мати доступ до всіх можливостей, які доступні та забезпечені мережею Telegram.

2.10.6 Beautiful Soup

Бібліотека BeautifulSoup на мові Python використовується для розбору HTML і XML документів. Вона створює дерево розбору з вихідного коду сторінки, яке використовується для вилучення даних з HTML-тегів, атрибутів і тексту. BeautifulSoup надає методи для навігації, пошуку та модифікації дерева розбору, що робить її надзвичайно корисною для проектів веб-скрепінгу.

2.11 Використані Large Language Models

2.11.1 ChatGPT

В якості моделей, які були натреновані на текстовому корпусі та використані для створення ієрархічних структур, було використано чотири різні Великі мовні моделі (Large Language Models) від OpenAI: GPT 3.5 turbo, GPT-4, GPT-4o, GPT-4-turbo. При цьому, ці моделі не є безкоштовними, і задля використання API від OpenAI треба сплачувати.

В той час, як модель GPT 3.5 turbo має лише 16385 токенів, то моделі GPT-4, GPT-4o, GPT-4-turbo мають 128000 токенів, що можуть значно покращити отримані результати.

Кожна з цих чотирьох версій є модернізацією своїх минулих аналогів, саме тому і було обрано використати їх та порівняти результати між собою.

2.11.2 Llama

Llama – це сімейство Великих мовних моделей, які були розроблені компанією Meta AI. Ці моделі є попередньо навчені (pretrained) та точно налаштованими (fine tuned). Вони є безкоштовними, та доступні на такій платформі як HuggingFace, звідки можна завантажити модель і використовувати.

Моделі сімейства Llama 2 мають від 7 до 70 мільярдів параметрів. Моделі Llama 2 перевершують багато інших open-source моделей по багатьом із класичних задач, як зазначено у досліді зробленому розробниками моделі [24].

2.12 Висновки до розділу 2

На даний момент вже існують класичні підходи для створення ієрархічних структур за допомогою віртуальних асистентів на базі LLM. Відповідно до опису технологій, було розглянуто схожі дослідження про різні підходи, які можуть бути використані задля побудови та створення ієрархічних стратегій з метою порівняння створених структур та аналізу, використовуючи метрики, описані у розділі 2.1.1.

Було детально описано архітектуру Великих мовних моделей, та шари, що використовуються у трансформерах. Всі описані програмні засоби, а саме мова програмування Python та її різні модулі та бібліотеки, будуть використані з метою побудови ієрархічних структур на основі LLM. Обрано наступні модулі та бібліотеки: Requests, Selenium, Telethon, Beautiful Soup. В якості мови реалізації обрано Python, з метою розробки програмного продукту для практичної реалізації, та застосунок Gephi для візуалізації отриманих результатів за допомогою LLM.

Запропоновано представлення основи стратегій у вигляді ієрархічних структур з метою наглядності та структуризації різних характеристик зібраного текстового корпусу (зібрана інформація про дрони). Обрано метрики, що використовуються для оцінювання складності побудованої ієрархічної структури: загальна кількість вузлів ієрархії, загальна кількість співпадаючих вузлів, кількість вузлів на останньому рівні, максимальна глибина вузлів гілок

Було розроблено детальну діаграму, яка покроково описує алгоритм побудови віртуального асистента, та системний підхід для створення ієрархічних стратегій. Спочатку, створюється база знань з онлайн ресурсів, такі як Інтернет сайти та канали мережі Telegram. Після цього, створюється векторне відображення отриманого текстового корпусу. Маючи це, різні LLM моделі навчаються на базі знань та отримують досвід з текстового

корпусу. Після цього, створюються вказівки, за допомогою таких технологій як Chain of Thoughts та Few-Shot Prompting. Завдяки цьому, LLM моделі створюють ієрархічні структури, і за допомогою застосунку Gephi вони візуалізуються.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Для збору бази знань, спершу було використано бібліотеку Telethon, завдяки цьому було зібрано текстовий корпус із телеграм каналів про дрони. Іншим аналогічним підходом при зборі текстових корпусів були бібліотеки Requests та Selenium, які дозволяють зібрати бази знань (knowledge base) за допомогою онлайн ресурсів (сайтів). Після того, як текстовий корпус готовий, обиралась велика мовна модель (LLM), такі як різні версії моделей ChatGPT та моделі сімейства Llama. Коли LLM натренована на текстовому корпусі, а саме модель отримала знання про зібрані дані, наступним етапом є створення ієрархічних структур за допомогою технології Chain of Thoughts (CoT), та технології звичайного промптингу. Також, розглядаються технології Zero Shot Prompting, коли модель не отримує ніякого прикладу, який вона має згенерувати, та Few Shot Prompting, коли моделі надається приклад, як має виглядати кінцевий результат. Коли LLM надала кінцевий результат, то він збирається у CSV файл та обробляється за допомогою застосунку Gephi з метою отримати результати з візуалізацією. Після цього, отримані результати з різними моделями та різними значенням гіперпараметру температури (temperature), вони оцінюються та порівнюються за допомогою різних метрик для ієрархічних структур.

3.1 Парсинг бази знань

3.1.1 Реалізація зчитування даних з Telegram каналу та їх обробки

У рамках цієї роботи для подальшого аналізу було розроблено програмний код на Python, який використовує бібліотеку Telethon для зчитування даних з каналу Telegram. Мета полягала в тому, щоб

автоматизувати процес збору повідомлень з каналу, зберігаючи їх у структурованому форматі для подальшого аналізу.

Цей програмний код призначений для зчитування повідомлень з певного каналу Telegram та збереження їх у CSV файл для подальшої обробки (рис. 3.1).

```
async def main():
    await client.start(phone_number)

    if not await client.is_user_authorized():
        await client.send_code_request(phone_number)
        await client.sign_in(phone_number, input('Enter the code: '))

    dialogs = await client.get_dialogs()
    print(dialogs)

    channel = None
    for dialog in dialogs:
        if dialog.is_channel:
            channel = dialog.entity
            break

    if channel is None:
        print("No channel found")
        return

    with open('drones.csv', 'w', newline='', encoding='utf-8') as csvfile:
        fieldnames = ['id', 'sender_id', 'date', 'message']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()

        async for message in client.iter_messages(channel, reverse=True):
            writer.writerow({
                'id': message.id,
                'sender_id': message.sender_id,
                'date': message.date.strftime('%Y-%m-%d %H:%M:%S'),
                'message': message.text
            })
with client:
    client.loop.run_until_complete(main())
```

Рисунок 3.1 – Лістинг реалізації парсингу телеграм каналів

Завдяки програмному коду можна зібрати базу повідомлень із телеграм каналу, і врешті-решт створити базу знань для тренування обраної великої мовної моделі (рис. 3.2).

id	sender_id	date	message
1	-1001809448528	2023-02-24 22:33:21	
5	-1001809448528	2023-02-24 22:44:59	
6	-1001809448528	2023-02-24 22:48:36	Офіційний канал Victory Drones
7	-1001809448528	2023-02-24 23:21:39	Запустили офіційний канал Victory Drones. Тут буде цікаво - як перемагати русню розумно і технологічно. Поширюйте серед військових, волонтрів та інших друзів лінк на канал: https://t.me/VictoryDrones2023
8	-1001809448528	2023-02-25 10:23:01	Мою конструкцію з виготовлення зовнішньої антени з простого радіо кабелю повторили тисячі військовихіз ЗСУ. Ось вона http://ur0uun.com/motorola.mp4 На початку війни я перебрав усі варіанти антен та знайшов найпростішу для повторення моїми військовими колегами. Ось калькулятор, як зробити антену на інші частоти (наприклад, UHF). https://nomonsuhendar.blogspot.com/2020/12/flower-pot-antenna-calculator.html Антена працює навіть із «білого» телевізійного кабелю з будь-якої хати, тільки треба додати пару витків. Хлопці у глухих лісах біля Білорусі підіймали антену з кабелю лескою на 25 метрів і робили зв'язок зі штабом на 40 км чере

Рисунок 3.2 – Приклад зібраних повідомлень

Як видно на рис. 3.2, зібрані повідомлення надходять разом із часом повідомлення та унікальним кодом користувача. Тим не менше, для бази знань будуть необхідні лише повідомлення.

Загалом, було оброблено 9787 повідомлень з мережі Telegram, які були зібрані, починаючи з 24 лютого 2023 року, та закінчуючи датою 15 травня 2024 року.

3.1.1.1 Опис програмного коду

Для взаємодії з API Telegram використовується бібліотека Telethon. Клієнт Telegram запускається та перевіряється, чи користувач авторизований.

Якщо користувач не авторизований, надсилається запит на отримання коду та виконується вхід. Всі діалоги (чати, групи, канали) отримуються за допомогою методу get_dialogs. Серед діалогів вибирається перший знайдений канал.

Усі повідомлення з вибраного каналу зчитуються та зберігаються у CSV файл у структурованому форматі (ідентифікатор повідомлення, ідентифікатор відправника, дата та текст повідомлення).

3.1.2 Реалізація зчитування даних з сайтів

В рамках цієї роботи парсинг даних з сайтів з інформацією про дрони був виконаний із використанням бібліотек Requests Selenium мови програмування Python.

Спершу, обирається набір посилань на сайті з необхідною інформацією. Далі, за допомогою функції SeleniumLoader створюється об’єкт завантажувача текстового корпусу. Оскільки LLM мають обмежений максимальний розмір підказки (наприклад, ChatGPT3 має обмеження в близько 4 000 токенів), то наразі нам потрібно розробити рішення, яке б враховувало це обмеження. Саме тому текстові дані розбиваються у менші підчастини (chunks), де розмір однієї частини визначено в 1000 токенів (рис. 3.3).

Рисунок 3.3 – Лістинг реалізації парсингу сайтів

Аналогічний підхід використовується і з бібліотекою Requests, яка є гарним та оптимальним варіантом при парсингу статичних сторінок.

При збору інформацію з Інтернет ресурсів, було оброблено 33 сайти, що містять інформацію про дрони.

3.2 Навчання моделі на базі знань

Коли база знань перетворена у векторний вигляд (Embedding Representation), можна перейти до задання запитів (промптів) до моделі, оскільки тепер обрана мовна модель натренована на текстовому корпусі та має інформацію про зібрані дані.

Векторний вигляд кожного слова створюється завдяки методу бібліотеки OpenAI. Після цього, кожне слово відображається у вигляді n-вимірного вектору, де n – це кількість вимірів простору.

База знань була переведена у векторну репрезентацію за допомогою моделі text-embedding-ada-002, яка доступна за допомогою API (Application Programming Interface) ChatGPT (рис. 3.4).

```
embeddings = OpenAIEmbeddings(model="text-embedding-ada-002")

my_activeloop_org_id = "dklepachevskyi"
my_activeloop_dataset_name = "drones_assistant"
dataset_path = f"hub://{my_activeloop_org_id}/{my_activeloop_dataset_name}"
db = DeepLake(dataset_path=dataset_path, embedding_function=embeddings)

db.add_documents(docs)
```

Рисунок 3.4 – Лістинг створення бази знань

Після створення векторної репрезентації бази знань у векторній базі даних DeepLake створюється текстовий корпус із попередньо зібраних даних.

В даному випадку, в якості великої мовної моделі було використано GPT3.5 Turbo Instruct.

Наприклад, на запит (prompt) «Яким дрономи найчастіше проводяться атаки?» модель дала наступну відповіді, де можна знайти такий дрон як “Shahed-131/136” (рис. 3.5).

```
query = "Яким дроном найчастіше проводяться атаки?"
docs = db.similarity_search(query)
print(docs[0].page_content)
```

Рисунок 3.5 – Приклад промпту

У відповіді на запит, модель навела текст, в якому описано тип дрону (рис. 3.6).

```
        Уночі Сили ППО збили 20 російських "шахедів"

Уночі росіяни атакували Україну 20 ударними безпілотниками типу "Shahed-131/136" і
```

Рисунок 3.6 – Відповідь моделі на запит

Як результат, можемо помітити, що модель успішно навчилась на текстовому корпусі, та забезпечила необхідну інформацію на запит. Хоча інформація і не є максимально точною, проте за допомогою технік промптингу можна цього уникнути і отримати бажаний результат, що буде описано в наступному розділі.

3.3 Тестування запитів до LLM

3.3.1 Моделі GPT

Оскільки тепер модель натренована на текстовому корпусі, то можна приступати до відправлення запитів (промптів) до моделі. За допомогою платного API від ChatGPT є можливість використати будь-які Великі мовні моделі від OpenAI. Для тестування запитів та отримання результатів були обрані наступні моделі від OpenAI: ChatGPT 3.5 turbo, ChatGPT 4.0, ChatGPT 4o, ChatGPT 4.0 turbo.

Для того, щоб модель давала більш чіткі результати, був використаний PromptTemplate, що являє собою шаблон для кожного запиту до моделі (рис. 3.7).

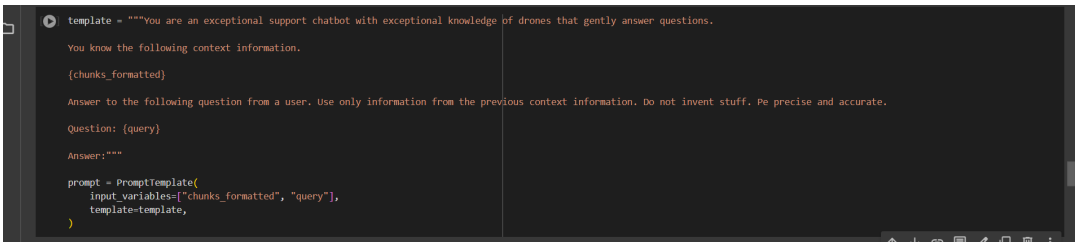


Рисунок 3.7 – Лістинг створення шаблону для запитів

Оскільки наша база знань порівняно невелика, для покращення роботи віртуального асистента та підвищення релевантності відповідей була використана технологія Few-shot prompting. Це забезпечує навчання в контексті, що спрямовує модель на кращу роботу. Така "демонстрація" для навчання роботи віртуального асистента слугує основою для наступних прикладів, де ми хочемо, щоб модель генерувала доречні відповіді.

У цьому підході також була впроваджена технологія промптингу – Ланцюжок думок (Chain of Thoughts) для покращення точності відповідей на запити, пов'язані з дронами. Використані приклади допомагають моделі краще зрозуміти, як формувати проміжні питання та відповіді (рис. 3.8). Приклади були додані в шаблон промпта.

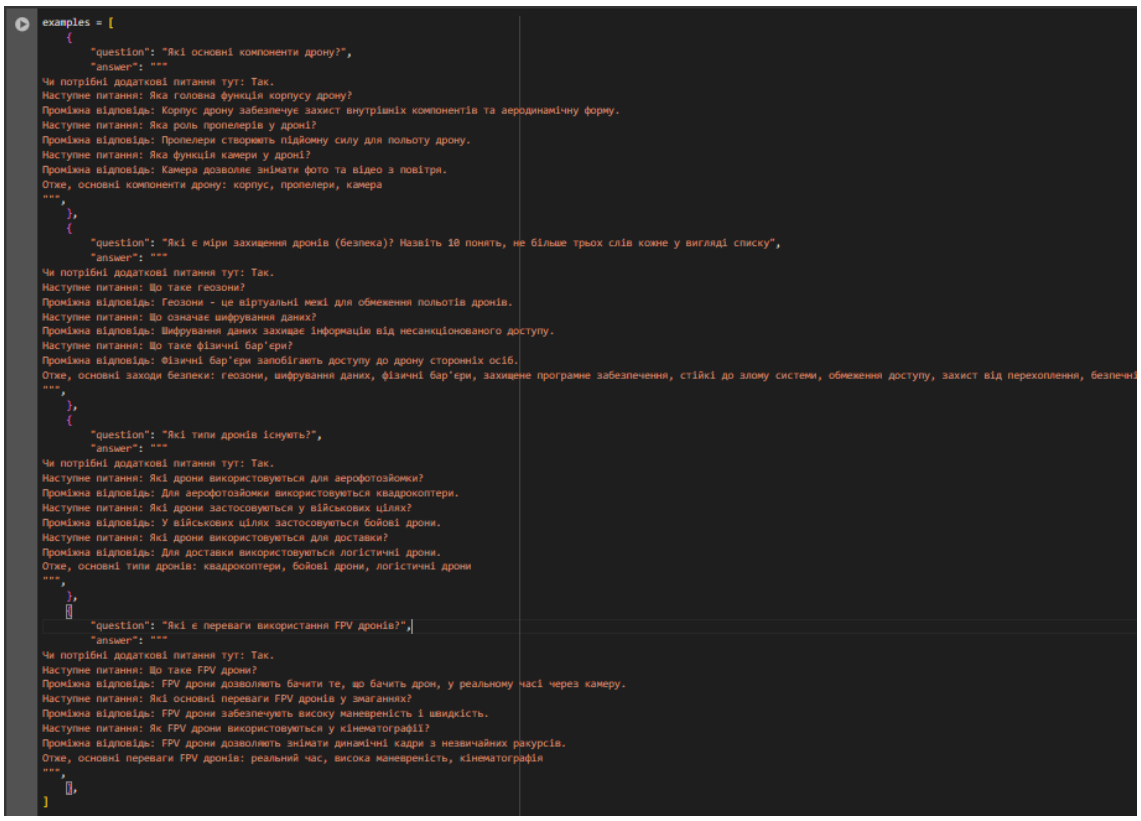


Рисунок 3.8 – Приклади Few-shot prompting та Ланцюжка думок

Після цього, коли шаблон для запиту створено, передається запит до моделі. Модель та гіперпараметр температури моделі передається у параметри функції ChatOpenAI. За допомогою методу run відбувається запуск моделі та отримується результат (рис. 3.9).

Рисунок 3.9 – Відповідь моделі на запит без додавання контексту окрім бази знань, zero-shot prompting

Також, нижче наведено приклад відповіді при використання Few-shot prompting та навчанню промптингу за допомогою Ланцюжка думок (рис.

3.10). Добре видно, що модель робить висновки на основі покрокових роздумів.

```
query = "Чому дрони є важливою зброєю під час війни?"

docs = db.similarity_search(query)
retrieved_chunks = [doc.page_content for doc in docs]
chunks_formatted = "\n\n".join(retrieved_chunks)

chat = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)
chain = LLMChain(llm=chat, prompt=few_shot_prompt_template)

answer = chain.run({"chunks_formatted": chunks_formatted, "query": query})
print(answer)
```

```
Дрони є важливою зброєю під час війни, оскільки:

1. Забезпечують розвідку в реальному часі.
2. Допомагають корегувати артилерійський вогонь точно на відстані.
3. Використовуються для контрбатарейної боротьби.
4. Дозволяють контролювати окремі ділянки фронту.
5. Використовуються для скидання гранат та ВОГів на ворожі позиції.

Отже, дрони забезпечують перевагу на фронті завдяки своїм можливостям у розвідці та бойових діях.
```

Рисунок 3.10 – Відповідь моделі на запит з використанням Ланцюжка думок

Усі промпти базувались на 5 описових характеристиках дронів: які типи дронів бувають, які характеристики дронів, міри безпеки запроваджені в дрони, технічне обслуговування дронів, та використання дронів.

3.3.2 Модель Llama 2

Також, окрім моделей сімейства GPT, було протестовану модель Llama 2 від Meta AI, яка має 7 міліардів параметрів. Загальний алгоритм роботи лишається таким самим, окрім створення об’єкту моделі (рис. 3.11).

```
from transformers import AutoTokenizer
import transformers
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer, Trainer, TrainingArguments


model_name = "meta-llama/Llama-2-7b-chat-hf"
tokenizer = AutoTokenizer.from_pretrained(model_name, use_auth_token=True)
model = AutoModelForCausalLM.from_pretrained(model_name, use_auth_token=True)
```

Рисунок 3.11 – Створення об’єкту моделі Llama2

Аналогічним чином з моделми GPT, створюється текстовий корпус і модель тренується на отриманій базі знань. Далі, була створена зручная

функція llama_response, яка отримує на вхід запит до моделі, та значення параметру температури, а на вихід дає згенерований результат (рис. 3.12).

```
def get_llama_response(prompt: str, temperature=0.0) -> None:
    """
    Generate a response from the Llama model.

    Parameters:
        prompt (str): The user's input/question for the model.

    Returns:
        None: Prints the model's response.
    """
    sequences = llama_pipeline(
        prompt,
        do_sample=True,
        top_k=10,
        num_return_sequences=1,
        eos_token_id=tokenizer.eos_token_id,
        max_length=256,
        temperature=temperature
    )
    print("Chatbot:", sequences[0]['generated_text'])

llama_pipeline = pipeline(
    "text-generation",  # LLM task
    model=model,
    tokenizer=tokenizer,
    torch_dtype=torch.float16,
    device_map="auto",
)
```

Рисунок 3.12 – Створення функцій для обробки запиту моделлю Llama2

3.4 Створення ієрархічної структури та візуалізація

За допомогою застосунку Gephi можна створити візуалізації з метою відобразити отримані за допомогою моделі результати в зрозумілому вигляді. Коли результати з LLM отримані, вони переводяться у CSV (Comma Separated Values) файли та завантажуються у застосунок Gephi.

З метою порівняння моделей, було використано три різні значення для параметру температури – 0.0, 0.3, 0.7, щоб охопити різні крайні ситуації параметру «креативності» моделі.

Завантаживши файли, які зазначають об’єкти кожної структури та взаємодії, можна перейти до візуалізації отриманих результатів (рис. 3.13).

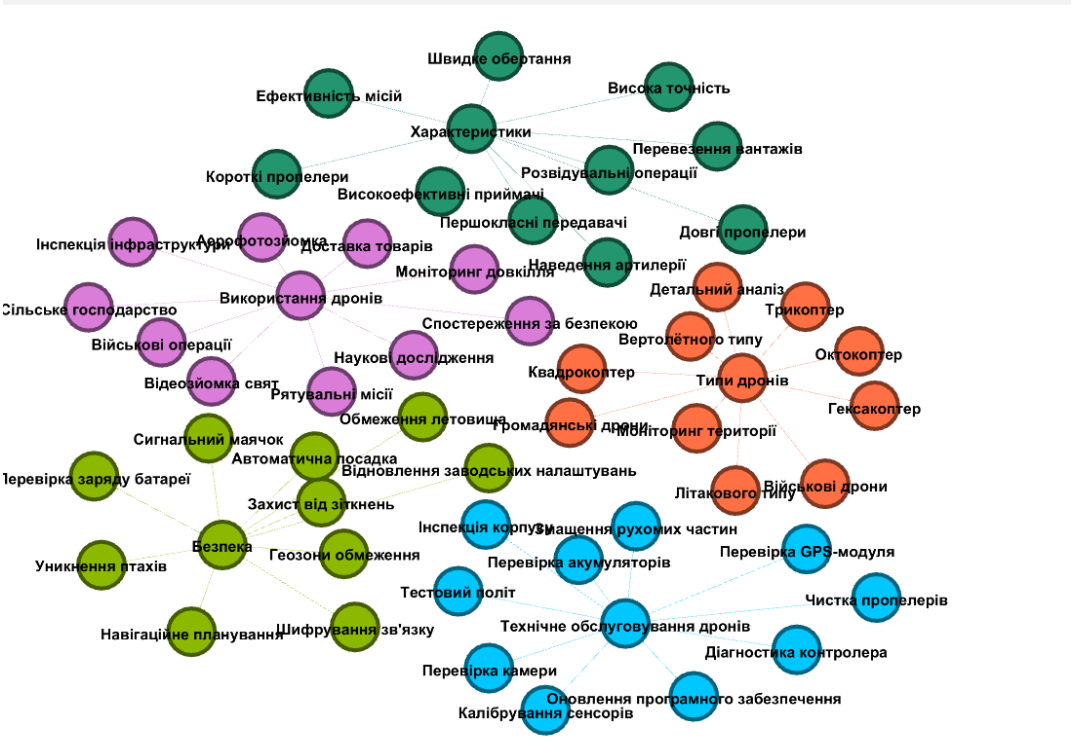


Рисунок 3.13 – Структура отримана з GPT 4 turbo та температурою 0.7

Були створені наступні п’ять описових характеристик дронів: використання, характеристики, типи, технічне обслуговування, та безпека дронів. Кожна з п’яти структур отримала свої згенеровані результати моделю. За допомогою метрик, описаних у другому розділі, можна розрахувати їх та отримати представлення о комплексності моделі.

При наборі таких самих промптів з тією ж моделю GPT 4 turbo, але з параметром температури, що дорівнює 0, результати вже значно відрізняються (рис. 3.14). Більш детальне порівняння наведене у розділі 3.5.

Рисунок 3.14 – Структура отримана з GPT 4 turbo та температурою 0.0

Оскільки при температурі 0.7 модель стає більш креативною, а при температурі 0.0 модель не має креативності, то варто спробувати і проміжне значення – наприклад, температуру зі значенням 0.3, лишивши модель тою самою (рис. 3.15).

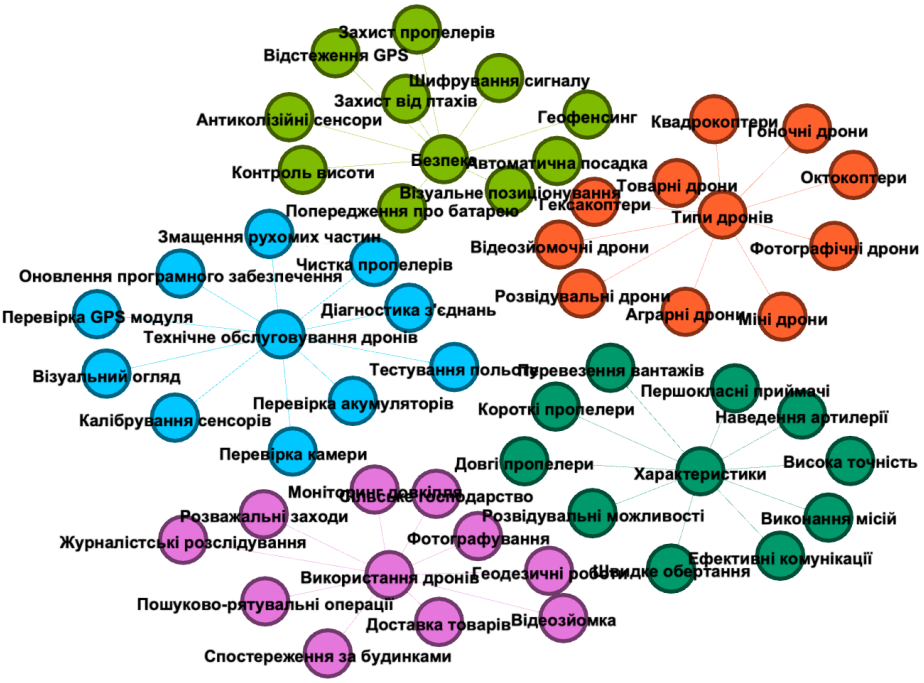


Рисунок 3.15 – Структура отримана з GPT 4 turbo та температурою 0.3

Натренувавши й інші модель, та спробувавши набір із трьох різних значень для параметру температури, дасть нам можливість порівняти моделі у сенсі ієрархічних структур та відповідних метрик. При створенні структури з моделлю GPT-3.5 та параметром температури 0.0, було отримано 100 різних вузлів (рис. 3.16)

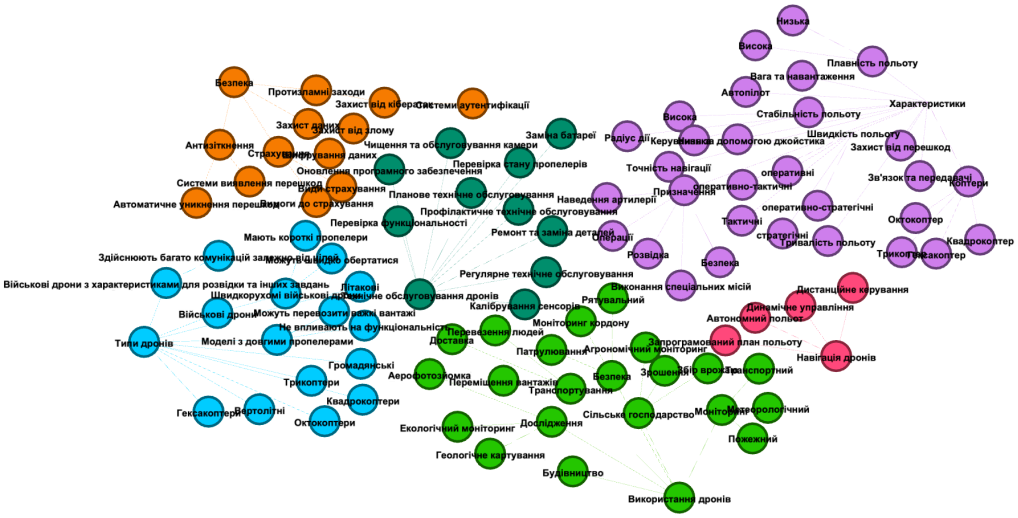


Рисунок 3.16 – Структура отримана з GPT 3.5 та температурою 0.0

Проте, збільшивши температуру до значення 0.3, кількість згенерованих вузлів зменшується до 78, проте і креативність результатів моделі стає вища (рис. 3.17).

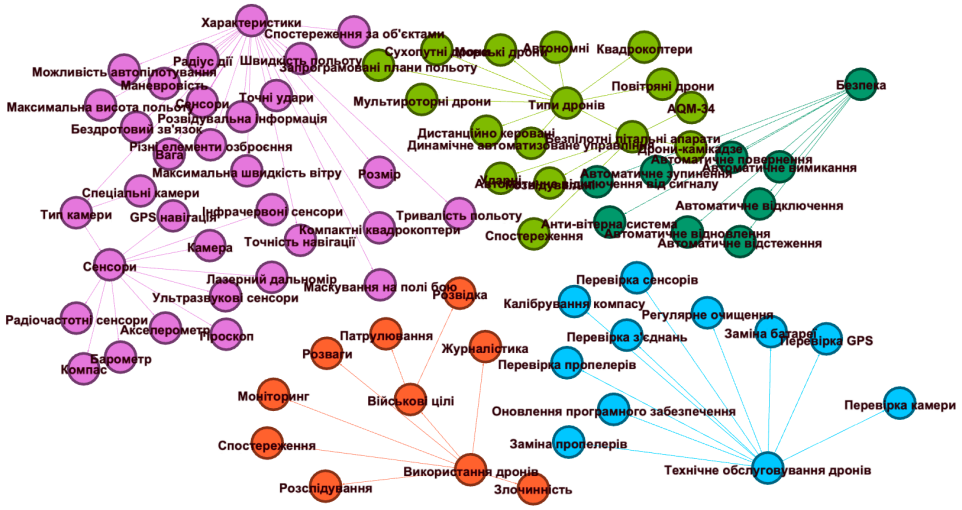


Рисунок 3.17 – Структура отримана з GPT 3.5 та температурою 0.3

Якщо збільшити температури ще, до значення 0.7, то можна помітити, що результати для такої характеристики як типи дронів стають ще значно більш креативними (рис. 3.18).

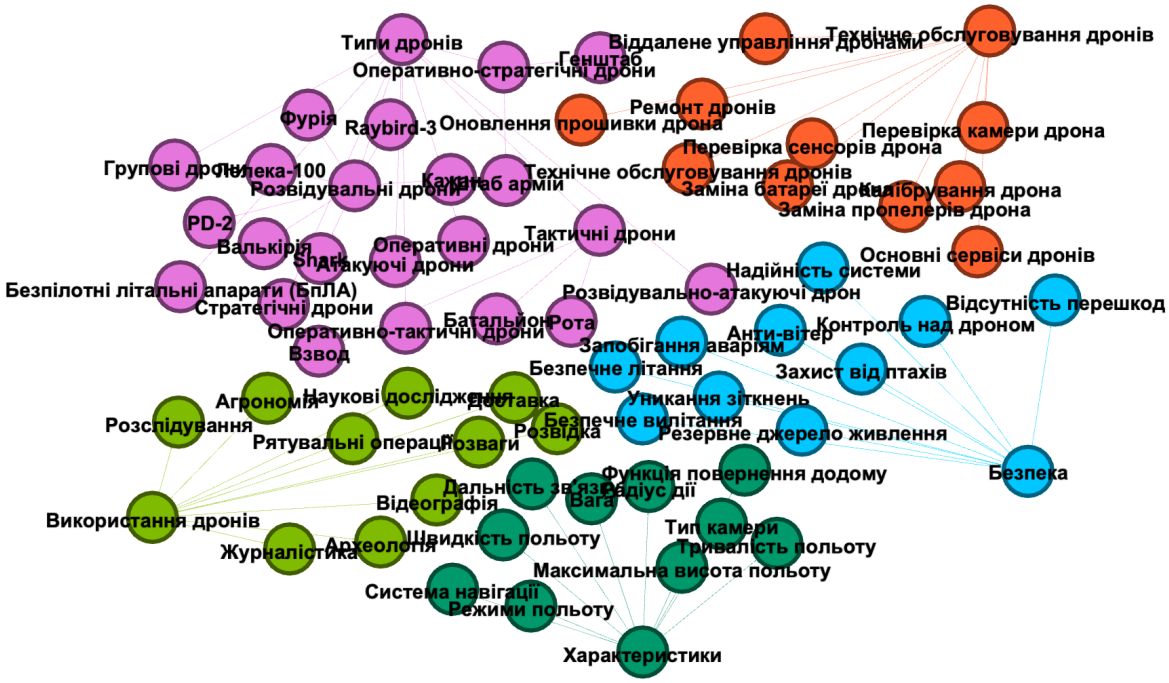


Рисунок 3.18 – Структура отримана з GPT 3.5 та температурою 0.7

Після переходу до моделі GPT 4.0, результати змінились, та загальна кількість вузлів зменшилась, проте стала більш якісною (рис. 3.19).

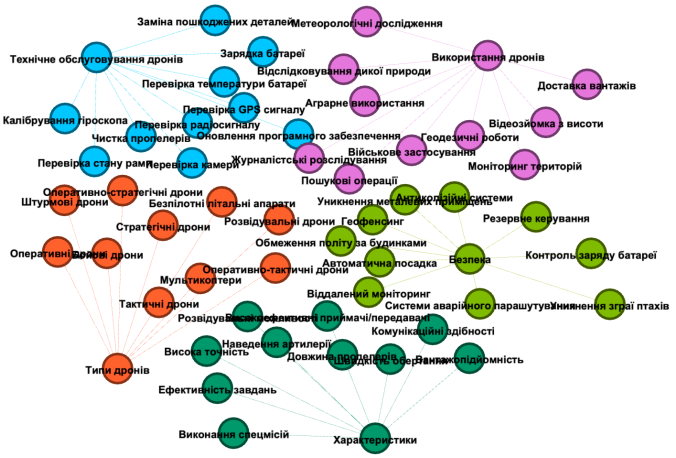


Рисунок 3.19 – Структура отримана з GPT 4.0 та температурою 0.0

При зміні параметру температури до значення 0.3, модель генерує приблизно таку саму кількість результатів, проте креативність моделі помітна у згенерованих Типах дронів (рис. 3.20).

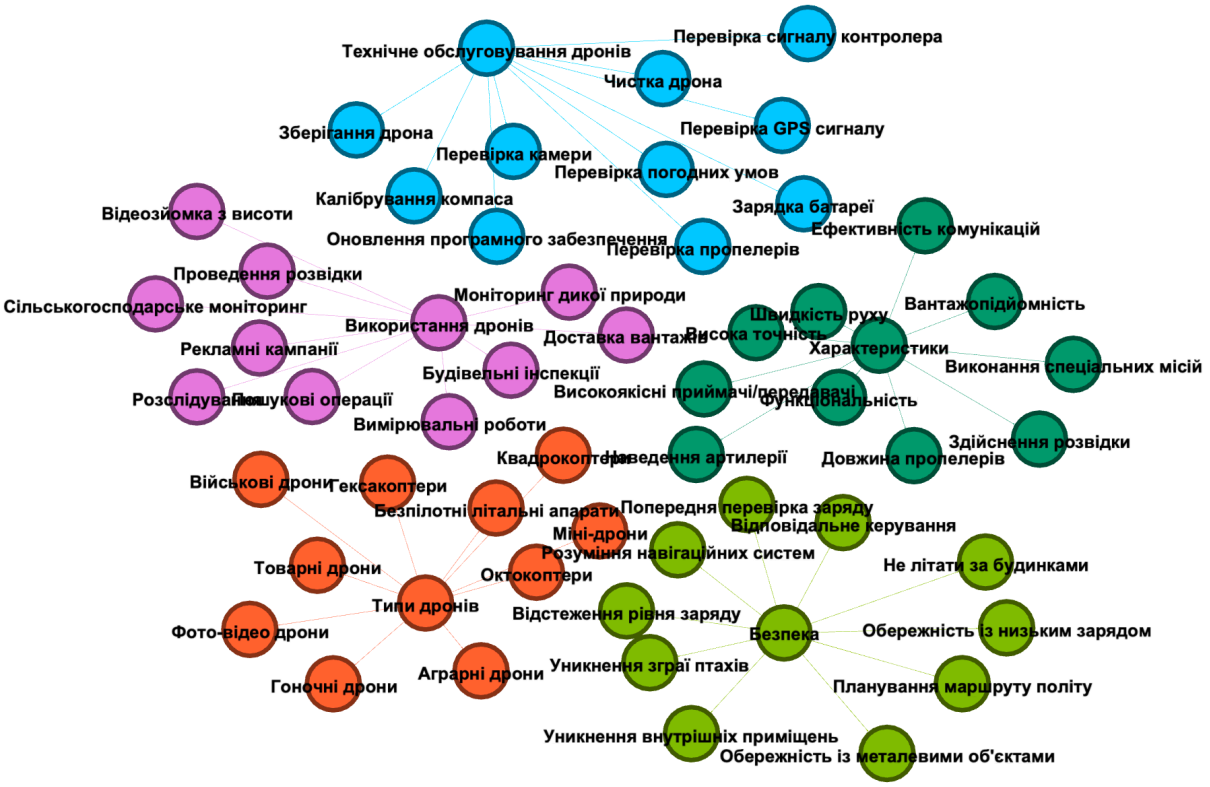


Рисунок 3.20 – Структура отримана з GPT 4.0 та температурою 0.3

При зміні температури до значення 0.7 з тією ж моделлю GPT 4.0, модель впроваджує трохи більш креативні згенеровані відповіді (рис. 3.21).

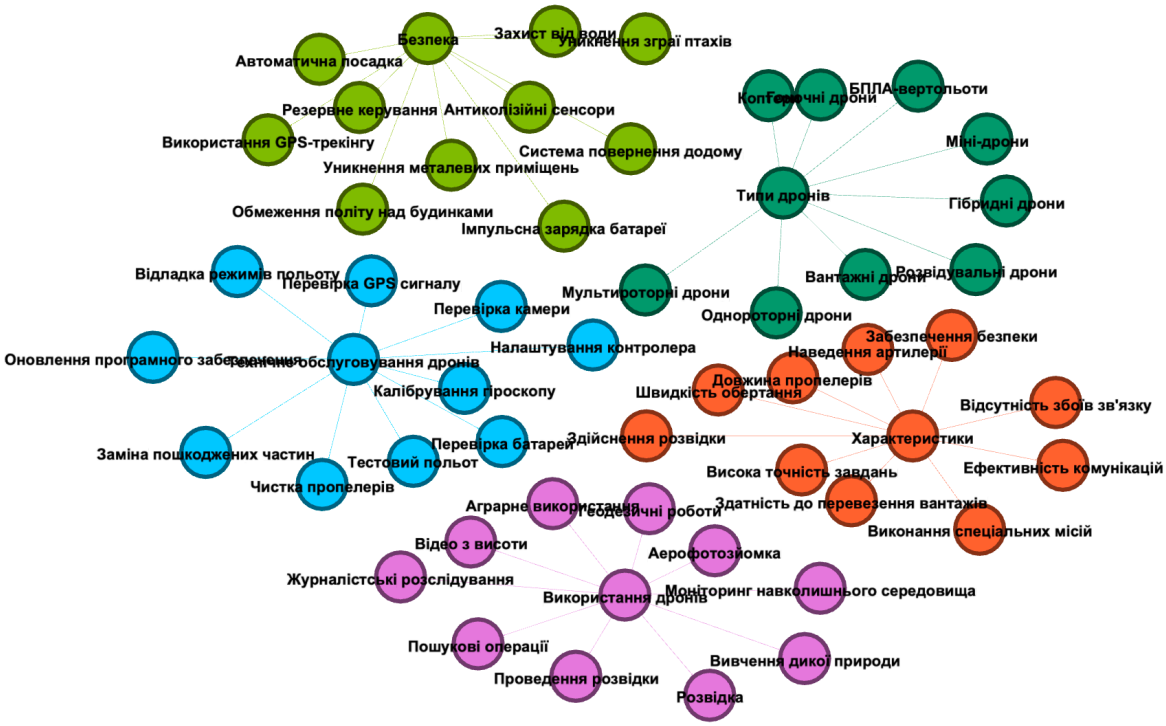


Рисунок 3.21 – Структура отримана з GPT 4.0 та температурою 0.7

Остання з моделей сімейства GPT, яка була протестована, GPT 4o, є найбільш новою, оскільки її перший випуск відбувся 13 травня 2024 року. Тому, оскільки модель є дуже свіжою, то й очікувані результати мають бути більш якісними (рис. 3.22).

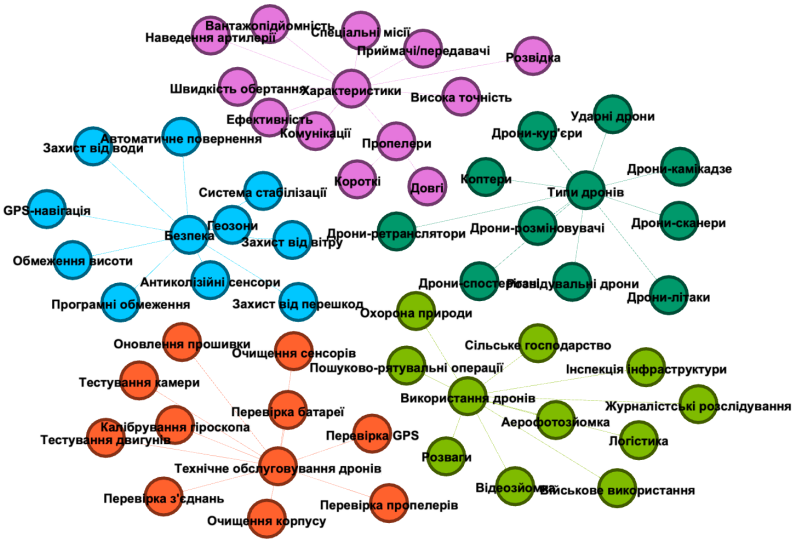


Рисунок 3.22 – Структура отримана з GPT 4o та температурою 0.0

Кількість згенерованих характеристик виявилась близькою до попередньої моделі. Збільшивши температуру моделі до 0.3, результати не сильно змінились, і навіть кількість згенерованих результатів лишилась майже такою самою (рис. 3.23).

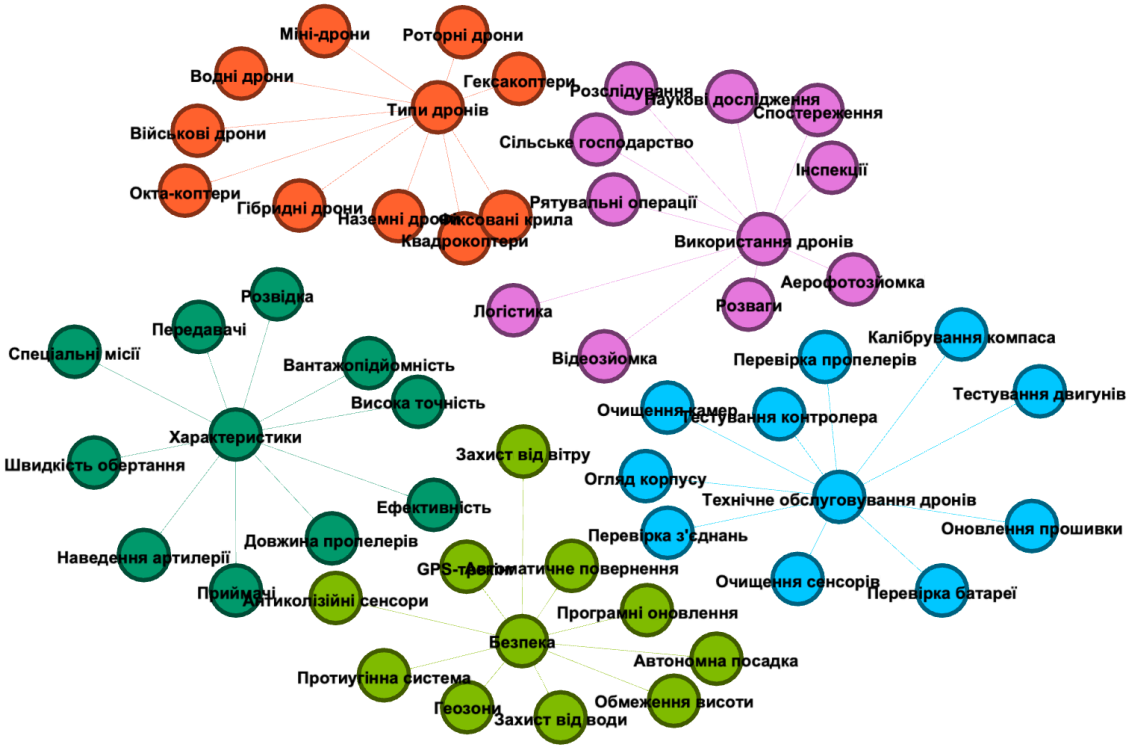


Рисунок 3.23 – Структура отримана з GPT 4o та температурою 0.3

Останній експеримент з цією моделлю був із збільшенням температури до значення 0.7 (рис. 3.24).

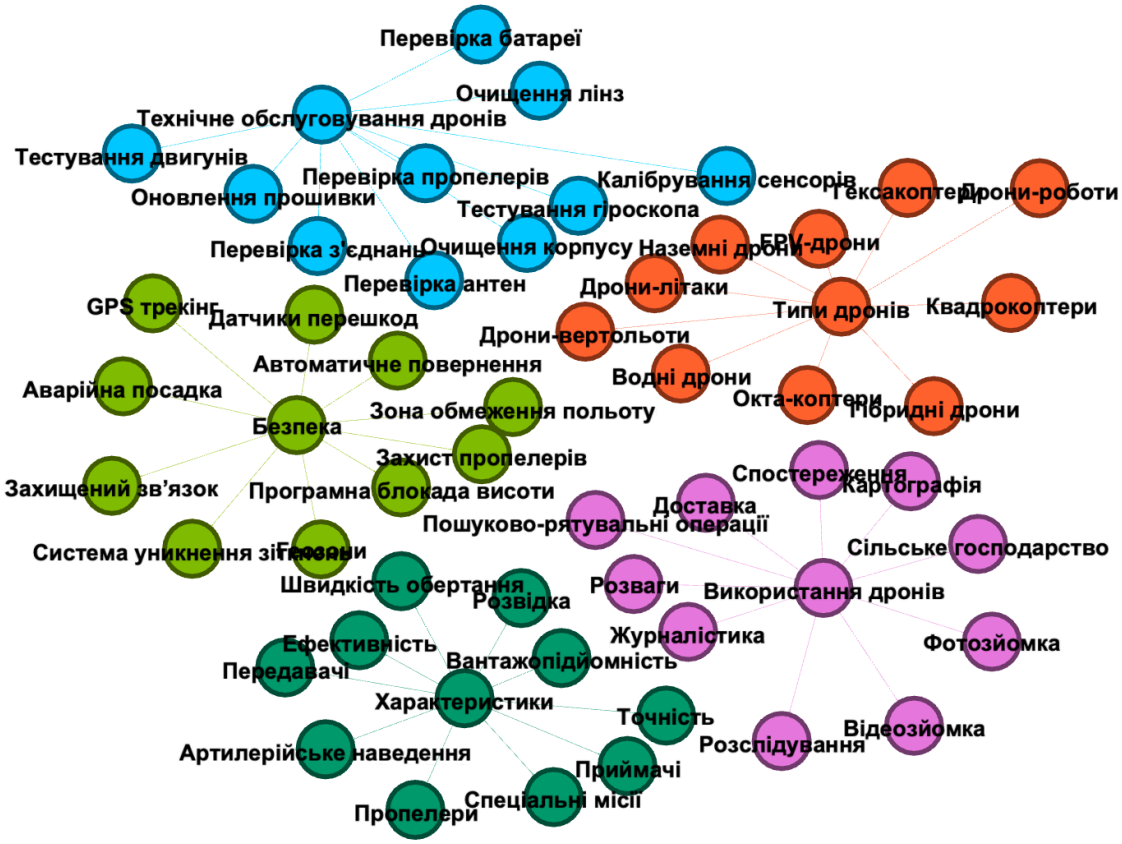


Рисунок 3.24 – Структура отримана з GPT 4o та температурою 0.7

Перейшовши від моделей сімейства GPT, також було проведено експерименти з моделю Llama 2 від MetaAI. З температурою рівною 0.0, модель згенерувала більше результатів (рис. 3.25).

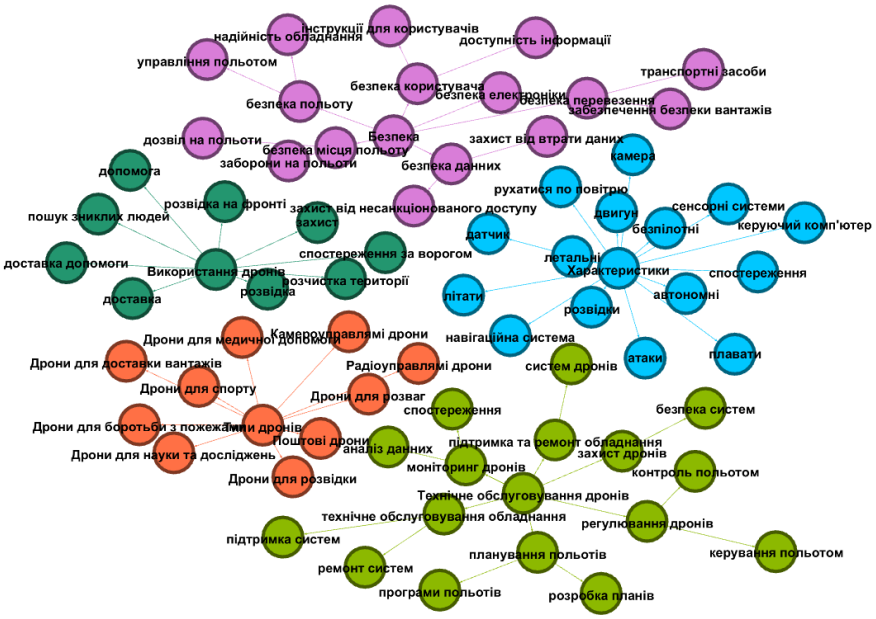


Рисунок 3.25 – Структура отримана з Llama2 та температурою 0.0

Навіть при збільшенні температури до 0.3, модель продовжує генерувати коректні результати (рис. 3.26).

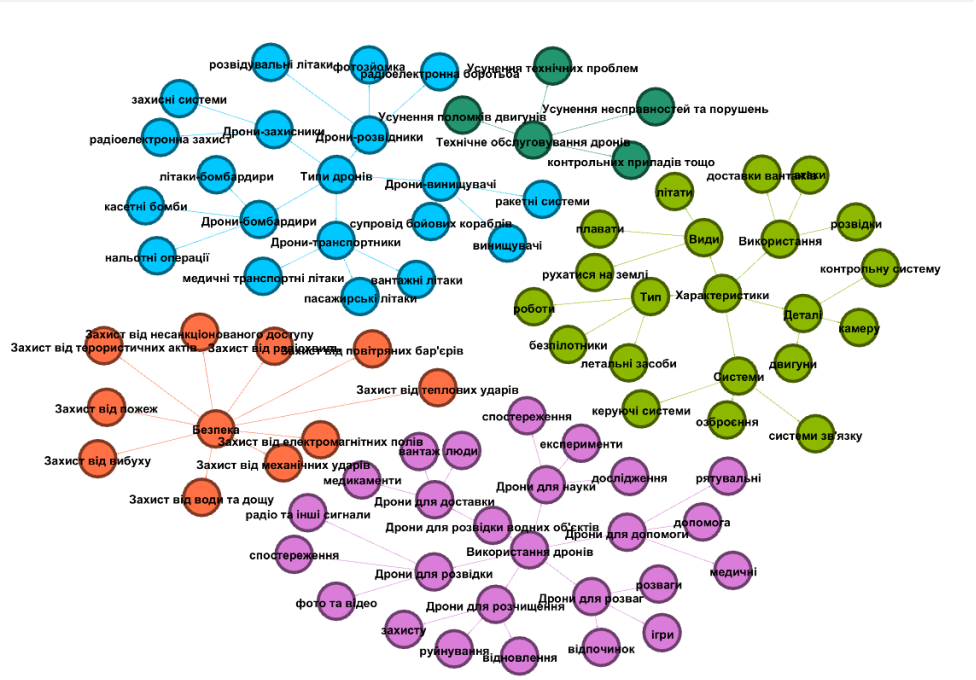


Рисунок 3.26 – Структура отримана з Llama2 та температурою 0.3

При збільшенні температури до значення 0.7, модель згенерувала більш глибоку ієрархічну структуру (рис. 3.27).

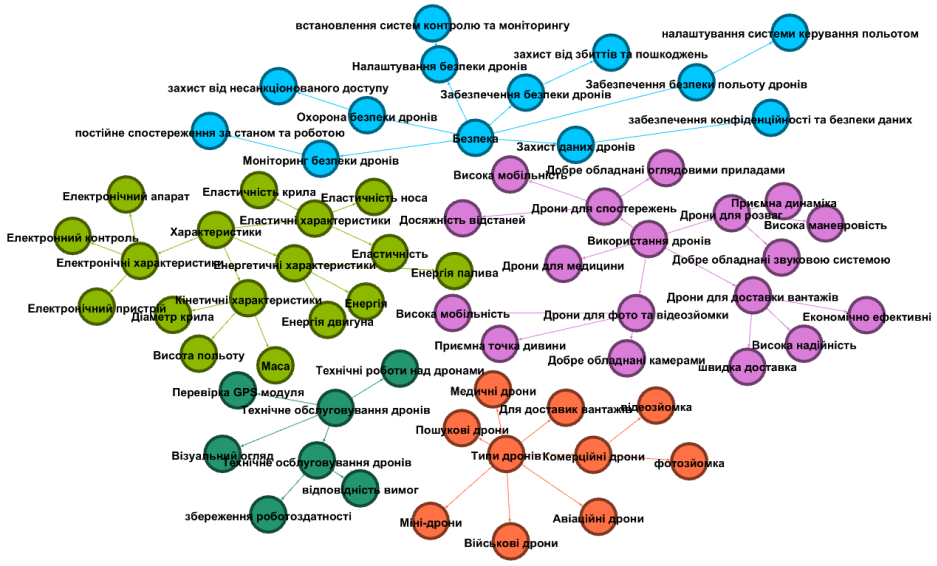


Рисунок 3.27 – Структура отримана з Llama2 та температурою 0.7

3.5 Аналіз та порівняння моделей

За допомогою метрик, що описані у розділі 2.1.1, можна оцінити отримані ієрархічні структури, щоб зрозуміти комплексність та складність різних використаних моделей (ChatGPT, Llama).

За допомогою формул, що описані у розділі 2.1.1, були обраховані результати метрик для різних моделей і внесені у результуючу таблицю (табл. 3.1).

Таблиця 3.1 – Результати метрик різних LLM

Модель	Температура	Загальна к-ть вузлів (nodes)	К-ть вузлів на остан-ньому рівні	Глибина вузлів
GPT-3.5	0	100	42	3
GPT-3.5	0.3	78	73	2
GPT-3.5	0.7	67	62	2
GPT 4.0	0	55	50	2
GPT 4.0	0.3	55	50	2
GPT 4.0	0.7	54	49	2
GPT 4o	0	57	52	2
GPT 4o	0.3	55	50	2
GPT 4o	0.7	53	48	2
GPT 4.0 turbo	0	53	48	2
GPT 4.0 turbo	0.3	57	52	2
GPT 4.0 turbo	0.7	61	56	2
Llama 2	0	71	56	3
Llama 2	0.3	83	62	3
Llama 2	0.7	65	46	3

З метою порівняти моделі на схожість та подібність результатів між собою, за допомогою метрики, описаної у Розділі 2.1.1, було перевірено скільки є співпадінь вузлів між усіма моделями (табл. 3.2).

Таблиця 3.2 – Схожість моделей відповідно до перетину відповідей

Модель	4o_0	4o_0.3	40_0.7	4turbo_0	4turbo_0.3	4turbo_0.7	4_0	4_0.3	4_0.7	3.5_0	3.5_0.3	3.5_0.7	llama_0	llama_0.3	llama_0.7
4o_0	58	30	25	14	14	11	13	10	15	11	12	9	6	6	6
4o_0.3	30	58	31	12	13	12	11	17	15	12	15	12	6	6	8
40_0.7	25	31	56	12	13	8	8	11	9	11	14	11	6	6	6
4turbo_0	14	12	12	57	36	23	16	10	13	9	7	13	6	6	7
4turbo_0.3	14	13	13	36	58	22	17	16	16	12	8	8	6	6	8
4turbo_0.7	11	12	8	23	22	56	12	11	12	15	7	7	6	6	7
4_0	13	11	8	16	17	12	56	19	22	7	8	12	6	6	6
4_0.3	10	17	11	10	16	11	19	56	19	12	11	7	6	6	8
4_0.7	15	15	9	13	16	12	22	19	56	11	10	8	6	6	7
3.5_0	11	12	11	9	12	15	7	12	11	96	15	10	5	5	6
3.5_0.3	12	15	14	7	8	7	8	11	10	15	98	19	5	5	5
3.5_0.7	9	12	11	13	8	7	12	7	8	10	19	78	6	6	6
llama_0	6	6	6	6	6	6	6	6	6	5	5	6	71	14	9
llama_0.3	6	6	6	6	6	6	6	6	6	5	5	6	14	83	8
llama_0.7	6	8	6	7	8	7	6	8	7	6	5	6	9	8	65

Можемо побачити, що моделі однієї LLM, але з різними температурами, все одно мають найбільшу кількість співпадінь. Також, співпадінь між моделями сімейства GPT також більше, аніж кількість співпадінь між моделями Llama 2 та моделями GPT.

3.6 Висновки до розділу 3

Технології та програмні засоби, описані та досліджені в розділах 1 та 2, були використані в цьому розділі з метою побудови ієрархічних структур за допомогою різних LLM моделей, аби порівняти їх між собою. Проведено практичну реалізацію системного підходу щодо реалізації ієрархічних стратегій за допомогою LLM, які дозволили реалізувати ієрархічні

структури. Було досліджено параметр температури моделі, який відповідає за «креативність» моделі під час генерування відповідей на запити.

Спочатку, було зібрано текстовий корпус з різних каналів мережі Telegram, а також з різних ресурсів інтернету та сайтів. Після цього, моделі були натреновані на зібраних даних, щоб моделі могли навчитись на зібраних даних предметної області «Дрони. Технології дронів. Використання дронів в умовах війни». Загалом було оброблено 9787 повідомлені з каналі мережі Telegram, та 33 сайти з Інтернет ресурсів, що являє собою велику комплексну базу інформації, яка була оброблена за допомогою різних LLM моделей.

Було виявлено, що більш комплексна модель, така як Llama 2, генерує більше відповідей на такі самі запити, які отримували й інші моделі. При цьому, модель Llama 2 завжди генерувала три ієрархічні рівні, в той час як більшість з моделей сімейства GPT зазвичай генерували лише два ієрархічні рівні. З таблиці 3.1 видно, що загальна кількість вузлів для моделі Llama 2 була 65, в той час як моделі сімейства GPT в основному мали менше ніж 60 вузлів. У таблиці 3.2, яка відображає схожість моделей між собою, добре видно, що моделі сімейства GPT мають більше спільних вузлів, в той час як перетин між моделями GPT та Llama 2 є мінімальним.

Як і очікувалось, при збільшенні значення параметру температури, креативність моделі збільшувалась, і як наслідок, модель могла генерувати відповіді, які навіть не зустрічались у зібраному текстовому корпусі.

4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на дослідженні ієрархічних структур, створених за допомогою Великих мовних моделей (LLM).

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, 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 часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

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 характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір самої програми.

F_2 – якісний аналіз даних.

F_3 – графічні показники.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Python

б) C++

Функція F_2.

а) Застосування вбудованих функцій.

б) Створення своїх обчислень значень.

Функція F_3:

а) Використання шаблонних графіків.

б) Створення своїх.

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

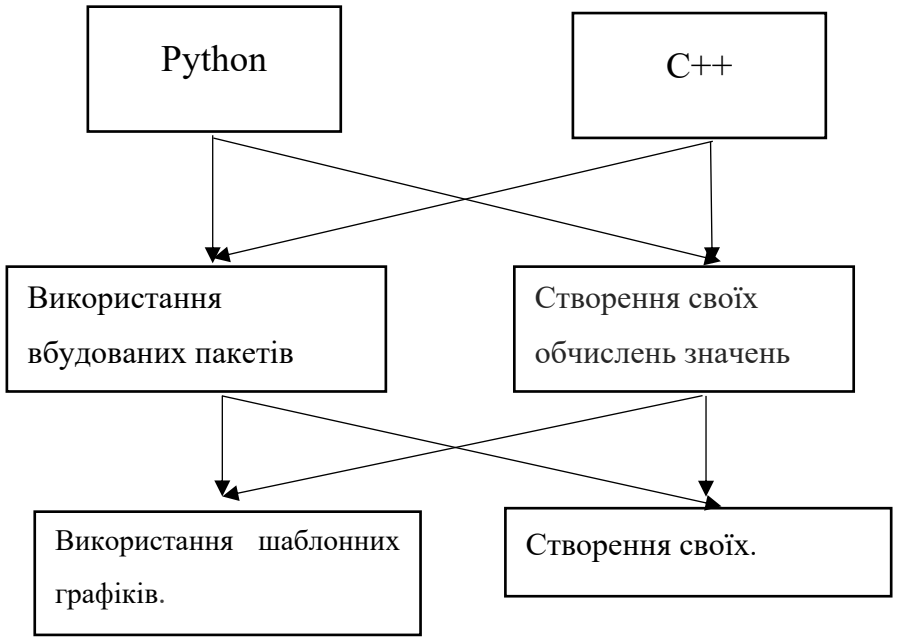


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантиів 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 програма, доступність багатьох бібліотек	Швидкість обробки великих масивів даних
	Б	Висока швидкість обробки великих даних	На написання коду необхідно мати базові навички та вміння
F_2	*А*	Доступність та легкість при написанні	Іноді не відповідає задачі яку треба розв’язати
	Б	Ідеально описують усі необхідні характеристики	Достатньо затратно реалізовувати свої алгоритми для подальшої реалізації
F_3	*А*	Загально прийнята реалізація	Обмеженні можливості використання
	Б	При виконанні власних досліджень краще може передавати висновки	Необхідно достатньо багато часу для написання програми для побудови та знаходження всього необхідного в задачі.

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 загальнодоступності. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F_2:

Програма допускає обрання обох варіантів. Можливо використати варіанти А чи Б.

Функція F_3:

Реалізація першого варіанту є сприйнятливою для програми. Це варіант А.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

$$F_1\text{а} - F_2\text{а} - F_3\text{а}$$
$$F_1\text{а} - F_2\text{б} - F_3\text{а}$$

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:

- *X1* – швидкодія мови програмування;
- *X2* – об’єм пам’яті для обчислень та збереження даних;
- *X3* – час навчання даних;
- *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	60	80	110
Об’єм пам’яті	X2	Мб	60	50	30
Час попередньої обробки даних	X3	мс	80	70	60
Потенційний об’єм програмного коду	X4	кількість рядків коду	35	25	20

За даними таблиці 4.3 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

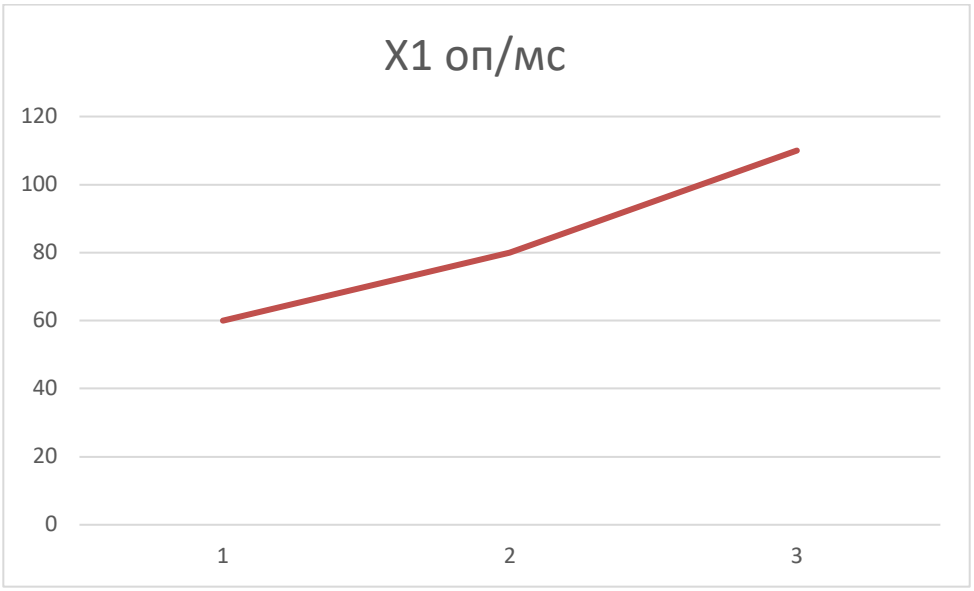


Рисунок 4.2 – X1, швидкодія мови програмування

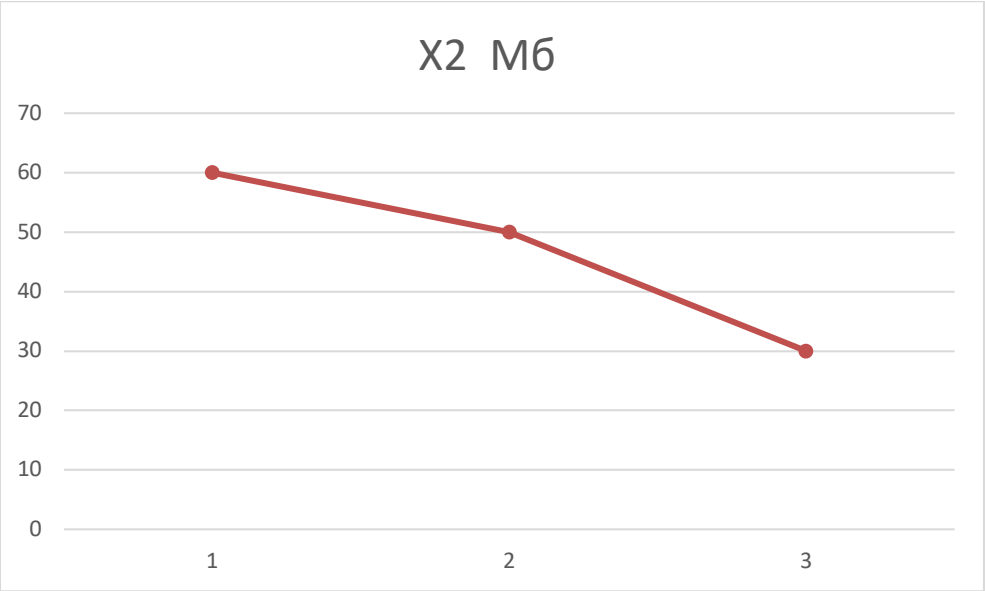


Рисунок 4.3 – X2, об’єм пам’яті

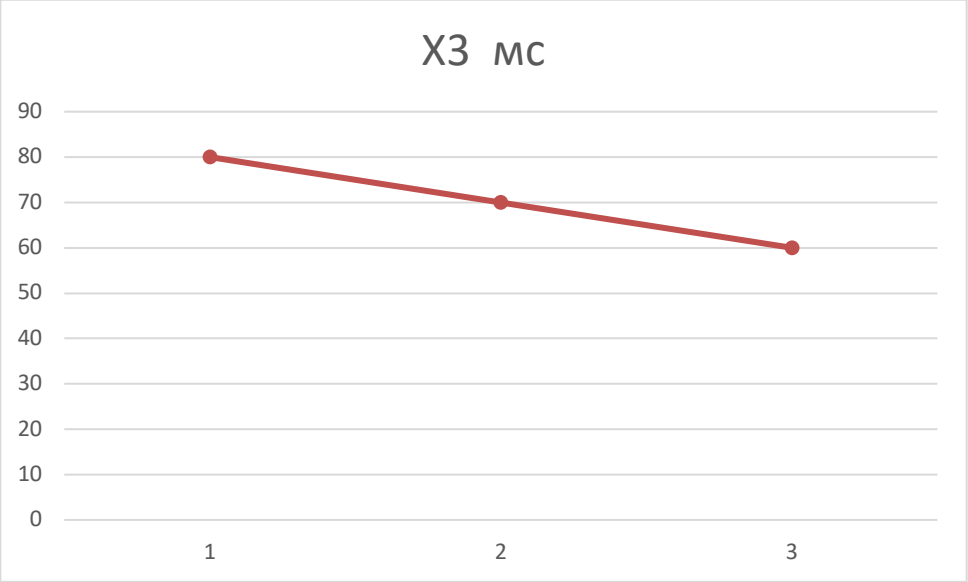


Рисунок 4.4 – X3, час попередньої обробки даних

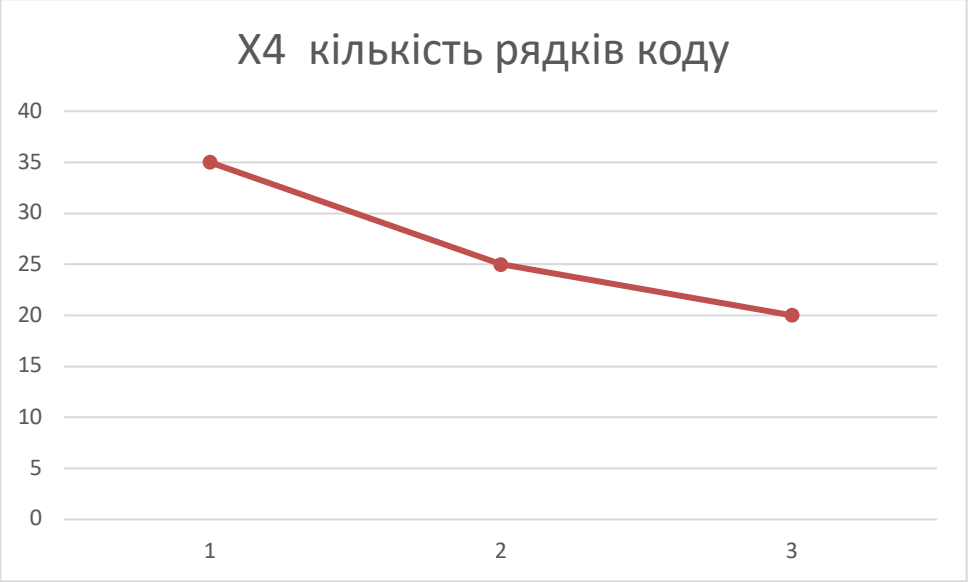


Рисунок 4.5 – X4, потенційний об’єм програмного коду

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 значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;
– перевірку придатності експертних оцінок для подальшого використання;
– визначення оцінки попарного пріоритету параметрів;
– обробку результатів та визначення коефіцієнту значимості.

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	Відхи-лення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	1	2	2	1	1	1	2	10	-7,5	56,25
X2	Об'єм пам'яті	Mb	3	4	3	3	4	3	4	24	6,5	42,25
X3	Час попередньої обробки даних	мс	2	1	1	2	2	2	1	11	-6,5	42,25
X4	Потенційний об'єм програмного коду	Кількість рядків коду	4	3	4	4	3	4	3	25	7,5	56,25
	Разом		10	10	10	10	10	10	10	70	0	197

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:

$$T = \frac{1}{n} R_{ij} = 17{,}5 \tag{4.2}$$

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

$$\Delta_i = R_i - T. \tag{4.3}$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197. \tag{4.4}$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 197}{7^2(4^3 - 4)} = 0{,}754 > W_k = 0{,}67. \tag{4.5}$$

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 занесемо 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	>	>	>	>	>	>	>	>	1,5
X2 і X4	<	>	<	<	>	<	<	<	0,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

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

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

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

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

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

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

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:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

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

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	
	1	2	3	4	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	1,5	0,5	4,5	0,28	16,25	0,28	59,125	0,28
X3	1,5	0,5	1	0,5	3,5	0,22	12,25	0,21	41,875	0,2
X4	1,5	1,5	1,5	1	5,5	0,34	21,25	0,35	77,875	0,36
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):

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

де n – кількість параметрів;

$K_{вi}$– 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	А	X1	95	20	0,16	3,4
F3	А	X2	90	24	0,28	8,54
	Б	X3	25	16	0,2	3,2
F4	А	X4	28	25	0,36	8,36

За даними з таблиці 4.6 за формулою:

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

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 3{,}4 + 8{,}54 + 8{,}36 = 20{,}3;$$

$$K_{K2} = 3{,}4 + 3{,}2 + 8{,}36 = 14{,}96\,.$$

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:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – 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_P =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_P =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 22000 грн., один аналітик в області даних з окладом 24000. Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} грн., \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_Ч = \frac{22000 + 22000 + 24000}{3 \cdot 21 \cdot 8} = 134{,}92 \ \ грн. \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_Ч \cdot Т_i \cdot К_Д, \quad (4.16)$$

де $С_Ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 134.92 \cdot 852 \cdot 1.2 = 137942{,}20$ грн.

II. $С_{ЗП} = 134.92 \cdot 868.88 \cdot 1.2 = 140675{,}14$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 137942{,}20 \cdot 0.22 = 30347{,}28$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 140675{,}14 \cdot 0.22 = 30948{,}53$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 22000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 22000 \cdot 0,2 = 52800 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 52800 \cdot (1 + 0.2) = 63360 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 63360 \cdot 0,22 = 13939,2 \text{ грн.}$$

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

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.4 \cdot 0.25 \cdot 10000 = 3500 \text{ грн.,}$$

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:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.4 \cdot 10000 \cdot 0.08 = 1120 \text{ грн.,}$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} =(Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.35 =$$

$$= 627,2 \text{ години},$$

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:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 627,2 \cdot 0,2 \cdot 0,3 \cdot 5,22 = 196,43 \text{ грн.},$$

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:

$$С_Н = Ц_{ПР} \cdot 0.67 = 10000 \cdot 0,67 = 6700 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{ЕКС} = C_{ЗП} + C_{ВІД} + C_{А} + C_{Р} + C_{ЕЛ} + C_{Н}, \quad (4.17)$$

$C_{ЕКС}$ = 63360 + 13939,2 + 3500 + 1120 + 196,43 + 6700 = 88815,63 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$C_{М\text{-}Г} = C_{ЕКС} / T_{ЕФ}$ = 88815,63 / 627,2 = 141,60 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:

$$C_{М} = C_{М-Г} \cdot T, \quad (4.18)$$

I. $C_{М}$ = 141,6 · 852 = 120643,2 грн.

II. $C_{М}$ = 141,6 · 868,88 = 123033,4 грн.

Накладні витрати складають 67% від заробітної плати:

$$C_{Н} = C_{ЗП} \cdot 0{,}67, \quad (4.19)$$

I. $C_{Н}$ = 137942,20 · 0,67 = 92421,27 грн.

II. $C_{Н}$ = 140675,14 · 0,67 = 94252,34 грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_{М} + C_{Н}, \quad (4.20)$$

I. $С_{ПП}$ = 137942,20 + 30347,28 + 120634,2 + 92421,27 = 381344,95 грн.

II. $С_{ПП}$ = 140675,14 + 30948,53 + 123033,4 + 94252,34 = 388909,41 грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{Кj} / С_{Фj}, \quad (4.21)$$

$$К_{ТЕР1} = 20,3 / 381344,95 = 5,323 \cdot 10^{-5},$$

$$К_{ТЕР2} = 14,96 / 388909,41 = 3,847 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 5,323 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, 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:

$К_{ТЕР} = 5,323 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

– вибір програмного продукту – Python;

– реалізація важливої постановки з допомогою вбудованих функцій;

– використання стандартного інтерфейсу для побудови значень.

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

4.8 Висновки до розділу 4

В даному розділі було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, 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 цій роботі було розглянуто технологію CoT (Chain of Thoughts) для використання з LLM (Large Language Models, Великі мовні моделі) з метою отримання ієрархічних структур. Завдяки доскональному вивченню схожих дослідів та статей, було виявлено різні стратегії використання даних технології для створення віртуального асистенту на основі бази даних, зібраних з телеграм каналів.

У першому розділі було знайдено модифікації методу CoT з метою покращення отриманих результатів, які генеруються за допомогою великої мовної моделі. Ця технологія дає чудову можливість уникнення галюцинацій та отриманню бажаного результату від моделі.

У другому розділі були розібрані теоретичні відомості основних технологій, що задіяні у даній роботі. Завдяки цьому, можна мати загальне уявлення про використані методики та мати більше можливостей при налаштуванні параметрів.

У третьому розділі описано програмну реалізацію, по-перше всього було проведено збір даних за допомогою парсингу Телеграм каналів. Це дає можливість отримати велику базу знань, на якій потім треба навчити модель. Після того як модель навчена на великому об’ємі даних, за допомогою розглянутої технології CoT та звичайного промптингу, проводиться створення ієрархічних структур про відомості з бази знань. Метрики показали відносну стійкість ієрархічних структур, що є свідченням стабільності стратегії, яка може бути побудована навколо ієрархічних структур.

Загалом, було оброблено 9787 повідомлень з мережі Telegram, які були зібрані, починаючи з 24 лютого 2023 року, та закінчуючи датою 15 травня 2024 року. При збору інформації з Інтернет ресурсів, було оброблено 33 сайти, що містять інформацію про дрони. Оброблені ресурси склали текстовий корпус, на якому були натреновані п’ять різних LLM моделей, та

результати яких були візуалізовано за допомогою застосунку Gephi. За допомогою різних метрик порівнянь, виявилось, що моделі сімейства GPT дають більш схожі результати між собою, в той час як модель Llama 2 давала більш комплексний результат. Побудовані моделі являють собою віртуальних асистентів, які здатні узагальнити інформацію у вигляді створення ієрархічної структури з великого та комплексного обсягу даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

[1] Mc Gurk S., Abela C., Debattista J. Towards Ontology Quality Assessment. *CEUR Workshop Proceedings*. 2017. URL: https://ceur-ws.org/Vol-1824/ldq_paper_2.pdf

[2] Ланде Д. В., Дмитренко О.О., Радзієвська О. Г. Побудова онтологій в галузі права за даними сервісу Google Scholar. *Інформація і право,* 2019, 1, С. 74-85.

[3] Ланде Д. В. Формування мереж природних ієрархій термінів на основі аналізу текстових корпусів з правової тематики. *Правова інформатика,* 2014, 1, С. 12-17.

[4] Lande D., Strashnoy L. Hierarchical Formation of Causal Networks Based on ChatGPT. *SSRN Electronic Journal*. 2023. 13 p. URL: http://dx.doi.org/10.2139/ssrn.4440629

[5] Cheng Peng, Xi Yang, Aokun Chen, Zehao Yu, Kaleb E Smith, Anthony B Costa, Mona G Flores, Jiang Bian, Yonghui Wu, Generative large language models are all-purpose text analytics engines: text-to-text learning is all your need, Journal of the American Medical Informatics Association, 2024. URL: https://doi.org/10.1093/jamia/ocae078

[6] Mandvikar S. Augmenting Intelligent Document Processing (IDP) Workflows with Contemporary Large Language Models (LLMs). *International Journal of Computer Trends and Technology*. 2023. Vol. 71, no. 10. P. 80–91. URL: https://doi.org/10.14445/22312803/ijctt-v71i10p110

[7] Minh Duc V., Han W., Zhuang L., Jieshan C., Shengdong Z., Zhenchang X., Chunyang C., GPTVoiceTasker: LLM-Powered Virtual Assistant for Smartphone. 2024. URL: https://doi.org/10.48550/arXiv.2401.14268

[8] Lande D., Strashnoy L., Driamov O. Analytic Hierarchy Process in the Field of Cybersecurity Using Generative AI. *SSRN Electronic Journal*. 2023. URL: https://doi.org/10.2139/ssrn.4621732

[9] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, Denny Zhou, Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, arxiv.org, 2022. URL: https://doi.org/10.48550/arXiv.2201.11903

[10] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Synthetic prompting: generating chain-of-thought demonstrations for large language models. In Proceedings of the 40th International Conference on Machine Learning (ICML'23), 2023. Vol. 202. JMLR.org, Article 1273, 30706–30775.

[11] Zhang Z., Yao Y., Zhang A., Tang X., Ma X., He Z., Wang Y., Gerstein M., Wang R., Liu G., Zhao H. Igniting language intelligence: The hitchhiker's guide from chain-of-thought reasoning to language agents. 2023. arXiv preprint arXiv:2311.11797. URL: https://arxiv.org/pdf/2311.11797

[12] Shizhe Diao, Pengcheng Wang, Yong Lin, Rui Pan, Xiang Liu, Tong Zhang, Active Prompting with Chain-of-Thought for Large Language Models, 2023, arXiv:2302.12246. URL: https://arxiv.org/pdf/2302.12246

[13] Gyeong-Geon Lee, Ehsan Latif, Xuansheng Wu, Ninghao Liu, Xiaoming Zhai, Applying large language models and chain-of-thought for automatic scoring, Computers and Education: Artificial Intelligence, Volume 6, 2024, 100213, ISSN 2666-920X, URL: https://doi.org/10.1016/j.caeai.2024.100213.

[14] Cutbill Adam, Monsler Eric, Hayashi Eric, Personalized Home Assistant Using Large Language Model with Context-based Chain of Thought Reasoning, Technical Disclosure Commons, (March 25, 2024); URL: https://www.tdcommons.org/dpubs_series/6814

[15] Jieyi Long, Large Language Model Guided Tree-of-Thought, 2023; arXiv:2305.08291, URL: https://arxiv.org/pdf/2305.08291

[16] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, Karthik Narasimhan, Tree of Thoughts: Deliberate Problem Solving with Large Language Models arXiv:2305.10601, URL: https://arxiv.org/pdf/2305.10601

[17] Zhenyu Bi, Daniel Hajialigol, Priya Pitre, Zhongkai Sun, Jie Hao, Xuan Wang, Tree- or Chain-of-Thought? Exploring Complex Reasoning in Multi-Hop Question Answering, URL: https://southnlp.github.io/southnlp2024/papers/southnlp2024-poster-69.pdf

[18] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L, Gomez A., Kaiser L., Polosukhin I. Attention Is All You Need. 2017. URL: https://doi.org/10.48550/arXiv.1706.03762

[19] Ajitesh Kumar, Large Language Models: Types, Examples, 2024, URL: https://vitalflux.com/large-language-models-concepts-examples/

[20] Ba J., Kiros J., Hinton G. Layer Normalization. 2016. URL: https://doi.org/10.48550/arXiv.1607.06450

[21] Pepperkorn M., Kouwenhoven T., Brown D., Jordanous A. Is Temperature the Creativity Parameter of Large Language Models? 2024. URL: https://doi.org/10.48550/arXiv.2405.00492

[22] Ackley D., Hinton G., Sejnowski T. A learning algorithm for Boltzmann Machines. 1985. URL: https://doi.org/10.1016/S0364-0213(85)80012-4

[23] Topsakal O., Akinci T., Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast, International Conference on Applied Engineering and Natural Sciences. 2023.

[24] Touvron H. et al, Llama 2: Open Foundation and Fine-Tuned Chat Models URL: https://doi.org/10.48550/arXiv.2307.09288

ДОДАТОК А. ЛІСТИНГ ПРОГРАМИ

Модуль tgparser.ipynb

```
from telethon.sync import TelegramClient
import csv

api_id = ***
api_hash = '***'
phone_number = '***'

client = TelegramClient('my_session', api_id, api_hash)

async def main():
    await client.start(phone_number)

    if not await client.is_user_authorized():
        await client.send_code_request(phone_number)
        await client.sign_in(phone_number, input('Enter the code: '))

    dialogs = await client.get_dialogs()
    print(dialogs)

    channel = None
    for dialog in dialogs:
        if dialog.is_channel:
            channel = dialog.entity
            break

    if channel is None:
        print("No channel found")
        return

    with open('rozvidkavoroga.csv', 'w', newline='', encoding='utf-8')
as csvfile:
        fieldnames = ['id', 'sender_id', 'date', 'message']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()

        async for message in client.iter_messages(channel, limit=None,
reverse=True):
            writer.writerow({
```

```
                    'id': message.id,
                    'sender_id': message.sender_id,
                    'date': message.date.strftime('%Y-%m-%d %H:%M:%S'),
                    'message': message.text
                })

    print(f"All messages have been saved")

with client:
    client.loop.run_until_complete(main())
```

Модуль drones_assistant.ipynb

```
pip install langchain==0.0.208

pip install deeplake openai==0.27.8 tiktoken

pip install unstructured selenium

# Додаємо API ключ OpenAI та ActiveLoop

import os

os.environ["OPENAI_API_KEY"] = "***"
os.environ["ACTIVELOOP_TOKEN"] = "***"

from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores import DeepLake
from langchain.text_splitter import CharacterTextSplitter
from langchain import OpenAI
from langchain.document_loaders import SeleniumURLLoader
from langchain import PromptTemplate

# Для інформації про дрони ми будемо використовувати такі посилання:
urls        =        ['https://www.bezpeka-shop.com/ua/blog/poleznye-
sovety/klassifikatsiya-dronov-kakie-vidy-i-tipy-byvayut-chast-pervaya/',
            'https://www.bezpeka-shop.com/ua/blog/poleznye-
sovety/klassifikatsiya-dronov-kakie-vidy-i-tipy-byvayut-chast-vtoraya/',
            'https://dou.ua/forums/topic/40895/',
            'https://worldvelosport.com/article/shho-take-dron-i-yaki-
voni-buvayut',
```

'https://www.flydron.com.ua/sovety/chto-takoe-dron-osobennosti-sovremennykh-kvadrokopterov',
'https://flytechnology.ua/chim-vidriznyayutsya-drony-vid-kvadrokopteriv',
'https://drony.org.ua/bezpilotni-litalni-aparaty-drony-i-kvadrokoptery-riznyci-mizh-nymy',
'https://ukrdrone.com.ua/shho-take-kvadrokopter/',
'https://duikt.edu.ua/ua/news-1-641-8062-scho-take-bpla-ta-yaka-riznicya-mizh-terminami-%E2%80%9Cdron%E2%80%9D-ta-%E2%80%9Cbpla%E2%80%9D-%E2%80%93-tematika-zustrichi-studentiv-specialnosti-kompyuterna-inzheneriya-z-fahivcyami-praktikami_kafedra-kompyuternoi-inzhenerii'
'https://chas.news/current/fpv-dron-scho-tse-take-yaki-zavdannya-vikonue-ta-skilki-koshtue',
'https://chas.news/current/fpv-dron-scho-tse-take-yaki-zavdannya-vikonue-ta-skilki-koshtue',

'https://ti.ua/ua/news/chto_takoe_kvadrokoptery_i_kakimi_oni_byvayut/',
'https://mil.in.ua/uk/articles/fpv-drony-zbroya-shho-zminyla-suchasnu-vijnu/',
'https://www.technotaras.com.ua/blog/dron-kamikadze',
'https://modelistam.com.ua/ua/klassy-kvadrokopterov-vidy-naznachenie-a-161/',
'https://speka.media/yaki-droni-je-u-rosiyan-velikii-oglyad-vrjx0p',
'https://tsn.ua/exclusive/v-ukrayini-budut-unikalni-vidi-viysk-scho-vidomo-pro-ce-2510860.html',
'https://dronecenter.ua/review-of-the-dji-phantom-4-pro-quadcopter',
'https://www.flydron.com.ua/ohlyady-kvadrokopteriv/chto-takoe-fpv-dron-i-dlya-chego-on-nuzhen',
'https://wondertech.ua/ua/blog/chim-vidriznyaetsya-fpv-dron',
'https://mind.ua/publications/20268730-shcho-take-fpv-dron-i-yak-navchitisya-nim-keruvati',
'https://stylus.ua/uk/articles/1609.html',
'https://ukraineaidmanual.com/drones-on-front-line/',
'https://fakty.com.ua/ua/ukraine/20231116-mozhe-stoyaty-za-atakamy-na-moskvu-ta-raketnyj-zavod-shho-vidomo-pro-ukrayinskyj-udarnyj-bpla-bober/',
'https://forbes.ua/war-in-ukraine/armii-droniv-yak-u-viyni-v-ukraini-bpla-zaminyuyut-artileriyu-aviatsiyu-i-kateri-06122022-10273',
'https://skadovsk.city/articles/274686/kvadrokopteri-ta-ihni-sferi-zastosuvannya',

```
            'https://neboperemogy.fund/news/shakhedy-ne-drony/',
            'https://www.radiosvoboda.org/a/drony-v-
armiyi/32697283.html',
            'https://www.holosameryky.com/a/ataky-drony/7570642.html',
            'https://glavcom.ua/country/incidents/minstratehprom-
povidomiv-jaki-vidi-bpla-zaraz-mozhe-vihotovljati-ukrajina-976281.html',
            'https://uk.wikipedia.org/wiki/%D0%94%D1%80%D0%BE%D0%BD',

'https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D0%B7%D0%BF%D1%96%D0%BB%D0%BE%D
1%82%D0%BD%D0%B8%D0%B9_%D0%BB%D1%96%D1%82%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%
B9_%D0%B0%D0%BF%D0%B0%D1%80%D0%B0%D1%82',
            'https://fx-drones.com/shcho-take-dron/']

    loader = SeleniumURLLoader(urls=urls)
    docs_not_splitted = loader.load()

    text_splitter        =        CharacterTextSplitter(chunk_size=1000,
chunk_overlap=0)
    docs = text_splitter.split_documents(docs_not_splitted)

    loader = SeleniumURLLoader(urls=urls)
    data = loader.load()

    print(data[0])

    embeddings = OpenAIEmbeddings(model="text-embedding-ada-002")

    my_activeloop_org_id = "dklepachevskyi"
    my_activeloop_dataset_name = "drones_assistant"
    dataset_path                                                          =
f"hub://{my_activeloop_org_id}/{my_activeloop_dataset_name}"
    db               =               DeepLake(dataset_path=dataset_path,
embedding_function=embeddings)

    db.add_documents(docs)

    query = "Що таке дрон?"
    docs = db.similarity_search(query)
    print(docs[0].page_content)

    template = """You are an exceptional support chatbot with exceptional
knowledge of drones that gently answer questions.
```

```
You know the following context information.

{chunks_formatted}

Answer to the following question from a user. Use only information from the previous context information. Do not invent stuff. Pe precise and accurate.

Question: {query}

Answer:"""

prompt = PromptTemplate(
    input_variables=["chunks_formatted", "query"],
    template=template,
)

query = "Чому дрони є важливою зброєю під час війни?"

docs = db.similarity_search(query)
retrieved_chunks = [doc.page_content for doc in docs]

chunks_formatted = "\n\n".join(retrieved_chunks)
prompt_formatted  =  prompt.format(chunks_formatted=chunks_formatted, query=query)

llm = OpenAI(model="gpt-3.5-turbo-instruct", temperature=0)
answer = llm(prompt_formatted)
print(answer)

from langchain import PromptTemplate, FewShotPromptTemplate

# Приклади для few-shot prompting
examples = [
    {
        "question": "Які основні компоненти дрону?",
        "answer": """
Чи потрібні додаткові питання тут: Так.
Наступне питання: Яка головна функція корпусу дрону?
Проміжна відповідь: Корпус дрону забезпечує захист внутрішніх компонентів та аеродинамічну форму.
Наступне питання: Яка роль пропелерів у дроні?
```

Проміжна відповідь: Пропелери створюють підйомну силу для польоту дрону.

Наступне питання: Яка функція камери у дроні?

Проміжна відповідь: Камера дозволяє знімати фото та відео з повітря.

Отже, основні компоненти дрону: корпус, пропелери, камера

"""",

},

{

"question": "Які є міри захищення дронів (безпека)? Назвіть 10 понять, не більше трьох слів кожне у вигляді списку",

"answer": """

Чи потрібні додаткові питання тут: Так.

Наступне питання: Що таке геозони?

Проміжна відповідь: Геозони - це віртуальні межі для обмеження польотів дронів.

Наступне питання: Що означає шифрування даних?

Проміжна відповідь: Шифрування даних захищає інформацію від несанкціонованого доступу.

Наступне питання: Що таке фізичні бар'єри?

Проміжна відповідь: Фізичні бар'єри запобігають доступу до дрону сторонніх осіб.

Отже, основні заходи безпеки: геозони, шифрування даних, фізичні бар'єри, захищене програмне забезпечення, стійкі до злому системи, обмеження доступу, захист від перехоплення, безпечні канали зв'язку, виявлення втручання, фаєрволи

"""",

},

{

"question": "Які типи дронів існують?",

"answer": """

Чи потрібні додаткові питання тут: Так.

Наступне питання: Які дрони використовуються для аерофотозйомки?

Проміжна відповідь: Для аерофотозйомки використовуються квадрокоптери.

Наступне питання: Які дрони застосовуються у військових цілях?

Проміжна відповідь: У військових цілях застосовуються бойові дрони.

Наступне питання: Які дрони використовуються для доставки?

Проміжна відповідь: Для доставки використовуються логістичні дрони.

Отже, основні типи дронів: квадрокоптери, бойові дрони, логістичні дрони

"""",

},

{

"question": "Які є переваги використання FPV дронів?",

```
            "answer": """
        Чи потрібні додаткові питання тут: Так.
        Наступне питання: Що таке FPV дрони?
        Проміжна відповідь: FPV дрони дозволяють бачити те, що бачить дрон, у реальному часі через камеру.
        Наступне питання: Які основні переваги FPV дронів у змаганнях?
        Проміжна відповідь: FPV дрони забезпечують високу маневреність і швидкість.
        Наступне питання: Як FPV дрони використовуються у кінематографії?
        Проміжна відповідь: FPV дрони дозволяють знімати динамічні кадри з незвичайних ракурсів.
        Отже, основні переваги FPV дронів: реальний час, висока маневреність, кінематографія
        """,
            },
        ]

        # Шаблон для контексту та запитань
        template = """You are an exceptional support chatbot with exceptional knowledge of drones that gently answer questions.

        You know the following context information, which includes added examples for better understanding.

        {chunks_formatted}

        Answer to the following question from a user. Use only information from the previous context information. Do not invent stuff. Be precise and accurate.

        Question: {query}

        Answer:"""

        # Шаблон для прикладів
        example_template = """
        User: {question}
        AI: {answer}
        """

        # Створення шаблону для прикладів
        example_prompt = PromptTemplate(
            input_variables=["question", "answer"],
```

```
        template=example_template
    )

    # Остання частина шаблону
    suffix = """
    User: {query}
    AI: """

    # Шаблон FewShotPromptTemplate
    few_shot_prompt_template = FewShotPromptTemplate(
        examples=examples,
        example_prompt=example_prompt,
        prefix=template,
        suffix=suffix,
        input_variables=["chunks_formatted", "query"],
        example_separator="\n\n"
    )

    query = "Чому дрони є важливою зброєю під час війни?"

    docs = db.similarity_search(query)
    retrieved_chunks = [doc.page_content for doc in docs]
    chunks_formatted = "\n\n".join(retrieved_chunks)

    chat = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)
    chain = LLMChain(llm=chat, prompt=few_shot_prompt_template)

    answer = chain.run({"chunks_formatted": chunks_formatted, "query":
query})
    print(answer)

    from langchain.chat_models import ChatOpenAI
    from langchain import LLMChain

    query = "які є Використання дронів? Типові і ні. Назвіть 10 понять, не
більше трьох слів кожне у вигляді списку"

    docs = db.similarity_search(query)
    retrieved_chunks = [doc.page_content for doc in docs]

    chunks_formatted = "\n\n".join(retrieved_chunks)
```

```
template = """You are an exceptional support chatbot with exceptional knowledge of drones that gently answers questions.

You know the following context information.

{chunks_formatted}

Answer the following question from a customer. Do not invent stuff. Be precise and accurate. Use all your knowledge

Question: {query}

Answer:"""

prompt = PromptTemplate(
    input_variables=["chunks_formatted", "query"],
    template=template,
)

prompt_inputs = {
    "chunks_formatted": chunks_formatted,
    "query": query
}

llm = ChatOpenAI(model="gpt-4-turbo", temperature=0.7)

chain = LLMChain(llm=llm, prompt=prompt)

answer = chain.run(prompt_inputs)
print(answer)
```

Модуль llama_drones_assistant.ipynb

```
!pip install transformers torch accelerate

pip install datasets flask

!huggingface-cli login

from transformers import AutoTokenizer
import transformers
```

```
      import torch
      from transformers import AutoModelForCausalLM, AutoTokenizer, Trainer,
TrainingArguments


      model_name = "meta-llama/Llama-2-7b-chat-hf"
      tokenizer         =         AutoTokenizer.from_pretrained(model_name,
use_auth_token=True)
      model        =        AutoModelForCausalLM.from_pretrained(model_name,
use_auth_token=True)

      import requests
      from bs4 import BeautifulSoup

      def scrape_website(url):
          output = requests.get(url)
          soup = BeautifulSoup(output.content, 'html.parser')
          text = soup.get_text()
          return text

      urls        =        ['https://www.bezpeka-shop.com/ua/blog/poleznye-
sovety/klassifikatsiya-dronov-kakie-vidy-i-tipy-byvayut-chast-pervaya/',
              'https://www.bezpeka-shop.com/ua/blog/poleznye-
sovety/klassifikatsiya-dronov-kakie-vidy-i-tipy-byvayut-chast-vtoraya/',
              'https://dou.ua/forums/topic/40895/',
              'https://worldvelosport.com/article/shho-take-dron-i-yaki-
voni-buvayut',
              'https://www.flydron.com.ua/sovety/chto-takoe-dron-
osobennosti-sovremennykh-kvadrokopterov',
              'https://flytechnology.ua/chim-vidriznyayutsya-drony-vid-
kvadrokopteriv',
              'https://drony.org.ua/bezpilotni-litalni-aparaty-drony-i-
kvadrokoptery-riznyci-mizh-nymy',
              'https://ukrdrone.com.ua/shho-take-kvadrokopter/',
              'https://duikt.edu.ua/ua/news-1-641-8062-scho-take-bpla-ta-
yaka-riznicya-mizh-terminami-%E2%80%9Cdron%E2%80%9D-ta-
%E2%80%9Cbpla%E2%80%9D-%E2%80%93-tematika-zustrichi-studentiv-specialnosti-
kompyuterna-inzheneriya-z-fahivcyami-praktikami_kafedra-kompyuternoi-
inzhenerii'
              'https://chas.news/current/fpv-dron-scho-tse-take-yaki-
zavdannya-vikonue-ta-skilki-koshtue',
              'https://chas.news/current/fpv-dron-scho-tse-take-yaki-
zavdannya-vikonue-ta-skilki-koshtue',
```

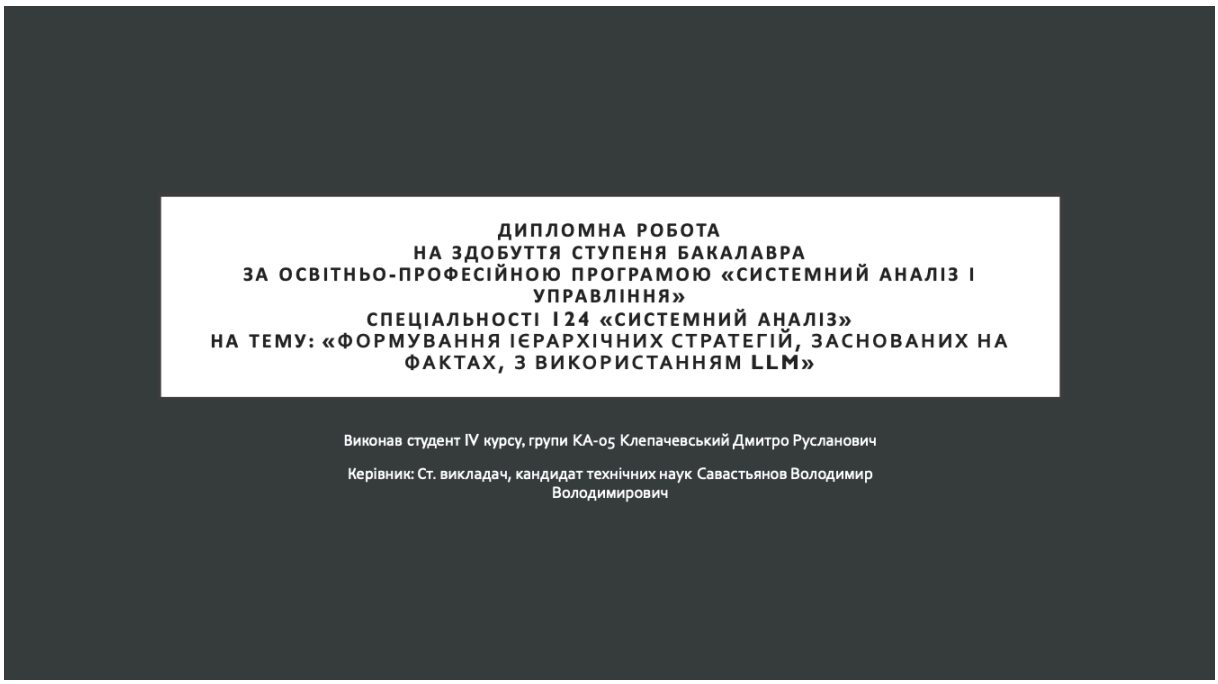
```
'https://ti.ua/ua/news/chto_takoe_kvadrokoptery_i_kakimi_oni_byvayut/',
              'https://mil.in.ua/uk/articles/fpv-drony-zbroya-shho-zminyla-
suchasnu-vijnu/',
              'https://www.technotaras.com.ua/blog/dron-kamikadze',
              'https://modelistam.com.ua/ua/klassy-kvadrokopterov-vidy-
naznachenie-a-161/',
              'https://speka.media/yaki-droni-je-u-rosiyan-velikii-oglyad-
vrjx0p',
              'https://tsn.ua/exclusive/v-ukrayini-budut-unikalni-vidi-
viysk-scho-vidomo-pro-ce-2510860.html',
              'https://dronecenter.ua/review-of-the-dji-phantom-4-pro-
quadcopter',
              'https://www.flydron.com.ua/ohlyady-kvadrokopteriv/chto-
takoe-fpv-dron-i-dlya-chego-on-nuzhen',
              'https://wondertech.ua/ua/blog/chim-vidriznyaetsya-fpv-dron',
              'https://mind.ua/publications/20268730-shcho-take-fpv-dron-i-
yak-navchitisya-nim-keruvati',
              'https://stylus.ua/uk/articles/1609.html',
              'https://ukraineaidmanual.com/drones-on-front-line/',
              'https://fakty.com.ua/ua/ukraine/20231116-mozhe-stoyaty-za-
atakamy-na-moskvu-ta-raketnyj-zavod-shho-vidomo-pro-ukrayinskyj-udarnyj-
bpla-bober/',
              'https://forbes.ua/war-in-ukraine/armii-droniv-yak-u-viyni-v-
ukraini-bpla-zaminyuyut-artileriyu-aviatsiyu-i-kateri-06122022-10273',
              'https://skadovsk.city/articles/274686/kvadrokopteri-ta-ihni-
sferi-zastosuvannya',
              'https://neboperemogy.fund/news/shakhedy-ne-drony/',
              'https://www.radiosvoboda.org/a/drony-v-
armiyi/32697283.html',
              'https://www.holosameryky.com/a/ataky-drony/7570642.html',
              'https://glavcom.ua/country/incidents/minstratehprom-
povidomiv-jaki-vidi-bpla-zaraz-mozhe-vihotovljati-ukrajina-976281.html',
              'https://uk.wikipedia.org/wiki/%D0%94%D1%80%D0%BE%D0%BD',

'https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D0%B7%D0%BF%D1%96%D0%BB%D0%BE%D
1%82%D0%BD%D0%B8%D0%B9_%D0%BB%D1%96%D1%82%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%
B9_%D0%B0%D0%BF%D0%B0%D1%80%D0%B0%D1%82',
              'https://fx-drones.com/shcho-take-dron/']

      data = [scrape_website(url) for url in urls]

      from datasets import Dataset
```

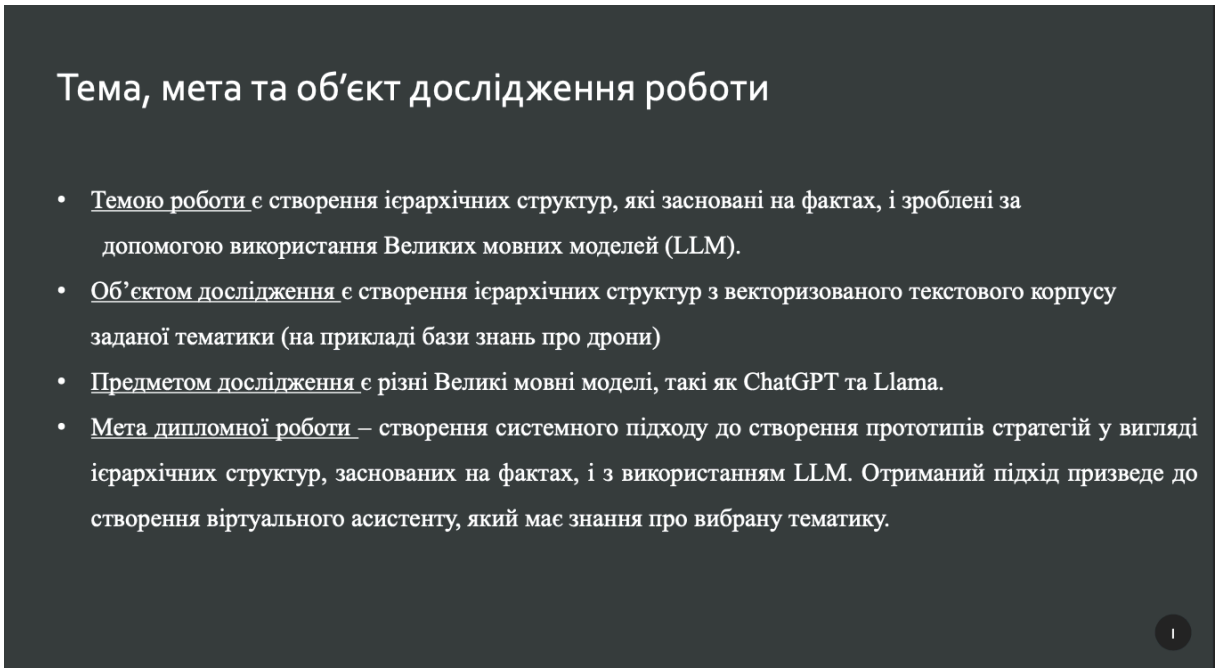
```
    def tokenize_function(examples):
        return      tokenizer(examples["text"],      padding="max_length",
truncation=True)

    raw_data = {"text": data}
    dataset = Dataset.from_dict(raw_data)
    tokenized_datasets = dataset.map(tokenize_function, batched=True)

    def get_llama_response(prompt: str) -> None:
        """
        Generate a response from the Llama model.

        Parameters:
            prompt (str): The user's input/question for the model.

        Returns:
            None: Prints the model's response.
        """
        sequences = llama_pipeline(
            prompt,
            do_sample=True,
            top_k=10,
            num_return_sequences=1,
            eos_token_id=tokenizer.eos_token_id,
            max_length=256,
        )
        print("Chatbot:", sequences[0]['generated_text'])

    llama_pipeline = pipeline(
        "text-generation",  # LLM task
        model=model,
        tokenizer=tokenizer,
        torch_dtype=torch.float16,
        device_map="auto",
    )

    prompt = 'Які дрони є найефективнішими для розвідки?'
    get_llama_response(prompt)
```

Модуль check_overlapping_nodes.ipynb

```
import numpy as np
import pandas as pd

# імпортуємо таблицю, в якій кожна колонка містить елементи для різних моделей
df = pd.read_excel('gpt_results.xlsx')
overlap_matrix = np.zeros((15, 15), dtype=int)
columns = df.columns

for i in range(len(columns)):
    for j in range(len(columns)):
        # Знайти число елементів які перетинаються між різними моделлями
        overlap_matrix[i, j] = len(set(df[columns[i]]) & set(df[columns[j]]))

# Перетворюємо результати у тип даних Датафрейм та зберігаємо як таблицю
overlap_df = pd.DataFrame(overlap_matrix, index=columns, columns=columns)
overlap_df.to_csv("overlapping_features.csv")
```

ДОДАТОК Б. ГРАФІЧНИЙ МАТЕРІАЛ





Актуальність та очікуваний результат роботи

- Актуальність роботи пов'язана з великою кількістю релевантної інформації, яку можна швидко та якісно обробити та представити у зручному вигляді за допомогою LLM.
- В результаті роботи на мові Python були побудовані та навчені різні моделі, такі як ChatGPT та Llama, моделі були навчені на зібраній базі знань про дрони та були порівняні між собою за різними метриками оцінювання ієрархічних структур. За допомогою засобу Gephi результати, отримані за допомогою LLM, були відображені у зручному вигляді з метою візуалізації.

2

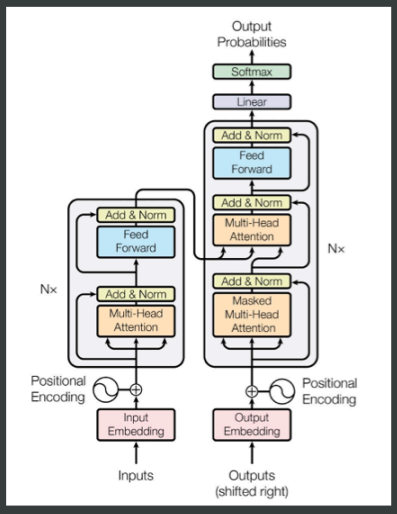
Набір даних

- У рамках цієї роботи для подальшого аналізу було використано декілька баз знань – одна з використання постів зі каналів мережі Telegram та з сайтів, що містять інформацію про дрони.
- Було розроблено програмний код на Python, який використовує бібліотеку Telethon для зчитування даних з релевантнів каналів мережі Telegram. Мета полягала в тому, щоб автоматизувати процес збору повідомлень з каналів, зберігаючи їх у структурованому форматі для подальшого аналізу.
- Для поповнення бази знань інформацією з релевантних сайтів був виконаний парсинг із використанням бібліотек Requests, Selenium мови програмування Python.
- Загалом, було оброблено 9787 повідомлень з мережі Telegram, які були зібрані, починаючи з 24 лютого 2023 року, та закінчуючи датою 15 травня 2024 року. При збору інформації з Інтернет ресурсів, було оброблено 33 сайти, що містять інформацію про дрони.

3

Структура LLM (Large Language Model)

- **Модель трансформер** - це найпоширеніша архітектура великої мовної моделі. Вона складається з енкодера та декодера.
- Трансформери працюють з механізмами самоуваги, що дозволяє моделі навчатися швидше
- Енкодер у трансформері переводить вхідну послідовність відображення символів $(x_1, \dots, x_n)$ до послідовності неперервних відображень $z = (z_1, \dots, z_n)$. Отримавши z, декодер генерує вихідну послідовність символів $(y_1, \dots, y_n)$



4

Embedding Layer

- Шар вбудовування (embedding layer) створює репрезентації з вхідного тексту, фіксуючи семантичне і синтаксичне значення тексту, щоб модель могла зрозуміти контекст. Шар вбудування складається з двох компонент: Positional Encoding та Input Embedding, після чого дві компоненти додаються.

Positional Encoding:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

5

Normalization Layer

- Шар нормалізації (Layer Normalization) напряму оцінює нормалізаційну статистику від суми вхідних даних до нейронів в схованих шарах моделі, таким чином, що нормалізація не впроваджує ніяких нових залежностей
- Ця техніка гарно працює з моделями трансформерів, що використовуються LLM, та завдяки цьому підвищується якість моделі та час тренування.

Layer Normalization:

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}$$

6

Attention Mechanism

- Механізм уваги (attention mechanism) дозволяє мовній моделі зосередитися на конкретних частинах вхідного тексту, які мають відношення до поставленого завдання.
- Multi-Head Attention є ключовою компонентою моделі трансформера, яка використовує декілька одиничних механізмів уваги. Механізм уваги може бути описаний як перетворення трьох компонент: Query (запит), Key (ключ), Value (значення), до вихідного значення.

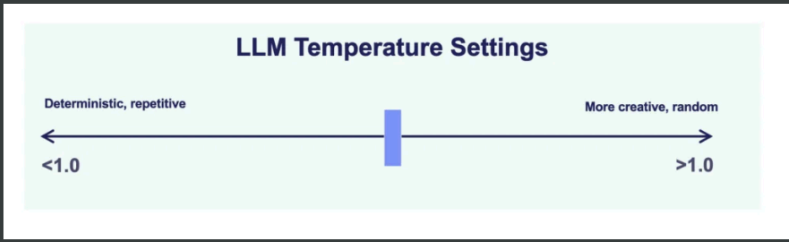
Attention Mechanism:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

7

Параметр Температури моделі (temperature)

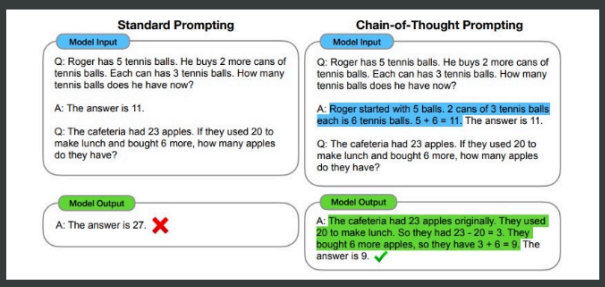
- Оскільки часто великі мовні моделі використовуються у задачах, які вимагають креативність, був створений параметр температури моделі (model's temperature).
- Параметр температури регулює кількість випадковості (randomness) моделі, яка призводить до більш різносторонніх результатів



8

Chain of Thoughts

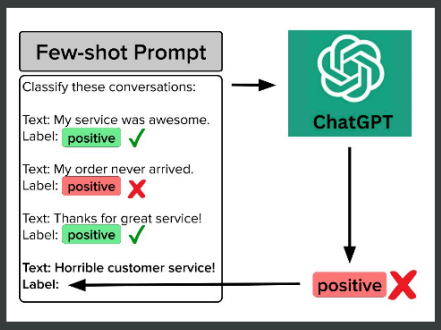
- Ланцюжок думок (Chain Of Thoughts) - це техніка промпт інжинірингу, призначена для того, щоб спрямувати великі мовні моделі (LLM) на послідовність проміжних кроків, що ведуть до бажаної відповіді. Цей підхід розширює можливості міркувань LLM (Великої мовної моделі), дозволяючи їм вирішувати один крок за раз, замість того, щоб вирішувати всю проблему одночасно.



9

Few-Shot Prompting

Few-shot Prompting техніка тренує модель на кращу продуктивність. Цей метод промпт інжиніринга полягає в тому, що на додаток до запиту LLM надаються декілька прикладів бажаного результату



10

Використані технології для тренування моделі

LangChain - це фреймворк, призначений для розробки додатків, які ефективно та раціонально використовують великі мовні моделі (LLM)

Deep Lake – векторне сховище, що забезпечує зберігання вбудовувань та їхніх метаданих. Deep Lake інтегрується з LangChain, що полегшує розробку додатків

Selenium – це потужний модуль мови Python, який використовується для контролю веб браузерів та запровадження автоматизації роботи з браузерами.

Requests – це бібліотека мови Python, яка дозволяє відправляти HTTP запити, використовуючи мову Python

11

Тестування різних ВММ для збору даних для подальшого формування ієрархічних структур

Для тестування запитів та отримання результатів були обрані наступні моделі від OpenAI: ChatGPT 3.5 turbo, ChatGPT 4.0, ChatGPT 4o, ChatGPT 4.0 turbo та модель від Meta LLAMA 2. Для кожної моделі був присутній PromptTemplate та приклад, щоб натренувати модель видавати відповіді у стилі промптингу Chain Of Thoughts, тобто отримувати кінцевий результат після логічного ланцюжка думок.

Усі промпти базувались на 5 описових характеристиках дронів: які типи дронів бувають, які характеристики дронів, міри безпеки запроваджені в дрони, технічне обслуговування дронів, та використання дронів.

Згенеровані відповіді кожної моделі відрізнялись та містили як пункти, так і підпункти для кожної описової характеристики.

Всі відповіді були збережені в індивідуальні CSV файли для подальшого формування ієрархічних структур у програмі Gephi

12

Метрики оцінювання ієрархічних структур

- Загальна кількість вузлів ієрархії $N = \sum_{i=1}^{L} n_i$
- Кількість вузлів на останньому рівні $L = |\{v \in V \mid level(v) = max_i(level_i)\}|$
- Глибина вузлів гілок $D = \max(depth(v) \mid v \in V)$
- Кількість співпадаючих вузлів $M = \sum_{i=1}^{N} \delta(a_i, b_i)$

13

Результати 1. Парсинг даних з Телеграм

- Програмний код призначений для зчитування повідомлень з певного каналу Telegram та збереження їх у CSV файл для подальшої обробки

```python
async def main():
    await client.start(phone_number)

    if not await client.is_user_authorized():
        await client.send_code_request(phone_number)
        await client.sign_in(phone_number, input('Enter the code: '))

    dialogs = await client.get_dialogs()
    print(dialogs)

    channel = None
    for dialog in dialogs:
        if dialog.is_channel:
            channel = dialog.entity
            break

    if channel is None:
        print("No channel found")
        return

    with open('drones.csv', 'w', newline='', encoding='utf-8') as csvfile:
        fieldnames = ['id', 'sender_id', 'date', 'message']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()

        async for message in client.iter_messages(channel, reverse=True):
            writer.writerow({
                'id': message.id,
                'sender_id': message.sender_id,
                'date': message.date.strftime('%Y-%m-%d %H:%M:%S'),
                'message': message.text
            })
with client:
    client.loop.run_until_complete(main())
```

14

Результати 2. Парсинг даних з сайтів

- В рамках цієї роботи парсинг даних з сайтів з інформацією про дрони був виконаний із використанням бібліотек Requests Selenium мови програмування Python.

15

Результати 3. Навчання моделі на текстовому корпусі

- Коли база знань перетворена у векторний вигляд (Embedding Representation), можна перейти до задання запитів (промптів) до моделі, оскільки тепер обрана мовна модель натренована на текстовому корпусі та має інформацію про зібрані дані

```
embeddings = OpenAIEmbeddings(model="text-embedding-ada-002")

my_activeloop_org_id = "dklepachevskyi"
my_activeloop_dataset_name = "drones_assistant"
dataset_path = f"hub://{my_activeloop_org_id}/{my_activeloop_dataset_name}"
db = DeepLake(dataset_path=dataset_path, embedding_function=embeddings)

db.add_documents(docs)
```

16

Результати 4. Навчання моделі на текстовому корпусі

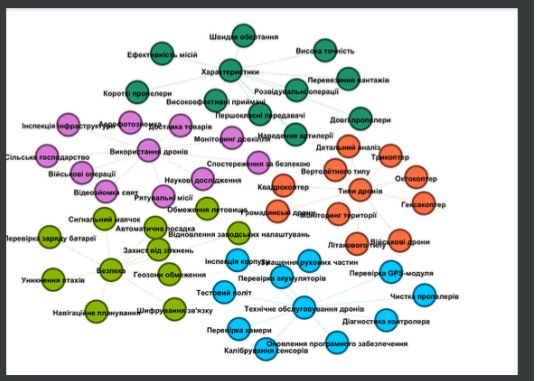
- Оскільки тепер модель натренована на текстовому корпусі, то можна приступати до відправлення запитів (промптів) до моделі

- Подібним чином було натреновано 5 різних видів моделей: GPT 3.5, GPT 4.0, GPT 4 turbo, GPT 4o, Llama 2. Кожна модель була натренована з трьома різними значеннями параметру температури – 0, 0.3, 0.7.

17

Результати 5. Створення ієрархічних структур

- Після того як моделі згенерували результати, то за допомогою застосунки Gephi вони перетворюються у ієрархічну репрезентацію.



- Наприклад, модель GPT 4 turbo з температурою 0.7, згенерувала наступну структуру:

18

Результати 6. Порівняння моделей

- За допомогою метрик, можна оцінити отримані ієрархічні структури за допомогою різних метрик, щоб зрозуміти комплексність та складність різних використаних моделей (ChatGPT, Llama).

Таблиця 1 – Результати метрик різних LLM

Модель	Температура	Загальна к-ть вузлів (nodes)	К-ть вузлів на останньому рівні	Глибина вузлів
GPT-3.5	0	100	42	3
GPT-3.5	0.3	78	73	2
GPT-3.5	0.7	67	62	2
GPT 4.0	0	55	50	2
GPT 4.0	0.3	55	50	2
GPT 4.0	0.7	54	49	2
GPT 4o	0	57	52	2
GPT 4o	0.3	55	50	2
GPT 4o	0.7	53	48	2
GPT 4.0 turbo	0	53	48	2
GPT 4.0 turbo	0.3	57	52	2
GPT 4.0 turbo	0.7	61	56	2
Llama 2	0	71	56	3
Llama 2	0.3	83	62	3
Llama 2	0.7	65	46	3

19

Результати 7. Порівняння моделей

- Було обраховано кількість співпадінь попарно між кожною моделлю.
- Моделі однієї самої LLM, але з різними температурами, все одно мають найбільшу кількість співпадінь.
- Також, співпадінь між моделями сімейства GPT також більше, аніж кількість співпадінь між моделями Llama 2 та моделями GPT.

Модель	4o_0	4o_0.3	4O_0.7	4turbo_0	4turbo_0.3	4turbo_0.7	4_0	4_0.3	4_0.7	3.5_0	3.5_0.3	3.5_0.7	llama_0	llama_0.3	llama_0.7
4o_0	58	30	25	14	14	11	13	10	15	11	12	9	6	6	6
4o_0.3	30	58	31	12	13	12	11	17	15	12	15	12	6	6	8
40_0.7	25	31	56	12	13	8	8	11	9	11	14	11	6	6	6
4turbo_0	14	12	12	57	36	23	16	10	13	9	7	13	6	6	7
4turbo_0.3	14	13	13	36	58	22	17	16	16	12	8	8	6	6	8
4turbo_0.7	11	12	8	23	22	56	12	11	12	15	7	7	6	6	7
4_0	13	11	8	16	17	12	56	19	22	7	8	12	6	6	6
4_0.3	10	17	11	10	16	11	19	56	19	12	11	7	6	6	8
4_0.7	15	15	9	13	16	12	22	19	56	11	10	8	6	6	7
3.5_0	11	12	11	9	12	15	7	12	11	96	15	10	5	5	6
3.5_0.3	12	15	14	7	8	7	8	11	10	15	98	19	5	5	5
3.5_0.7	9	12	11	13	8	7	12	7	8	10	19	78	6	6	6
llama_0	6	6	6	6	6	6	6	6	6	5	5	6	71	14	9
llama_0.3	6	6	6	6	6	6	6	6	6	5	5	6	14	83	8
llama_0.7	6	8	6	7	8	7	6	8	7	6	5	6	9	8	65

20

Висновки

- Було зібрано текстовий корпус з різних каналів мережі Telegram, а також з різних ресурсів інтернету та сайтів. Загалом було оброблено 9787 повідомлені з каналі мережі Telegram, та 33 сайти з Інтернет ресурсів
- Моделі сімейства GPT та Llama 2 були натреновані на зібраному текстовому корпусі про дрони. Завдяки цьому, моделі змогли створити ієрархічні представлення структур текстового корпусу.
- Завдяки додатку Gephi результати LLM були візуалізовані. Було виявлено, що більш комплексна модель, така як Llama 2, генерує більше відповідей на такі самі запити, які отримували й інші моделі.
- При цьому, модель Llama 2 завжди генерувала три ієрархічні рівні, в той час як більшість з моделей сімейства GPT зазвичай генерували лише два.
- В рамках цієї роботи, було побудовано віртуального асистента на основі різних LLM моделей, який здатен швидко представити комплексну структуру знань у зручному ієрархічному вигляді.

21

Дякую за увагу!

22