Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Мультимодальна мовна модель на базі мультиагентної архітектури для спільної обробки текстової та візуальної інформації
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-4

Іван Гончар
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки

(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект

(повна назва освітньої програми)
Керівник доц. Олександр Шевченко
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Гончару Івану Михайловичу
(прізвище, ім'я, по батькові)

1. Тема роботи Мультимодальна мовна модель на базі мультиагентної архітектури для спільної обробки текстової та візуальної інформації
затверджена наказом університету від 19 травня 20 25 р. № 378Ст
2. Термін подання студентом роботи до екзаменаційної комісії 17 червня 20 25 р.
3. Вихідні дані до роботи офіційна документація Langchain та Langgraph, Hugging Face, Transformers, Pytorch, Streamlit.

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та постановка задачі
2) Теоретичний аналіз
3) Опис програмної реалізації

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Дослідження предметної галузі	20.05.2025 – 22.05.2025	виконано
3	Постановка задачі	23.05.2025	виконано
4	Аналіз теоретичної частини	24.05.2025 – 27.05.2025	виконано
5	Програмна реалізація	28.05.2025	виконано
6	Оформлення пояснювальної записки	30.05.2025	виконано
7	Захист	17.06.2025	

Дата видачі завдання 19 травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Олександр Шевченко
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 67 с., 49 рис., 1 дод., 37 джерел.

AGENT, CHATGPT, LLAMA, LLM, MULTIMODALITY, PYTHON, TRANSFORMER.

Об’єкт дослідження – процес створення мультимодальних систем на основі агентних систем під контролем LLM моделей.

Предмет дослідження – мультимодальна агентна система для розгалуженого аналізу запитів користувача з глибокою обробкою різноманітних типів даних.

Мета роботи – розробка мультимодальної системи на основі моделі Transformer та LLM, на основі агентів для реалізації мультимодального аналізу даних та створенню чіткого потоку генерації та обробки мультимодальних даних.

Методи дослідження – аналіз існуючих наукових робіт, аналіз користувацьких вимог, аналіз існуючих аналогів та методів розробки мультимодальних систем.

У результаті розробки частини кваліфікаційної роботи було створено агентну систему для мультимодальної обробки та генерації інформації, що дозволить створити систему, яка буде більш повно задовольняти потреби користувача у одній екосистемі. Для реалізації даного проекту було використано Hugging Face, Llama 3.2 Vision, Langchain, Langgraph, ChatGPT API.

ABSTRACT

Bachelor`s thesis contains: 67 pp., 49 fig., 1 ann., 37 references.

AGENT, CHATGPT, LLAMA, LLM, MULTIMODALITY, PYTHON, TRANSFORMER.

The object of research is the process of creating multimodal systems based on agent systems under the control of LLM models.

The subject of research is a multimodal agent-based system for in-depth analysis of user requests with deep processing of various types of data.

Purpose – to develop an agent-based multimodal system based on the Transformer and LLM models to implement multimodal data analysis and create a clear flow of multimodal data generation and processing.

Research methods – analysis of existing scientific papers, analysis of user requirements, analysis of existing analogues and methods of developing multimodal systems.

As a result of the development of the qualification work, an agent system for multimodal processing and generation of information was created, which will create a system that will more fully meet the needs of the user in one ecosystem. To implement this project, Hugging Face, Llama 3.2 Vision, Langchain, Langgraph, ChatGPT API were used.

ЗМІСТ

Вступ 7
1 Аналіз предметної галузі 8
1.1 Аналіз розвитку методів аналізу текстової інформації 8
1.2 Аналіз існуючих тенденцій на ринку 15
1.3 Постановка задачі 18
1.3.1 Аналіз методів вирішення поставленого питання 18
1.3.2 Формування вимог до системи 19
1.3.3 Аналіз програмного забезпечення системи 20
2 Теоретичний аналіз 22
2.1 Трансформери як основні моделі сучасного світу 22
2.1.1 Ключові блоки 22
2.1.2 Архитектура 30
2.1.3 LLM quantization 37
2.1.4 LoRA 39
2.2 ChatGPT 42
2.3 Llama3 44
2.4 Аналіз агентів та їх застосування 47
3 Опис програмної реалізації 50
3.1 Ініціалізація додаткових моделей 50
3.2 Створення ключового агента 53
3.3 Створення додаткової оболонки 59
Висновки 62
Перелік джерел посилання 63
Додаток А Відомість кваліфікаційної роботи 67

ВСТУП

Сучасний світ інформаційних технологій стрімко розвивається та все більше росте потреба у створенні систем, які будуть спроможні оперувати не лише природно-мовною інформацією а й інтегруватися з іншими типами даних – зображення, аудіо, відео та інші. Не дивлячись на стрімкий розвиток сучасних LLM моделей, їх можливості аналізувати та виконувати різноманітні операції, вони все одно їх можливості є обмеженими через фокусування на одному з типів даних. Через дані обмеження світова спільнота стала активно розвивати напрямок мультимодальних систем, які буде можливо більш глибоко інтегрувати у сучасному світі та створити гнучкі системи без значних обмежень, які існують на сьогоднішній день. Прикладом таких систем можуть слугувати останні версії найпопулярніших LLM моделей: сімейство GPT, Gemini, Llama Vision-Instruct. Дані моделі довели, що створення даних систем має під собою окрім функціонального різноманіття так і ефективний шлях передачі інформації між різними доменами, що створює якісний ріст існуючих моделей.

Актуальність даної роботи базується на провідних дослідах головних компаній у сфері створення ШІ та необхідності передачі інформації, яку акумулюють сучасні LLM для створення універсальної системи у світі аналізу даних. Другим підтвердженням актуальності даної роботи є створення фреймворків та методів створення агентних систем, які намагаються вирішити дану проблему через використання різноманітних моделей задля створення універсальної системі у вигляді динамічної структури.

Таким чином, створення системи, яка буде оперувати різноманітними типами даних є перспективним та актуальним проектом, який буде використовувати знання LLM моделей повноцінно без обмежень у типі даних, як це було раніше.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

В останні роки технології великих мовних моделей (LLM) переживають стрімкий розвиток і знаходять все ширше застосування в різних областях: від чат-ботів і систем автоматизації бізнес-процесів до інтелектуальних помічників у науці та освіті. Паралельно активно розвивається напрям агентних систем – розподілених програмних комплексів, здатних самостійно приймати рішення, взаємодіяти один з одним і адаптуватися до умов середовища, яке може стрімко змінюватись через стрімкий розвиток технологій. Об'єднання можливостей LLM та агентниї архітектури відкриває нові перспективи: такі гібридні системи можуть не тільки генерувати осмислений текст, а й вибудовувати складні сценарії поведінки, планувати дії та відповідати за їх виконання.

Проте на стику цих напрямів залишається низка невирішених питань. З одного боку, сучасні LLM демонструють високу якість генерації послідовностей і здатність обробляти запити користувача, але їх поведінка може бути неконтрольованою або непередбачуваною. З іншого боку, класичні агентні системи, хоч і забезпечують чітку логіку взаємодії та контроль за виконанням завдань, часто обмежені набором заздалегідь прописаних правил і погано масштабуються на роботу з неструктурованими даними. Важливо зрозуміти, як поєднати сильні сторони обох підходів, мінімізуючи їх слабкості.

1.1 Аналіз розвитку методів аналізу текстової інформації

Аналіз текстової інформації став ключовою областю розвитку сучасного штучного інтелекту та пройшов через значні зміні від базового символьних методів до надглибоких нейронних мереж.

Першим етапом розвитку NLP був машинний переклад. Саме дана галузь аналізу текстової інформації стала основою для подальшого розвитку

усього напрямку NLP та створення сучасних LLM моделей як новітні версії ChatGPT. Першою з створених робіт, які були присвячена текстовій граматиці датується 1957 роком, автором Ноамом Чомскім, який висвітлив першу ідею машинного перекладу та генеративної граматики. Дана робота стала проривом та була основою для стрімкого розвитку популярності NLP у тих роках.

Але не дивлячись на значний ажіотаж та спроби розвитку даного напрямку, через складній задачі та недостатнього розвитку обчислювальних засобів у тих рока ALPAC (Automatic Language Processing Advisory Committee) представив звіт у якому висвітлив невигідність вкладу ресурсів до дослідження технологій пов’язаних з машинним перекладом та NLP вчасності, через що дослідження даної галузі не мало значного розвитку до 1980-их років.

Саму у 1980-их роках стався других цикл розвитку NLP технологій, який базувався на символічних підходах або ж на експертних системах. Даний тип методів базувався на чітко сформованих правилах, словниках, граматиці та онтологічних системах. Головна ідея цих методів була у представлені мов через дискретні символи такі як лексеми й їх оперуванням. Дані системи були розроблені для імітації поведінки людини-експерта. Однією з найпопулярніших систем у той момент був PATR-II, розроблений Стюартом Шайбером в 1986 році.

Дана система стала першою практичною реалізацією уніфікованих граматик. Суть даної системи була виражена у виді правила: «нетермінал – послідовність символів», які мали у собі класичний для мовної інтерпретації атрибути такі як род, число, семантичні мітки та інші. За допомогою даних атрибутів система могла більш точно аналізувати слова та створювати більш точні послідовності слів та якісніший переклад текстів.

Не дивлячись на значний розвиток методів NLP та вдосконаленню символьних методів у 1990-их роках були створені різноманітні статистичні методи, які мали значну перевагу над минулим поколінням NLP методів.

Однім із перших та найпопулярніших методів став Naive Bayes, який базувався на однойменній теоремі. Головною перевагою даного та наступних статистичних методів це автоматичний аналіз інформації, через що не було більше необхідно створювати складні масиви лінгвістичних правил. Окрім значного поліпшення точності дані методи принесли значну масштабованість без необхідності складної обробки навчальних даних (рисунок 1.1).

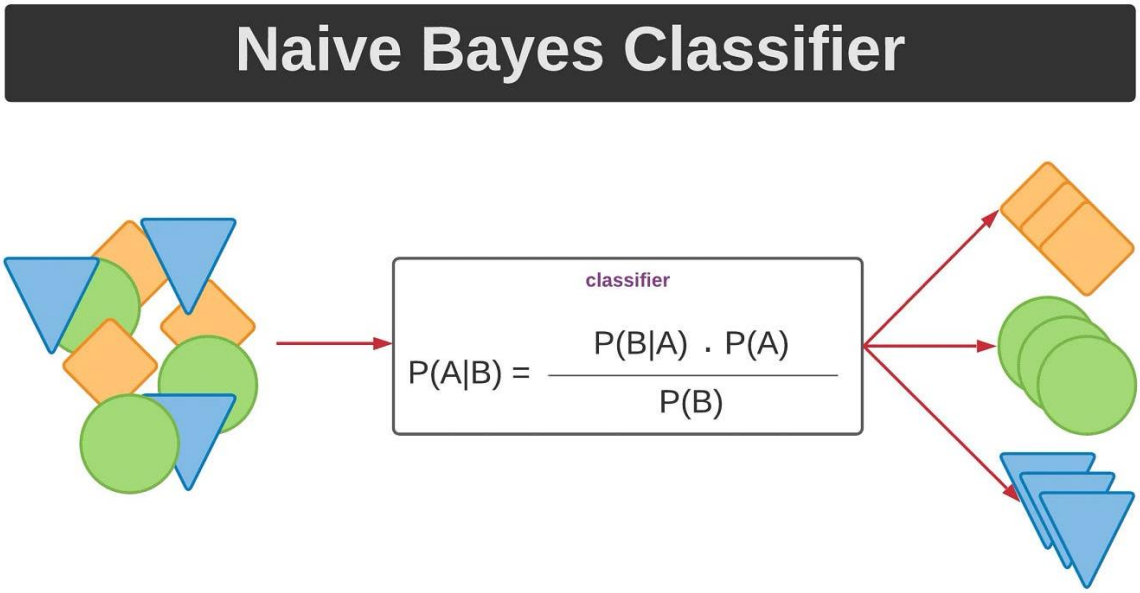

Рисунок 1.1 – Структура методу Naive Bayes [1]

Другим та найпопулярнішим методом став метод опорних векторів або ж SVM. Даний метод базується на розподіленні даних через створення гіперплощини та максимізації відстані опорних векторів, прикладів даних, які знаходяться ближче всього до гіперплощини. SVM став головним методом обробки текстової інформації через здатність обробки даних високої розмірності, що дає змогу знаходити більш складні залежності у даних (рисунок 1.2).

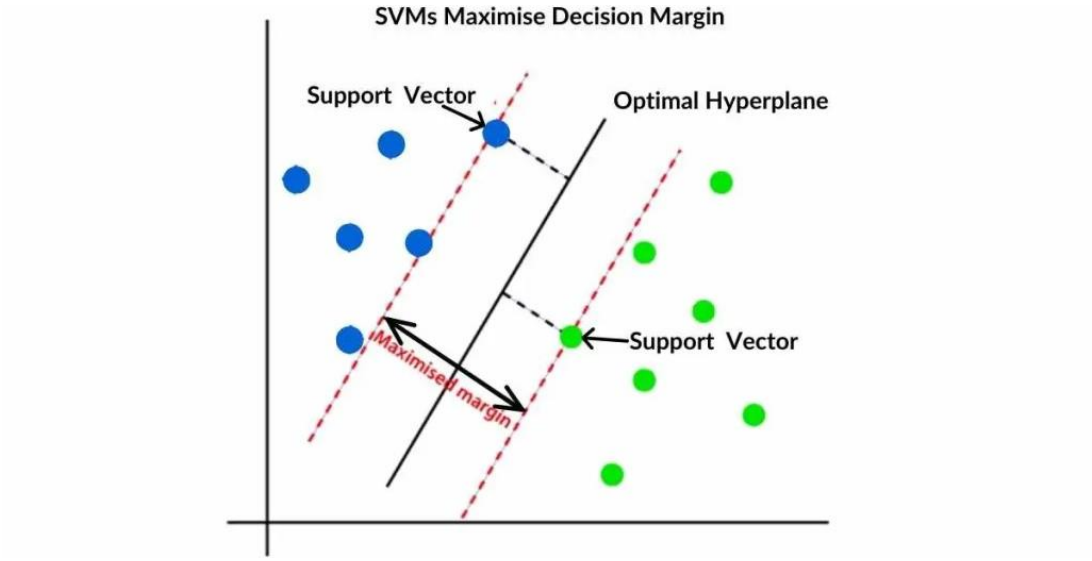

Рисунок 1.2 – Структура методу SVM [2]

Також у даний час були створені перші версії RNN моделі, які отримали більший розвиток у наступних десятиліттях. Головна особливість RNN моделей став рекурентний виклик спеціалізованих нейронних блоків, які зберігають інформацію з попередніх кроків (рисунок 1.3).

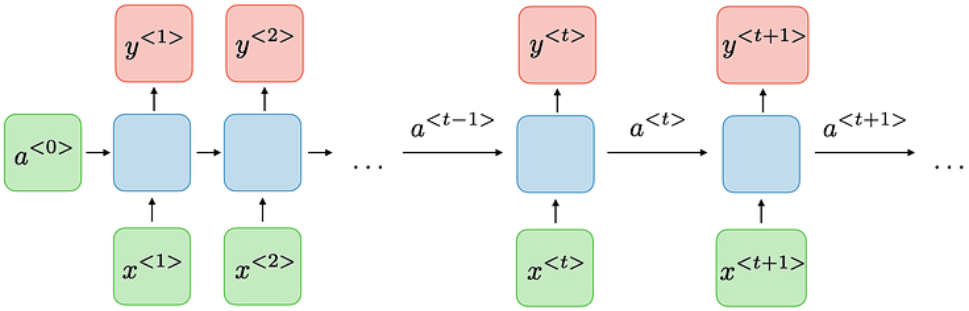

Рисунок 1.3 – Абстрактна структура RNN моделі [3]

Не дивлячись на створення даних моделей у 1990-их роках, значний розвиток дані моделі мали пізніше зі створенням додаткових технологій, які дали змогу даним моделям більш якісно аналізувати дані та формувати більш складні семантичні функції, тому аналіз даної моделі буде описаний пізніше.

Після розвитку 1980-их та 1990-их роках, світ NLP технологій мав період зниженого розвитку, через недостатню можливість аналізу текстів на основі старого представлення слів. Але дана тенденція була змінена у 2013 році зі створенням технології Word2Vec. Головною проблемою представлення слів до створення технології Word2Vec було представлення слів без урахування семантичної інформації через що використання таких методів як RNN було неефективним тому, що більш складні методи не мали можливості детально аналізувати функції, які знаходяться у текстовій інформації.

Word2Vec запропонував представляти слова у виді векторів, які були створені на основі додаткової нейронної мережі, яка буде знаходити залежності між словами та формувати багатовимірне уявлення слів, де буде зберігатися додаткове семантичне представлення інформації. Дана архітектурна одиниця стала однією з головних технологій, яка використовується у комбінації з новітніми NLP методами. Одним з методів Word2Vec є CBOW, який базується на Feed-Forward нейронній мережі, яка навчається на методі передбачення цільове на основі контексту (рисунок 1.4).

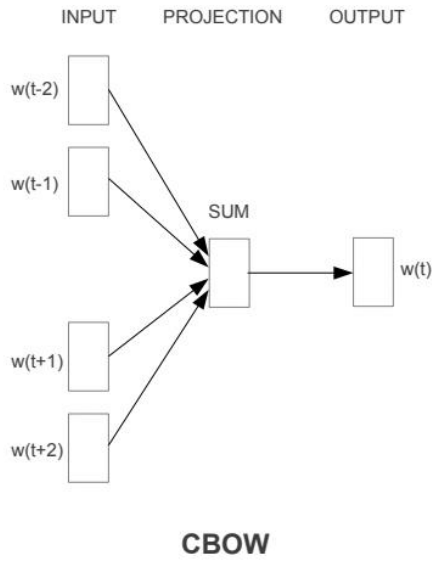

Рисунок 1.4 – Структура нейронної мережі методу CBOW [4]

Після навчання моделі метод Word2Vec перетворює слова у векторне представлення, яке містить семантичне представлення слів у багатовимірному виді (рисунок 1.5).

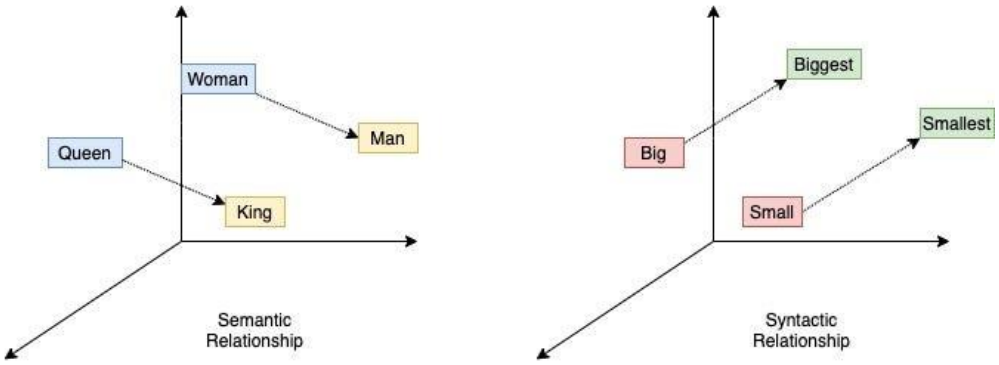

Рисунок 1.5 – Візуалізація векторів слів [4]

Дане представлення перетворює слова з базових One-hot encoding векторів, які не несуть у собі ніякої інформації до складних структур, які у числовому виді описують кожне слово, що дає змогу моделям більш чітко аналізувати слова та знаходити складні залежності між ними (рисунок 1.6).

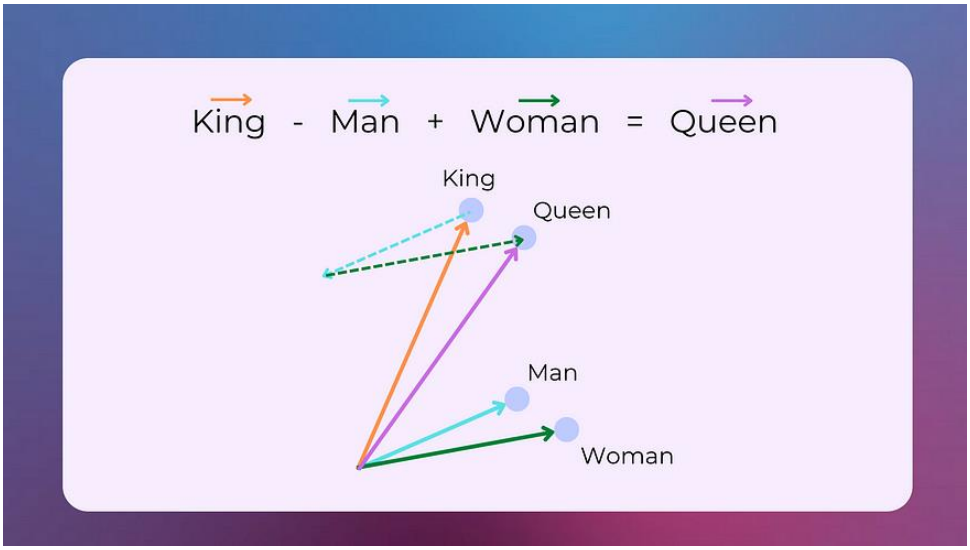

Рисунок 1.6 – Векторні операції між векторами у методі Word2Vec [5]

Прикладом зберігання семантичної інформації є взаємодія між векторами на основі базових операцій додавання та віднімання. Навіть при базовому відніманні та додаванні різноманітних векторів стало можливим отримувати осмислені результати, прикладом яких є рисунок 1.6, де показано як саме людина може перетворювати послідовність слів у іншу. Дане наповнення векторів слів семантичною інформацією стало ключовим етапом розвитку NLP методів тому, що більш складні моделі не мали можливості якісно навчатися на попередніх методах представлення текстової інформації.

Дане ускладнення представлення слів дало змогу для подальшого розвитку RNN моделей, яким не було достатньо використовувати базове представлення текстової інформації. Як було описано вище, RNN моделі на відміну від класичних Feed Forward нейронних мереж було рекурентне використання блокових структур для обробки текстової послідовності. Головною проблемою Feed Forward архітектури було використання незалежних шарів, які не мали змогу аналізувати порядок слів та контекст введеної інформації, через що навість незначні у текстовій послідовності мали значний вплив на результати обробки, що було неефективним у аналізу такої складної та динамічної структурі як текстові послідовності. У свою чергу RNN архітектура використовує рекурентний виклик блочних структур з спільними параметрами, що дає змогу моделі аналізувати як контекстуальну інформацію так і часові залежності у даних.

Але не дивлячись на значний розвиток RNN, вони мали у собі значні проблеми, через які вчені намагалися створити більш досконалі алгоритми аналізу текстової інформації. Головними проблемами RNN являються:

– послідовна природа обчислення: RNN моделі аналізують інформацію послідовно, через що процес обробки інформації витрачає значний об’єм часу;

– затухаючі та вибухові градієнти: однією з головних проблем RNN моделей є складе опрацювання градієнтів через рекурентне

використання вагів. Дана проблема значно ускладнює процес навчання через складний процес налаштування моделі;

– слабка якість аналізу довгострокових залежностей: послідовна природа аналізу послідовностей окрім значних витрат часу має у собі проблему втрати інформації у довгих послідовностях через перенавантаження блоку пам'яті у моделі, що призводить до значного погіршення якості аналізу довгих текстових послідовностей.

На основі даних проблем наукова спільнота активно створювала методи, які будуть здатні вирішити дану ситуацію. Головним проривом у даній сфері стало створення моделей типу Transformer у 2017 році. Дана архітектура стала основою для створення таких LLM моделей як ChatGPT, що стали головною рушійною силою сучасного світу (рисунок 1.7).

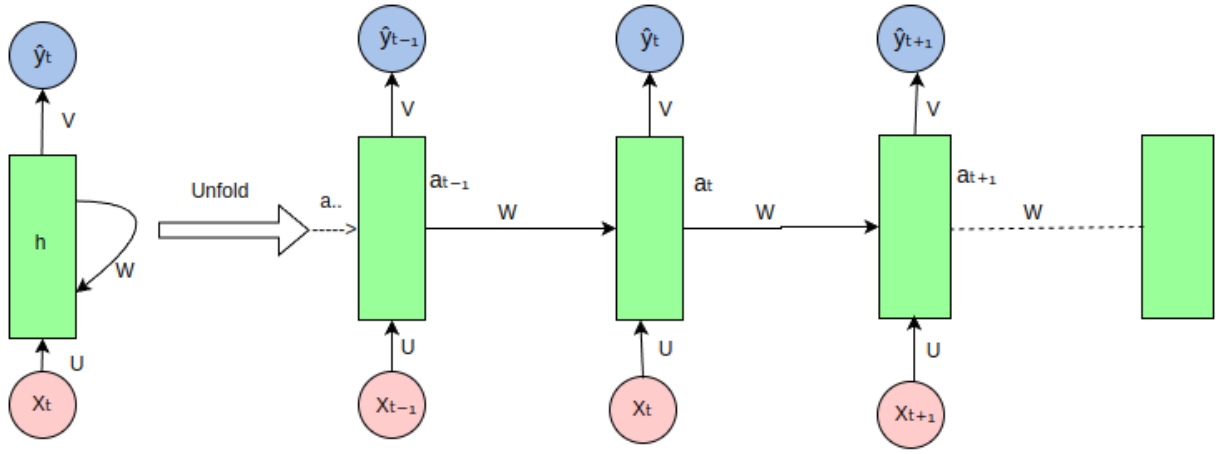

Рисунок 1.7 – Деталізована візуалізація архітектури RNN моделі [7]

1.2 Аналіз існуючих тенденцій на ринку

Одним з головних напрямків вдосконалення сучасних моделей є підтримка мультимодального аналізу та виводу моделей для створення універсальної системи, яка буде одночасно оброблювати будь-яку інформацію, яка може бути передана користувачем. Головними прикладами

підтримки даної технології є сімейство моделей CPT, Google Gemini, Llama та інші види LLM, які здатні оброблювати та аналізувати як текстові дані так і зображення. Але створення та розвиток мультимодільності має у собі такі недоліки як:

– значне ускладнення архітектури: створення моделей, які здатні оброблювати різноманітні дані призводить до значного ускладнення структури моделі та потребує створення складної структури потоку даних та їх уніфікування під час використання моделі та робить використання базових блоків багатьох моделей неможливими у використанні;

– уніфікація ембедінгів: різноманітні типи даних потребують особливих типів перетворення та різноманітної архітектури для аналізу вхідних функцій, що потребує створювати складний потік аналізу та комбінування підструктур у загальній структурі моделі;

– ускладнення масштабування: комбінування різноманітних структур призводить до значного збільшення параметрів у моделі, через що кількість параметрів даних моделей значно збільшується як через додавання нових структур для аналізу даних так і через додавання проміжних блоків для комбінування інформації між собою;

– складність навчання та аналізу результатів: створення складної мультимодальної структури призводить до значного ускладнення процесу навчання через нерівномірний процес оновлення параметрів, що може призводити до погіршення якості виконання додаткових задач. Також з даного пункту випливає ускладнення процесу оцінки якості моделей через неможливість повноцінного покриття та уніфікації всіх задач у єдину метрику;

– складність створення наборів даних для навчання: процес навчання будь-якої моделі потребує створення спеціалізованих наборів даних значних розмірів, через що створення складного мультимодального набору даних може мати значну кількість труднощів. Дана проблема базується як на складності розмітки даних та використання додаткових

інструментів, використання яких може значно погіршити якість розмітки. Друга проблема полягає у складному механізмі поєднання текстових, графічних та аудіо даних у єдиний потік.

Хоча створення складної мультимодальної системи теоретично будуть здатні ефективно аналізувати будь-які дані та якісно використовувати отриману інформацію у будь-якому сценарії, на даний момент створення однієї моделі, яка буде спроможна повноцінно оперувати великою кількістю даних є дуже складним та дорогим процесом, який буде недоступний для багатьох користувачів. Через це процес створення мультимодальної системи фокусується на створення моделей, які спроможні одночасно оперувати як текстовою інформацією так і зображеннями, через особливості архітектури, на якій базуються сучасні LLM моделі, що буде більш детально розглянуто у пункті 2. Однак навіть дана мультимодальність призводить до значного ускладнення структури моделі, наборів даних для навчання та складності у паралельному навчанні. Прикладом значної модальності є модель Perceiver IO, яка здатна оброблювати велику кількість типів даних але використання даної моделі за межами наукових досліджень є доволі складним через дані особливості:

– високі апаратні вимоги: використання даної моделі потребує значного кластера апаратного забезпечення (від 64 TPU), що призводить до неможливості у використанні малими та середніми компаніями та до великої складності у налаштуванні всієї системи;

– велика складність у налаштуванні моделі: дана модель має у собі велику кількість гіперпараметрів, через складний процес аналізу, при я кому навіть невелика зміна у значеннях призводить до критичного зниження якості аналізу запитів;

– ручне налаштування input/output-запитів: структура моделі потребую ручного налаштування механізмів введення та виведення даних з моделі для кожного з видів задач.

Мультимодальні системи мають великий потенціал для розвитку та використання у майбутньому, але на даний момент створення повноцінних систем аналізу великої кількості типів даних потребує дуже значного ускладнення потоку налаштування даних та ускладнення моделі.

1.3 Постановка задачі

1.3.1 Аналіз методів вирішення поставленого питання

Проаналізувавши сучасні тенденції у розвитку мультимодальних моделей мною було відійти від спроб створювати єдину модель, яка буде здатна оброблювати різноманітні дані та знайти нові методи комбінації та обробки мультимодальної природи даних у єдиний системі.

Для вирішення даного питання мною було обрано створити агентну систему, яка буде оперувати різноманітними моделями для створення процесу мультимодального аналізу запитів користувача. Створення агентної структури дасть можливість використовувати більш прості варіації моделей, які базово не спроможні оперувати у мультимодальному середовищі та оперувати більш складними потоками даних. Головними перевагами даного підходу є:

– модульність: модульна природа системи дає змогу окремного тестування кожного з компонентів та швидкої заміни компонентів, без значного втручання у структуру системи;

– знижена витрата ресурсів: мультимодальні системи спроможні опрацьовувати велику кількість сценаріїв у яких система буде оперувати лише частиною з наданого функціоналу та сценарії у яких система буде використовувати усі функції є рідкісними, через що модульна система не буде перевантажувати систему через обробку непотрібних аспектів.

Використання даного підходу значно розширює області використання мультимодальних систем та дає змогу створювати дані системи не лише

провідним компаніям у галузі створення нейронних мереж. На основі цього розповсюдження мультимодальних систем збільшиться, що може призвести до популяризації та збільшення попиту на дослідження та створення справжніх мультимодальних систем.

Однак не дивлячись на дані переваги агентного підходу необхідно розуміти його недоліки та складності для правильної його реалізації та майбутнього вдосконалення.

– затримка генерації: через послідовний аналіз характеристик у даних у агентних систем процес генерації фінальної відповіді може значно зростати, через що використання даного підходу у динамічних системах може створювати значні складності;

– втрата міжмодальної інформації: послідовний аналіз з використанням послідовного виклику моделей різної модальності призводить до втрати інформації, на основі якої система могла строювати більш деталізовані відповіді та якісні відповіді на запити користувачів;

– накопичення помилок: послідовний аналіз запиту користувача з лінійним викликом різноманітних структур може накопичувати помилки та передавати спотворену інформацію до головної моделі агента.

На основі даних проблем мультимодальна агента система не є кращим теоретичним методом реалізації даної задачі, але на даний момент даний підхід є кращим при аналізі складності реалізації.

1.3.2 Формування вимог до системи

Після процесу аналізу шляхів реалізації даного проекту необхідно сформувати про системи, на основі яких дана система буде створена.

– система повинна приймати від користувача текст та зображення (PNG/JPG) у єдиному запиті;

– система повинна генерувати текстові та зображення (PNG/JPG) у єдиному запиті;

– поєднання результатів обробки різних модальностей у єдиний семантичний репорт;

– агент-оркестратор на основі LangChain та LangGraph маршрутизує запити у відповідні модулі (LLM, CV) ;

– проведення семантичного та контекстного аналізу результатів різних модальностей із подальшим злиттям;

– відображення результатів аналізу у вигляді текстових пояснень та/або візуалізацій в інтерфейсі Streamlit;

– збереження та завантаження історії чат-сесій у базі MongoDB з можливістю пошуку за попередніми повідомленнями;

– збереження та поновлення контексту діалогу: при поверненні до попередньої сесії бот продовжує обробку з урахуванням історії.

1.3.3 Аналіз програмного забезпечення системи

Для реалізації даного проекту та дотримання поставлених вимог мною було обрано використовувати мову програмування Python через її значну інтеграцію у світ ШІ та провідну роль у розробці різноманітних моделей, значну підтримку агентних систем та наявності значного переліку бібліотек та фреймворків, які створені для спрощеної реалізації та налаштування даних систем та інтеграціями мови програмування Python з низькорівневими мовами як C\C++ для подальшої оптимізації. Через дані характеристики мови Python, вона є найбільш доцільною для реалізації даного проекту.

Другим ключовим фактором реалізації агентної системи є обирання фреймворку, який буде задовільнити заданим вимогам. Ключовою характеристикою даного проекту є реалізація мультимодальної інтеграції до структури проекту, дана характеристика стала ключовою у обранні фреймворку LangChain, який підтримує значну інтеграцію різноманітних типів даних, асинхронного виконання функцій та його додаткової

бібліотеки Langgraph, яка була створена для розробки розгалуженої графової структури агентів, задля реалізації динамічного потоку обробки, низькорівневої структури графових агентів, що дозволяє чітко формувати потік даних та хід виконання необхідних операції.

Задля інтеграції моделей ШІ і глибокого налаштування їх параметрів було використано екосистему Hugging Face, яка окрім надання доступу до великої кількості моделей, має у собі такі інструменти як Transformers та Diffusers, які використовуються для ініціалізації та виклику моделей пов’язаних з аналізом\генерації будь-яких типів даних, що є необхідним для реалізації мультимодальної системи.

Останньою частиною даного проекту є інтеграція баз даних, яка буде зберігати та передавати релевантну інформацію як до користувача так і для моделі для повторного її використання. Для вирішення даної проблеми мною було обрано використовувати NoSQL моделі, які базуються на JSON форматі даних. Через дані особливості така база даних як Mongodb стала одниз з найкращим варіантів реалізації зберігання та передавання даних у систему

2 ТЕОРЕТИЧНИЙ АНАЛІЗ

2.1 Трансформери як основні моделі сучасного світу

Трансформери – це один з домінуючих типів архітектури нейронних мереж, який використовується для обробки послідовних даних, найголовнішим прикладом цього являються текстові дані. Але дані моделі також використовуються для таких задач як комп’ютерний зір, розпізнавання мов та аналіз часових рядів.

Оригінальна модель трансформерів була вперше описана у 2017 році в статті «Attention is All You Need» та стала однією з ключових етапів розвитку нейронних мереж за останні роки. Оригінально запропонована як еволюція моделей на основі рекурентної нейронної мережі (RNN), що використовуються для машинного перекладу, моделі на основі трансформатора згодом досягли передових досягнень майже в усіх дисциплінах машинного навчання (ML). Хоча моделі типу трансформер можливо використовувати у різноманітних задачах, головна область використання трансформерів це чат-боти, відповіді на запитання, переклад текстів.

2.1.1 Ключові блоки

Головною особливістю моделей трансформерів є їхній механізм самоуваги, завдяки якому трансформаторні моделі отримують свою вражаючу здатність виявляти зв’язки (або залежності) між кожною частиною вхідної послідовності. На відміну від архітектури RNN і CNN, які їй передували, трансформаторна архітектура використовує лише Attention Layers та Feed-Forward layers. До появи трансформаторних моделей більшість завдань NLP покладалися на рекурентні нейронні мережі (RNN). Те, як RNN обробляє послідовні дані, за своєю суттю є серіалізованим: вони

приймають елементи вхідної послідовності по одному й у певному порядку. Це перешкоджає здатності RNN охоплювати довгострокові залежності, тобто RNN можуть ефективно обробляти лише короткі текстові послідовності. Цей недолік було певною мірою було усунено шляхом впровадження мереж довгострокової короткочасної пам’яті (LSTM), але залишається основним недоліком RNN. Механізми Attention, навпаки, можуть досліджувати всю послідовність одночасно і приймати рішення про те, як і коли зосередитися на конкретних часових кроках цієї послідовності. На додаток до значного покращення здатності розуміти довгострокові залежності, ця якість трансформерів також дозволяє виконувати завдання паралельно (рисунок 2.1).

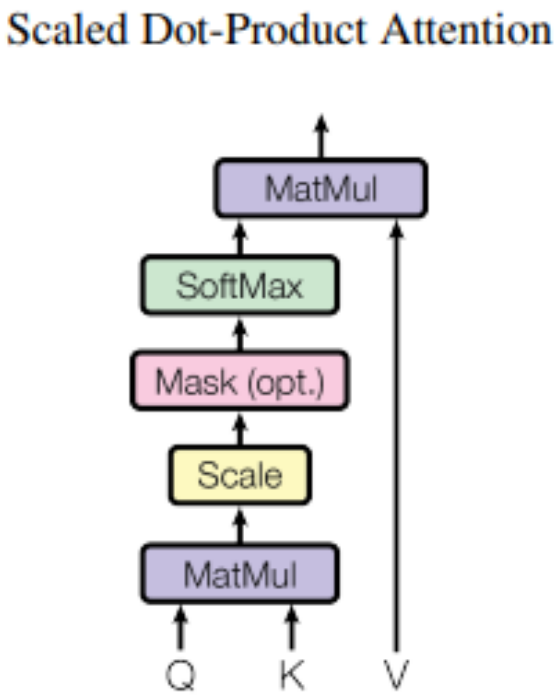

Рисунок 2.1 – Scaled Dot-Product Attention [8]

Scaled Dot-Product Attention – ключовий механізм у архітектурі трансформерів, який дозволяє моделі ефективно знаходити залежності між різноманітними елементами у переданих даних завдяки можливості аналізу контексту та семантичної інформації між частинами даних.

Scaled Dot-Product Attention базується на ідеї, що кожен елемент даних може бути пов'язаний на різні елементи у послідовності. Для цього Scaled Dot-Product Attention використовує три ключові елементи:

– query – є запитом, що формується для кожного елемента вхідної послідовності, який вимагає додаткової контекстної інформації. Він задає той контекст чи критерій, виходячи з якого визначається значимість інших елементів послідовності;

– key – ідентифікатор кожного елемента вхідної послідовності, що описує його змістовну та семантичну характеристику. Саме ключі використовуються для визначення ступеня подібності чи релегантності між елементами послідовності та запитом;

– value – змістовне заповнення елемента вхідної послідовності. Значення – це інформація, яка використовується для формування підсумкового виходу механізму уваги після того, як встановлено релевантність (через ключі та запити).

Хоча Scaled Dot-Product Attention став ключовою частиною трансформерів, проте він не є самостійною одиницею через його не навченість. Задля вирішення цієї проблеми і було створено Multi-Head Attention.

Multi-Head Attention базується на паралельному виклику декількох Attention модулів, до яких Query, Key та Value передаються з додаванням Linear блоків які є сховищем навчальних параметрів для Query, Key та Value, які будуть передані в окремий Scaled Dot-Product Attention, де будуть обчислені та поєднанні у єдиний вихідний блок, який на основі конкатенації та додаткового лінійного перетворення отримає додаткове перетворення, у якому буде зберегтися різноманітні функції та числельні представлення інформації з набору даних (рисунок 2.2).

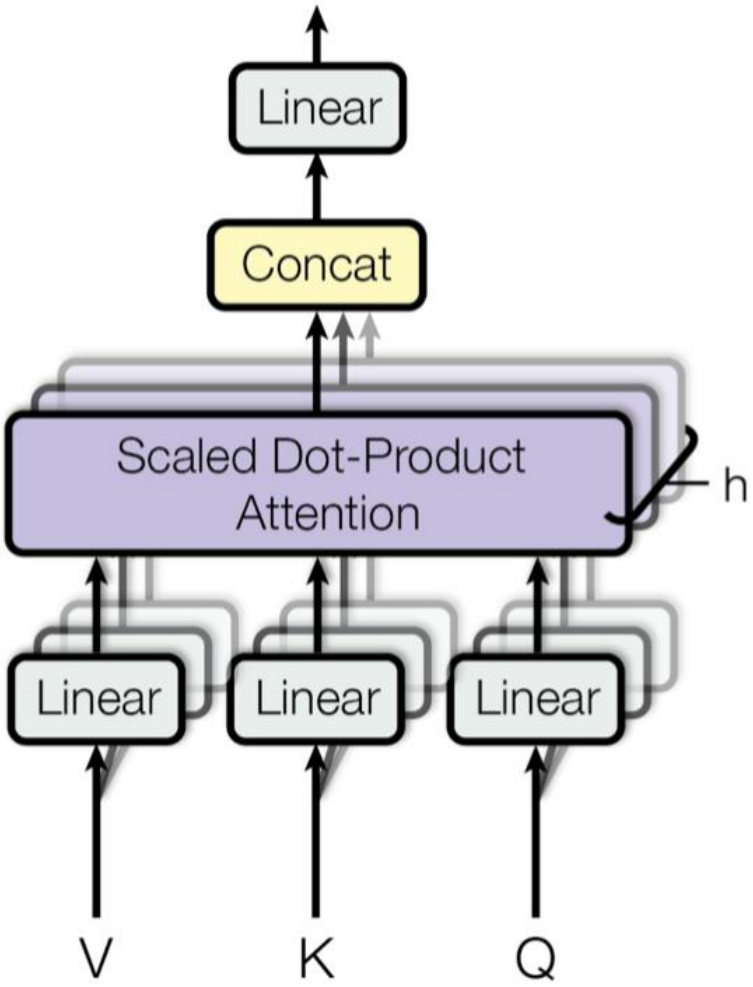

Рисунок 2.2 – Multi-Head Attention [8]

Метод перетворення вхідних векторів базується на ідеях, представлених у CNN, де кількість фільтрів задавало певну спеціалізацію для пошуку функцій. У випадку трансформерів і MHA, кожна голова в даних блоках буде відповідати за пошук різних просторових та семантичних патернах у даних, переданих моделі. Що робить дану систему більш динамічною та гнучкою у ситуаціях, коли функції можуть знаходитись у різноманітних частинах запиту користувача.

Дане формування також робить поведінку моделі більш детермінованою та зрозумілою (рисунок 2.3).

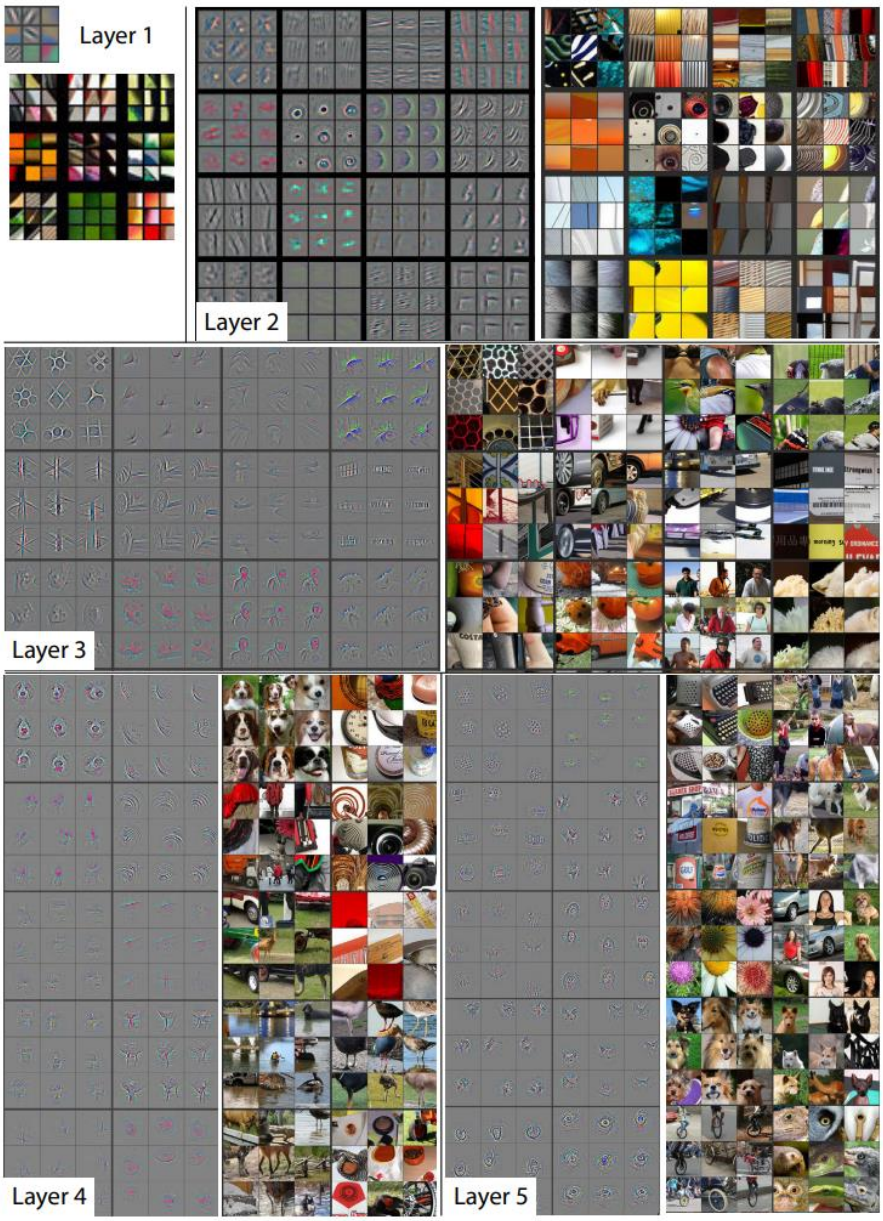

Рисунок 2.3 – Multi-Head Attention [9]

На основі інформації зі статті «Visualizing and Understanding Convolutional Networks», ми бачимо як саме відбувається навчання та змінна знайдених функцій. Хоча дана стаття описує навчання CNN, але принципи навчання будь-якої моделі мають схожі характеристики.

Першою ключовою особливістю являється ускладнення знайдених функцій та перехід від базових характеристик, для зображень це лінії, кути та кольори а для текстової інформації це положення слова у тексті до

складних образів для зображень а для текстової інформації це семантичні зв'язки між словами у тексті та контекст.

Другою ключовою особливістю являється специфікація кожного окремного фільтру для CNN або ж окремної голови у трансформерів. Це допомагає моделі уособлено шукати окремі особливості замість перенасичення вагів моделі величезною кількістю характеристик.

Також дана особливість була описана в оригінальній статті, де були показані активації різних голів на одному рівні в MHA, які підтверджують, що дана залежність, яка спершу була описана у CNN моделях, відноситься до кожного виду моделей (рисунок 2.4, 2.5).

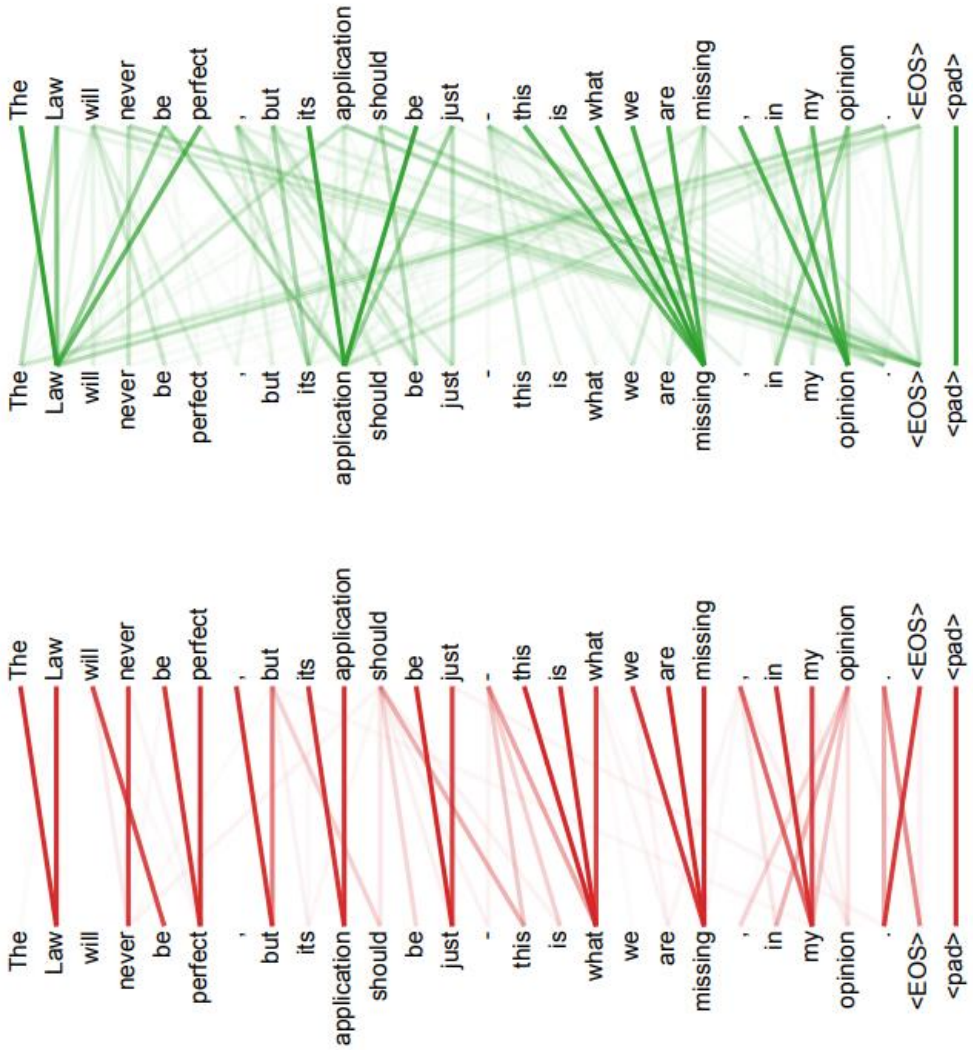

Рисунок 2.4 – Візуалізація активацій вагів на одному рівні [8]

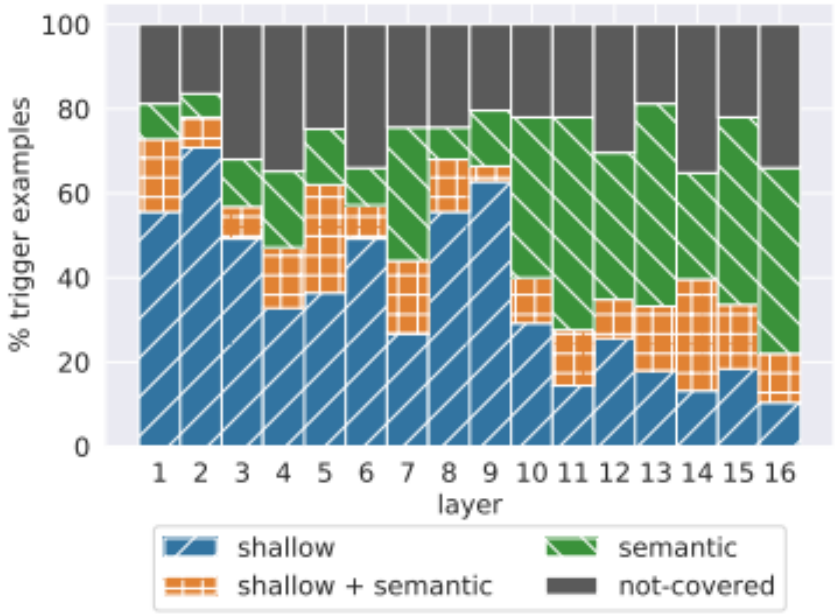

Рисунок 2.5 – Візуалізація ускладнення знайдених функцій у даних [15]

MHA окрім точного аналізу інформації має у собі ще дві ключові особливості. Першою особливістю MHA це Receptive Field, який на відміну від CNN охоплює усі дані одночасно без необхідності поступового аналізу інформації, що робить MHA схожим на Fully Connected Layer.

Однак MHA через особливості Scaled Dot-Product Attention не має проблем повно зв'язкових шарів у вигляді чіткої залежності від послідовності даних. У плані кількості параметрів MHA також має найменшу вимогу за навчальними параметрами (порівняння з CNN працює тільки для Vit і мультимодальних моделей) (рисунок 2.6).

$$PE_{(pos,i)} = \begin{cases} \sin\left(\frac{pos}{10000^{i/d_{model}}}\right) & \text{if } i \text{ is even} \\ \cos\left(\frac{pos}{10000^{(i-1)/d_{model}}}\right) & \text{if } i \text{ is odd} \end{cases}$$

Рисунок 2.6 – Positional encoding [8]

Наступним ключовим блоком архітектури трансформерів це Positional encoding. Як було сказано вище, Scaled Dot-Product Attention та MHA, що на ньому базується, не має чіткого зв'язку зі структурою текстової інформації та слів у запитах, тому Positional encoding необхідний для вирішення цієї проблеми завдяки додаванню інформації про місцезнаходження слів у послідовності даних. На основі функцій синуса та косинуса Positional encoding додає інформацію про послідовність інформації в даних, які передаються на вхід нашої моделі, даючи їй можливість зберігати всередині інформацію про положення даних у послідовності. Використовуючи періодичні функції синусів і косінусів, ми отримуємо в результаті ситуацію, коли відмінності між токенами у припущенні на близькій відстані будуть значними, а відмінності між далекими мінімальними (рисунок 2.7).

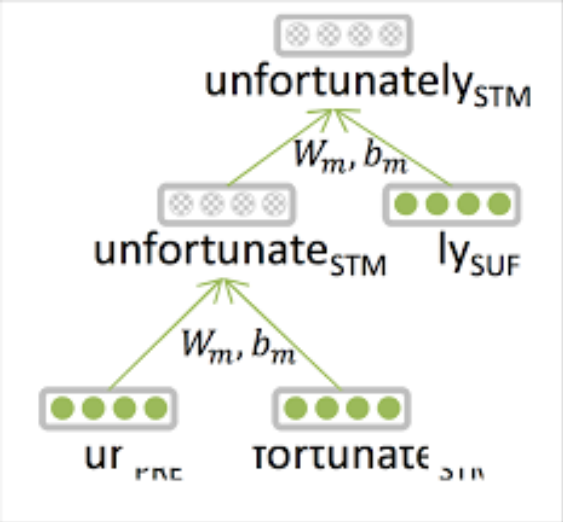

Рисунок 2.7 – Byte pair encoding [10]

Останньою частиною базових блоків є Byte pair encoding, хоч це не є блоком архітектури нейронної мережі, проте цей метод є так само невід'ємною частиною як оригінального трансформера, так і всіх наступних мовних моделей. В галузі обробки природної мови BPE отримав широке застосування для сегментації тексту на підслів (subwords) і став одним із

стандартних підходів токенізації в сучасних нейромережевих моделях (наприклад, GPT, GPT-2, GPT-3, BERT і т.д.). Головна ідея алгоритму BPE полягає в послідовному об'єднанні пар символів, що найбільш часто зустрічаються, або підпослідовностей символів у даних в єдині токени. Це дозволяє алгоритму ефективно кодувати текст, балансуючи між рівнем символів (занадто багато токенів) та рівнем слів (занадто великий словник і занадто багато рідких токенів)

2.1.2 Архитектура

Після аналізу кожного з ключових компонентів моделі та розумінню кожного з аспектів цієї моделі необхідно перейти до аналізу суцільної структури моделі та поглибленню інформації, яка була описана в попередньому пункті. Головною особливістю оригінальної моделі трансформерів це наявність двох ключових компонентів: Encoder and Decoder (рисунок 2.8).

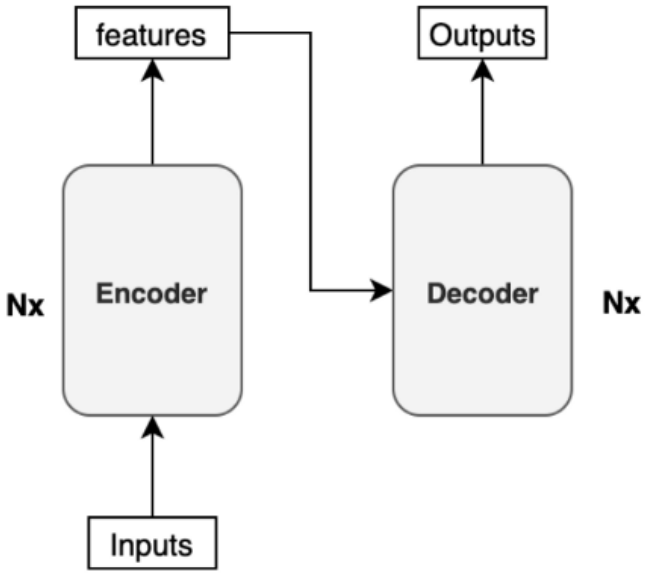

Рисунок 2.8 – Абстрактна візуалізація моделі Transformer [16]

Першою ключовою структурою в даній архітектурі є Encoder, який використовується для отримання функцій та характеристик із поданого користувачем тексту. Encoder так як і Decoder представляє собою послідовний виклик однотипних структурних блоків, які будуть виконувати свої необхідні функції. Як було сказано вище Encoder це структура, яка відповідає за отримання функцій з переданих даних (рисунок 2.9).

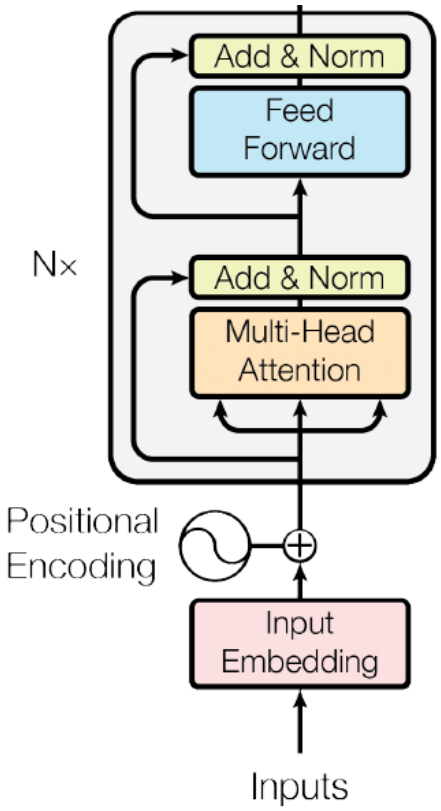

Рисунок 2.9 – Encoder блок [8]

Структура блоку Encoder складається з перерахованих раніше одиниць і додаткової структури Feed Forward layers, яка складається з базових FC layers, проте вона має під собою одні з найголовніших периметрів у моделі, тому він буде детальніше розказано далі.

Першою особливістю є використання Self Attention MHA, в якій Query, Key та Value базуються на одній матриці, яка була передана користувачем на вхід до моделі та у подальшому буде оброблена у MHA.

Другою важливою частиною архітектури є використання Residual connections, яке було представлено спочатку в CNN Resnet для аналізу зображень, проте стало стандартом для всіх нейронних мереж у майбутньому (рисунок 2.10).

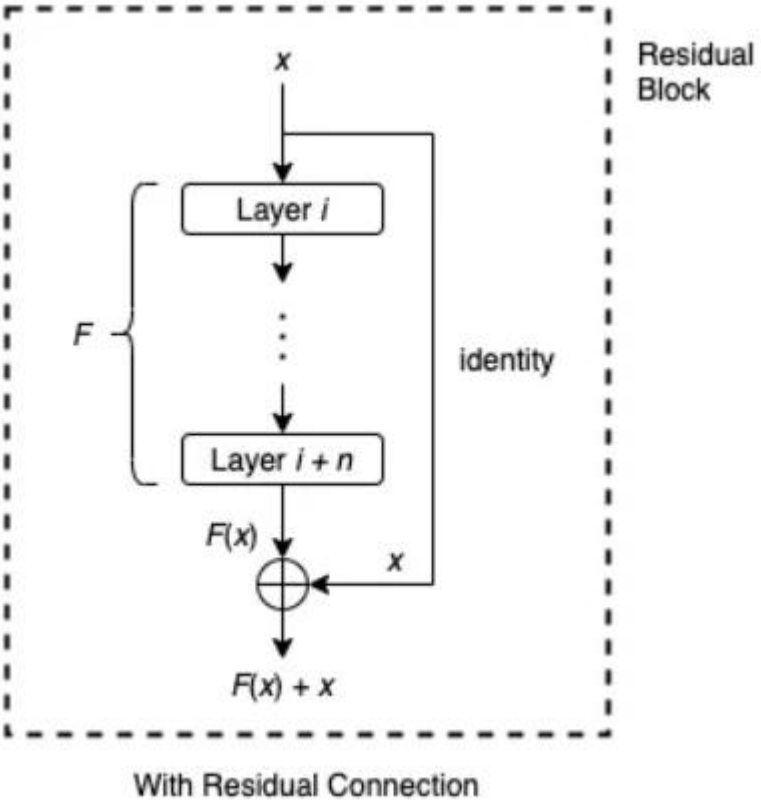

Рисунок 2.10 – Residual connections [17]

Residual connections це техніка, яка дозволила будувати значно глибші моделі, що раніше було недоступне через різні проблеми з градієнтами.

Головна властивість Residual connections – це функція Identity, яка зберігає інформацію між шарами, і при виникненні критичним проблем з градієнтами зберігає інформацію між шарами, дозволяючи продовжувати навчання та відновлювати градієнти та значення параметрів моделі (рисунок 2.11).

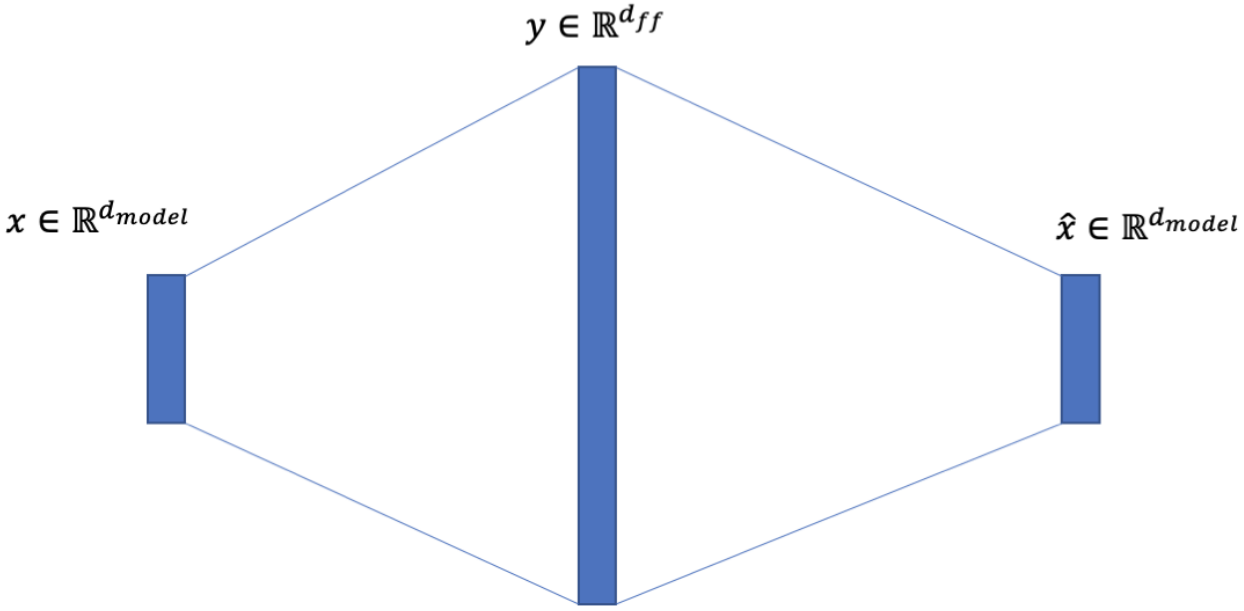

Рисунок 2.11 – Структура Feed Forward layers у блоку Encoder [15]

Наступною особливістю Encoder блоку є структура Feed Forward layers, яка є продовженням ідей, які були представлені у Scaled Dot-Product Attention. Для спрощення формула розрахунків FFL буде розписана без використання bias, однак і при його включенні закономірність зберігається, це було зроблено лише для спрощеного розуміння

$$FC_2 = FC_1(XW_1^T)W_2^T, \tag{2.1}$$

де, X – це матриця значень;

W_1^T – це ваги шару 1;

W_2^T – це ваги шару 2.

Ця структура повністю повторює структуру Scaled Dot-Product Attention, де X виконує роль запитів, W_1^T виконує роль ключів, а W_2^T виконує роль значень. Ми можемо сказати, що це пам'ять з механізмом ключ значення, яка стане ключовим механізмом аналізу структури та семантики інформації, яка передається до моделі. У статті Transformer Feed-Forward

Layers Are Key-Value Memories було проведено аналіз висновків Feed-Forward Layers у трансформері та було виявлено, як саме Feed-Forward Layers реагують на різні введення користувача (рисунок 2.12).

Key	Pattern	Example trigger prefixes
$\mathbf{k}^{1}_{449}$	Ends with *"substitutes"* (shallow)	*At the meeting, Elton said that "for artistic reasons there could be no substitutes* *In German service, they were used as substitutes* *Two weeks later, he came off the substitutes*
$\mathbf{k}^{6}_{2546}$	Military, ends with *"base"/"bases"* (shallow + semantic)	*On 1 April the SRSG authorised the SADF to leave their bases* *Aircraft from all four carriers attacked the Australian base* *Bombers flying missions to Rabaul and other Japanese bases*
$\mathbf{k}^{10}_{2997}$	a "part of" relation (semantic)	*In June 2012 she was named as one of the team that competed* *He was also a part of the Indian delegation* *Toy Story is also among the top ten in the BFI list of the 50 films you should*
$\mathbf{k}^{13}_{2989}$	Ends with a time range (semantic)	*Worldwide, most tornadoes occur in the late afternoon, between 3 pm and 7* *Weekend tolls are in effect from 7:00 pm Friday until* *The building is open to the public seven days a week, from 11:00 am to*
$\mathbf{k}^{16}_{1935}$	TV shows (semantic)	*Time shifting viewing added 57 percent to the episode's* *The first season set that the episode was included in was as part of the* *From the original NBC daytime version , archived*

Рисунок 2.12 – Аналіз тригерів у різних шарах Feed-Forward Layers [15]

У даному дослідженні було виявлено, що саме Feed-Forward Layers модель знаходити як просторову, так і семантичну зв'язок слів у реченні так наприклад у другому прикладі на зображенні х було виявлено, що 6 Feed-Forward шар знаходить одновірно як просторове розташування слів base\bases так і семантичне дані військових баз. На основі цієї візуалізації ми бачимо, що у структурних блоках моделі transformer, MHA відповідає лише за перетворення інформації, але не спроможні аналізувати саму інформацію з переданого масиву даних.

Також на даному зображенні видно, як із поглибленні моделі семантичні характеристики стають більш значними та роль просторового розташування знижується. Так на першому шарі інформація, яку модель оброблює базується лише на інформації про розташування слів у послідовності. Шостий шар вже має можливості оброблювати як просторову інформацію так і семантичну, а наступні шари вже відходять від

аналізу просторових характеристик і повністю базуються на семантичних характеристиках даних (рисунок 2.13).

Як було описано у пункті 1.1.1, при поглиблені моделі функції ускладняються, та переходять від базової просторової інформації до семантичної, як й у моделях CNN. Саме завдяки Feed-Forward Layers, моделі виду transformer оброблюють функції наявні у даних. Ця інформація є необхідною для подальшого розвитку моделей виду transformer та створенню Encoder та Decoder based models, через розуміння навчання та потоку інформації у моделі.

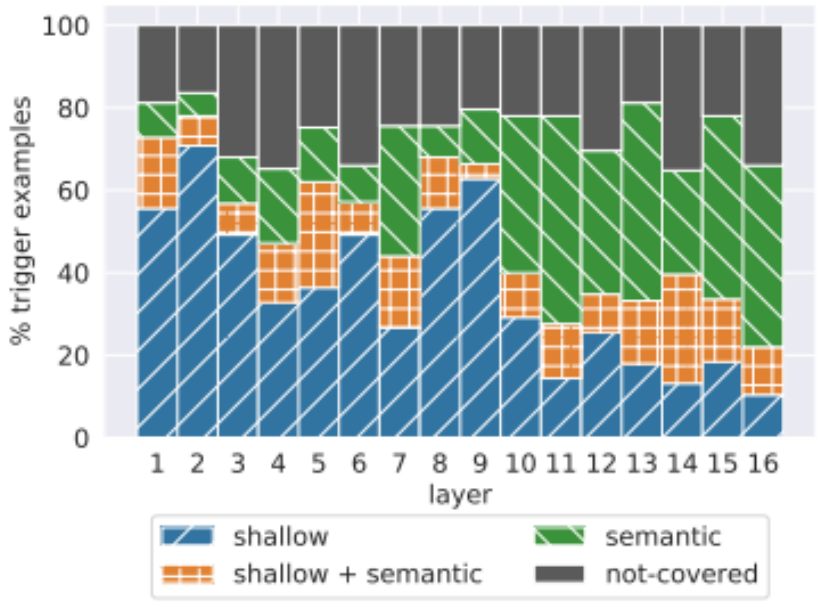

Рисунок 2.13 – Відображення залежності функцій від глибини моделі [15]

Однак важливо розуміти, як саме відбувається аналіз тригерів та вихідної інформації з шарів. Це стало можливе завдяки Residual connections, які знижують коливання у градієнтах та роблять навчання більш стабільним.

Дана стабільність не лише дає змогу поглиблювати моделі без втрати якості а й аналізувати будь-які моделі, де використовуються Residual connections (рисунок 2.14).

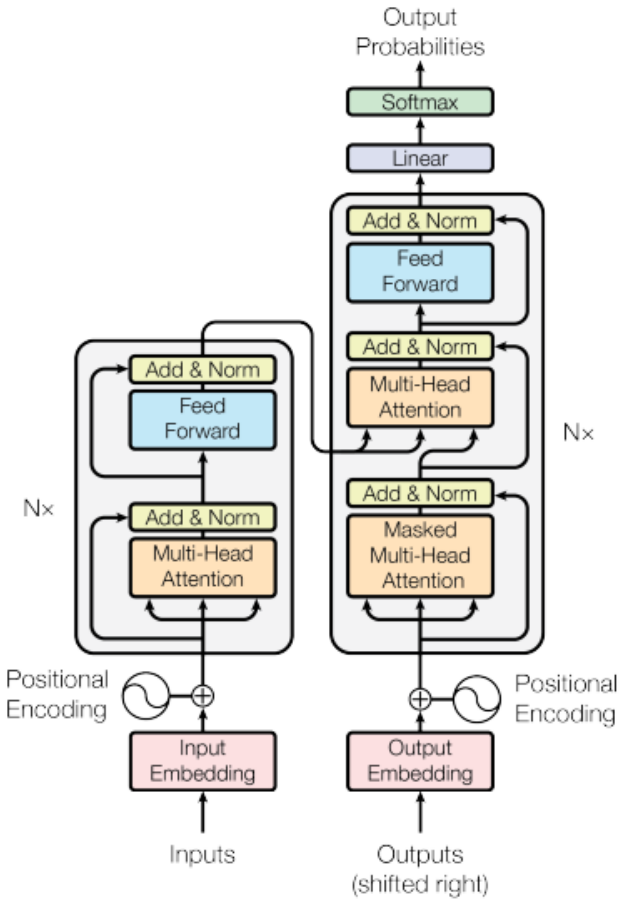

Рисунок 2.14 – Повна архітектура моделі Transformer [8]

Другою ключовою частиною архітектури є Decoder, який на відміну від Encoder використовується для генерації текстової послідовності на основі функцій переданих с Encoder.

Не дивлячись на те, що Transformer має здатність одночасно аналізувати послідовність тексту одночасно, але через те що Decoder як й RNN моделі спроможні генерувати лише один токен за виклик, базова структура MHA не може бути використана для структури Decoder.

Для вирішення цієї проблеми була розроблена Masked MHA. Цей блок приховує інформацію про майбутні слова, які модель ще не згенерувала і дозволяє як навчати модель, так і генерувати правильні відповіді у реальних завданнях (рисунок 2.15).

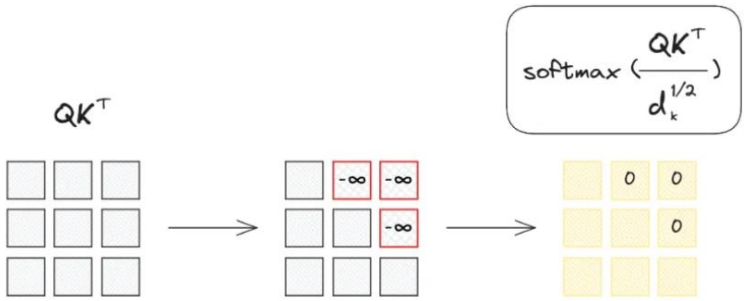

Рисунок 2.15 – Masked MHA [18]

Останнім ключовим пунктом архітектури Decoder це передача інформації з Encoder до другого блоку MHA. В даній реалізації Query це вихідна інформація з Masked MHA а Key і Value це інформація, яка була проаналізована завдяки структурі Encoder.

2.1.3 LLM quantization

Після становлення LLM моделей на базі трансформерів головною архітектурою на сьогоднішній день з’явилася потреба в їх подальшому вивченню та вдосконаленню. Головною ідеєю в покрашенні моделей стало зростання кількості вагів, ускладнення аналітичної якості й наближенню до здібностей людей.

Але даний процес зустрів одну з ключових проблем, а саме неконтрольоване збільшення ресурсів, які необхідні для навчання та взаємодії з системами, через що використання останніх моделей потребує величезних обчислювальних ресурсів та використання їх у звичайних проектах стає неможливим без використання додаткових технологій.

Саме для вирішення даної проблеми була розроблена технологія квантування, яка вирішує проблеми з надмірною розмірністю моделей (рисунок 2.16).

Головна проблема в прямому навчанні та використанні моделі полягає у надмірному використанні обчислювальних ресурсів та у надмірному

використанні апаратного забезпечення. На прикладі моделі, яка має 7 мільярдів параметрів використання пам’яті сягає 112GB, це означає, що навіть для невеликих для сучасних реалій моделей не є можливим використовувати навіть топове апаратне забезпечення у єдиному екземплярі тому, що розмірності сучасних моделей можуть сягати понад декілька мільярдів параметрів. Поки тенденція збільшення параметрів буде домінувати при створенні нових систем, не буде можливим використовувати моделі у повноцінному вигляді (рисунок 2.17).

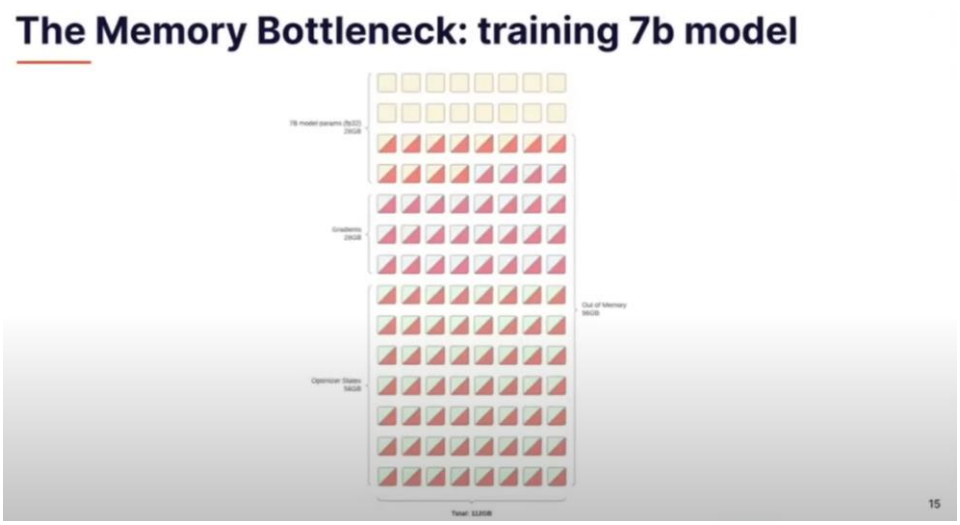

Рисунок 2.16 – Використання пам’яті при навчанні моделі [19]

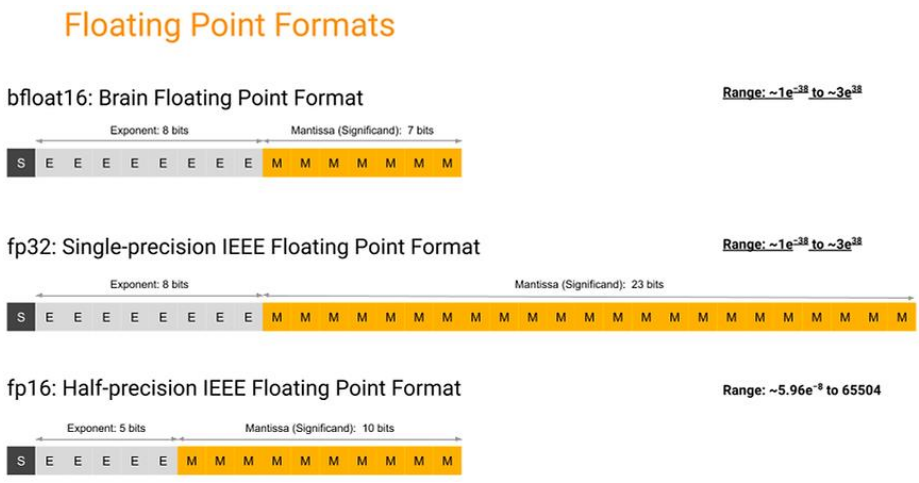

Рисунок 2.17 – Використання пам’яті різними типами даних [21]

Суть квантування полягає у знаходженні проекції даних більш складних типів до необхідних, меншої розмірності. Одним з головних прикладів методів квантування є affine quantization, метод який є одним з головних методів квантування на даний момент. Він полягає у відображенні неперервного діапазону чисел до дискретного набору значень (рисунок 2.18).

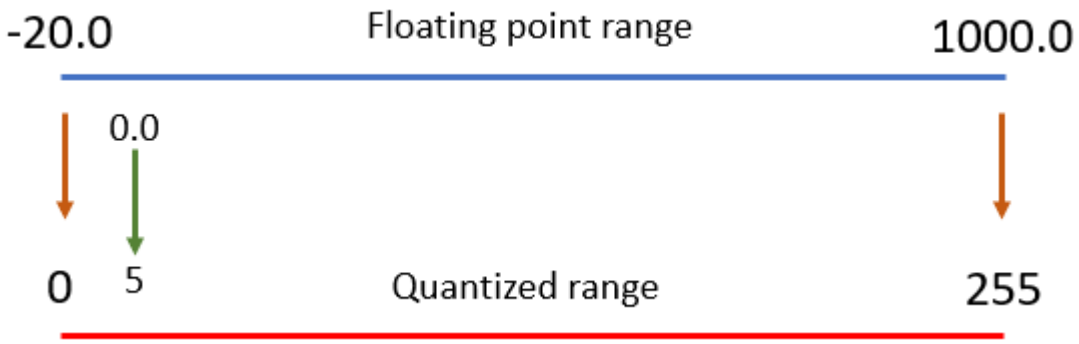

Рисунок 2.18 – Приклад роботи квантування [22]

Головними значеннями у affine quantization є значення кроку квантування, на основі якого ми будемо фільтрувати наші значення до дискретного представлення та числа, яке описує 0 у та необхідне для уникнення змішень при деквантувані. Дане перетворення дає змогу швидко та ефективно перетворювати дані з неперервного відображення до дискретного, й навпаки з мінімальними втратами інформації.

2.1.4 LoRA

Не дивлячись на високу якість LLM та їх універсальність, використання моделей без попереднього налаштування не зможе повністю використовувати їх можливості повністю. Для даних цілей використовувався метод Fine tuning, при якому відбувається до навчання вже навченої моделі (рисунок 2.19).

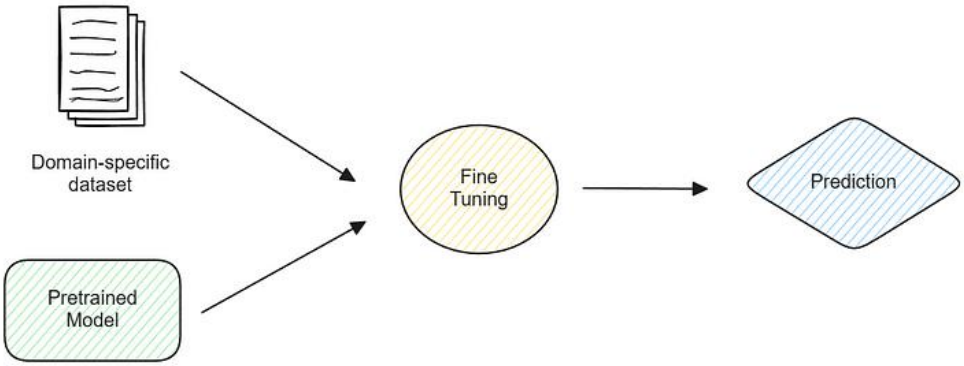

Рисунок 2.19 – Fine tuning [26]

Але даний метод Fine tuning при використанні з LLM не міг показувати якісні результати. Проблема в класичному Fine tuning полягає в переписуванні параметрів при спробі використовувати його використання, через що модель втрачає інформації з попередніх етапів навчання та погіршує свої показники не лише для обраної задачі, але й при базовому аналізі текстової інформації. Але через різноманіття доменів використання LLM моделей багаторазовий процес Fine tuning став необхідністю. Задля вирішення цієї проблеми була розроблена LoRA (Low-Rank Adaptation), яка вирішує дану проблему.

Першим кроком для реалізації LoRA є використання методики Low Rank matrix, тобто видалення з оригінальної матриці лінійно залежних стовпців і перетворення в 2 матриці, перша з яких зберігає лінійно незалежні стовпці, а друга зберігає репрезентацію стовпців через лінійно незалежні (рисунок 2.20).

Використання Low Rank matrix дає змогу оптимізувати та пришвидшити метод навчання й знизити використання пам’яті під час Fine tuning оскільки зберігається менше параметрів для оновлення вихідних ваг (з 9 у вихідній матриці до 3 у матриці A + 3 у матриці B).

Головна особливість використання LoRA є повна замороження параметрів головної моделі та додавання до вже існуючих параметрів

новостворених на навчених для специфічної задачі. Це використовується для збереження інформації, яка була отримана під час головного потоку навчання та додаванню нової інформації з мінімальним додаванню нових параметрів (рисунок 2.21, 2.22).

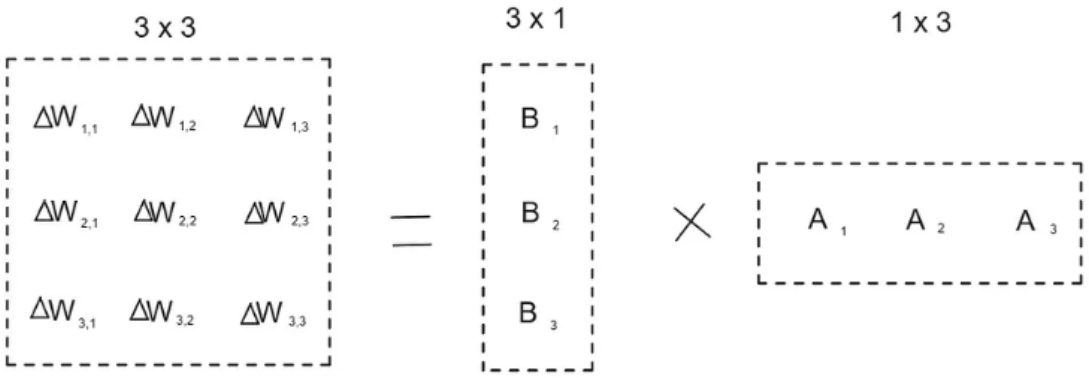

Рисунок 2.20 – Low Rank matrix [26]

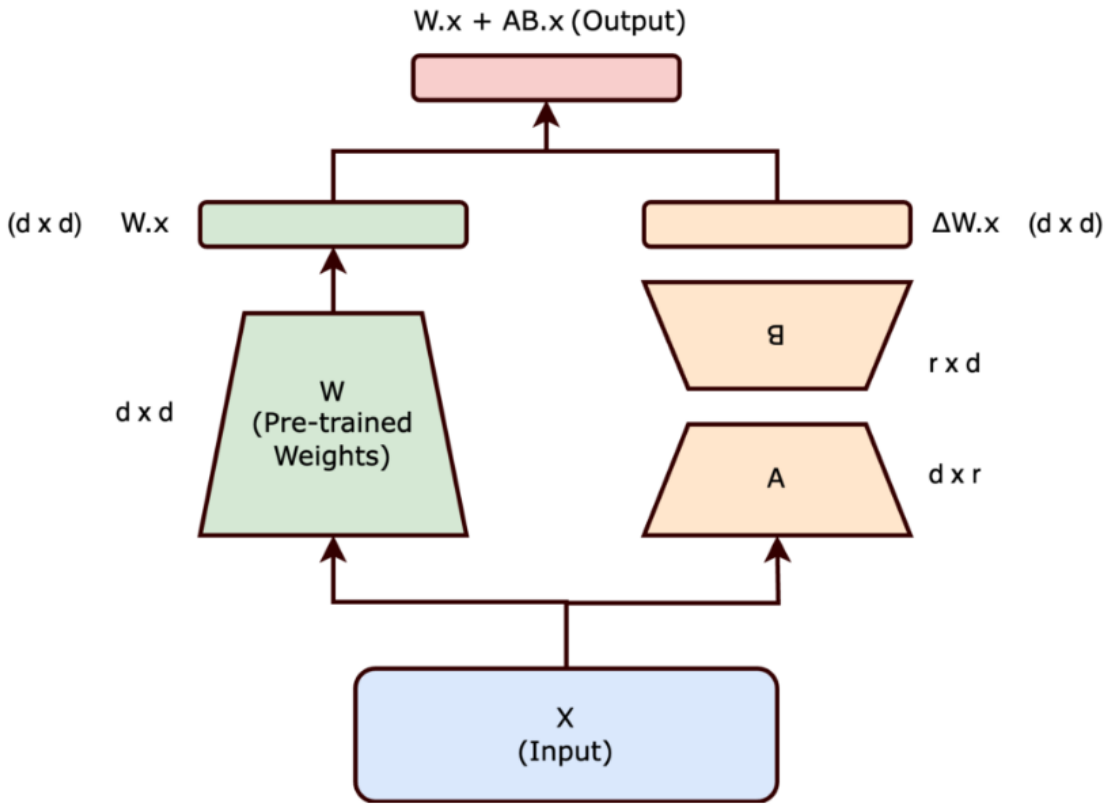

Рисунок 2.21 – Low-Rank Adaptation [27]

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} ($Adpt^D$)*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} ($Adpt^D$)*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
RoB_{large} (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
RoB_{large} (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
RoB_{large} ($Adpt^P$)†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
RoB_{large} ($Adpt^P$)†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
RoB_{large} ($Adpt^H$)†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
RoB_{large} ($Adpt^H$)†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
RoB_{large} (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

Model & Method	# Trainable Parameters	E2E NLG Challenge BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M ($Adapter^L$)*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M ($Adapter^L$)*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M ($Adapter^H$)	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L ($Adapter^L$)	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
GPT-2 L ($Adapter^L$)	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Рисунок 2.22 – Порівняння результатів різних методів Fine tuning [26]

Дане дослідження «дослідження» показує, що LoRA є конкурентним методом Fine tuning, який як зберігає точність моделі й гарно адоптує її до нових задач, так і потребує найменшу кількість навчальних параметрів, завдяки чому швидкість навчання є найбільшою з усіх методів, що зробило LoRA головним методом у сучасному світі.

2.2 ChatGPT

Наступним ключовим пунктом теоретичного дослідження даної дипломної роботи є аналіз конкретних моделей, які були використані у дипломному проекті. Першою моделлю, яка стала сполучною ланкою всього проекту є ChatGPT та його версія 4o mini. ChatGPT – це сімейство LLM, яке стало найбільш популярним у сучасному світі.

Одним з ключових аспектів моделей сімейства ChatGPT є Decoder-only архітектура, яка упускає з оригінальної структури Enoder елемент і фокусується лише на використанні Decoder блоків для послідовної генерації тексту на основі запиту користувача та вбудованих інструкцій. Хоча в оригінальній моделі був опис структури, де використовується як Encoder так і Decoder, але в подальшому розвитку архітектури використовуються моделі, які базуються на єдиному ключовому блоку.

Не дивлячись на те, що базова архітектура, яка була представлена в оригінальній статті, має найбільшу ефективність та універсальність, але дані моделі мають ключові проблеми, через які їх використання стало недоцільним на даний момент:

– подвійний розмір моделі: для реалізації повної структури Transformer розмірності блоків Encoder та Decoder є однакові, через витрати пам’яті на виконання задач збільшується у два рази;

– сповільнення та витрати на обчислення: через послідовну структуру викликів блоків часові витрати на опрацювання кожного запиту значно збільшується з додатковими витратами пам’яті на зберігання вихідної інформації з блоку Encoder;

– ускладнена реалізація та налагодження: через наявність подвійної структури реалізувати складну структуру потоку даних з необхідністю більш складних наборів даних, через різну структуру блоків;

На основі даних особливостей, використання моделей, які базуються на єдиному блоці архітектури є більш доцільним та простим, тому що повноцінна архітектура є надмірною для виконання більшості задач. У випадку моделей сімейства ChatGPT, використання Decoder-only архітектури є достатнім тому, що саме Decoder блок використовується для послідовної генерації інформації і не потребує повного аналізу інформації для послідовної генерації тексту.

Другою ключовою особливістю є RLHF (Reinforcement Learning from Human Feedback), який й допоміг створити ChatGPT3 та дати значне

поліпшення якості генерації тексту. RLHF – це метод при якому модель LLM налаштовується для генерації людино подібного тексту на основі Reinforcement Learning.

Перший крок даного метода базується на створенні набору ранжованих відповідей моделей та створенню reward model, яка буде оцінювати майбутні відповіді моделі. Другий крок є навчання основної моделі завдяки додатковій моделі оцінки. У результаті RLHF перетворює завдання «написати хороше продовження тексту» із суто ймовірнісної (максимізувати лог-правдоподібність) на завдання «генерувати відповіді, максимально близькі до людських ідеалів», формалізованим через навчену reward model. Це і є його суть – перевести суб'єктивні людські уподобання до формального оптимізаційного критерію.

2.3 Llama3

Наступною ключовою моделлю є Llama 3.2 Vision, яка реалізовує мультимодальну структуру для одночасного аналізу як тестової інформації так і зображень. Але для аналізу є важливим лише додаткова структура, яка відповідає для аналізу зображень та налагодження потоку даних у цій версії моделі тому, що базова структура базується на моделі LLaMA 3.1, яка базується на ідеях, які були описані у пункті 2.2.

Головна відмінність структури LLaMA Vision – це наявність Vision Encoder та Integration Mechanism, який збіднює інформацію з блоку Vision Encoder зображення в звичайну Decoder-base модель, яка використовується для аналізу та генерації тексту (рисунок 2.23).

Не дивлячись на те, що в звичайних генеративних моделях використання Encoder блоку є надмірним, але для вирішення проблеми с аналізом зображень необхідно використовувати Encoder-base структуру VIT.

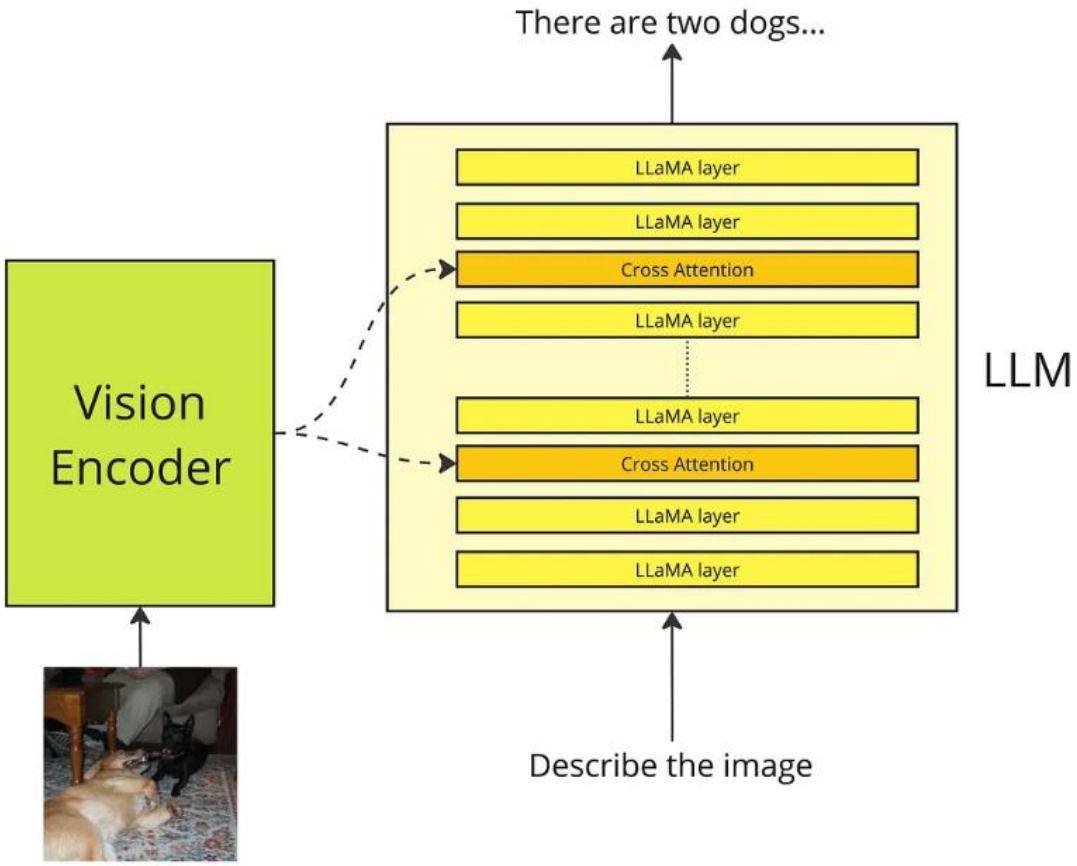

Рисунок 2.23 – Базова структура LLaMA Vision [32]

Першим кроком обробки зображення є етап розбиття фото на патчі по 32×32 пікселі, які після цього перетворюються на вектора розмірності 1280 й передаються до VIT Encoder-base моделі, яка перетворює зображення на набір функцій (рисунок 2.24).

Першою особливістю даної реалізації є збереження проміжних результатів. Дана особливість має схожість з Feature Pyramid Network, яка використовується у image detection. Як було описано у пункті 2.1.1, більш поглиблені шари аналізують більш семантично складну інформацію та може втрачати інформацію про базові параметри такі як: кольори, кути, положення та інше. Дана втрата інформації не є суттєвою при базовому аналізі зображень, але при текстовому описані втрата базових характеристик зображень буде призводити до втрати якості текстової генерації.

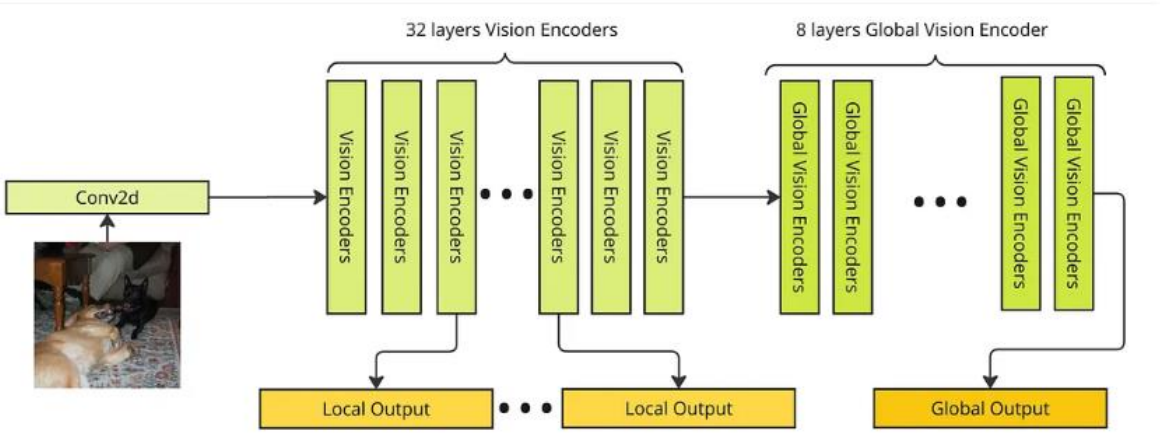

Рисунок 2.24 – Структура VIT Encoder [32]

Другою особливістю реалізації є реалізація потоку інформації зі VIT моделі до головного потоку. Для реалізації даного потоку були використані додаткові блоки адаптери, інтегровані до базової структури моделі на основі послідовного виклику декількох блоків Cross Attention, де текстова репрезентація інформації поєднувалася з функціями, отриманих зі зображення (рисунок 2.25).

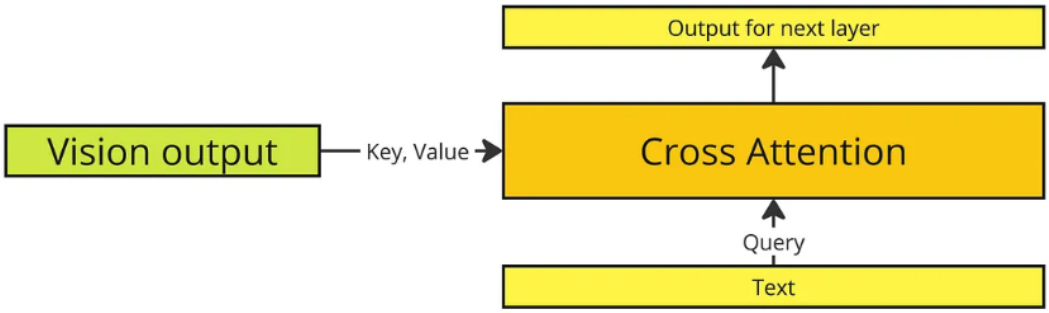

Рисунок 2.25 – Потік даних у LLaMA Vision [32]

Дана передача візуальної інформації відбувається кожні 5 шарів для додавання до представлення моделі більш складних і семантично насичених функцій, які VIT модель отримувала з зображення на різних рівнях семантичного аналізу.

Використання даної структури додавання інформації має такі переваги

– прогресивне уточнення: модель поступово отримує інформацію зі зображення, завдяки чому модель спроможна оброблювати функції одного рівня;

– повторне використання функцій: Модель може ефективно повторно використовувати візуальні характеристики, що проектуються, на різних рівнях Cross Attention, зберігаючи обчислювальну ефективність.

2.4 Аналіз агентів та їх застосування

Останньою ключовою частиною дипломного проекту є використання агентів для поєднання та автоматизації процесів у ході роботи моделі. Хоча LLM є складними та багатофункціональними алгоритмами, вони все одно мають свої обмеження, задля вирішення яких були створені агенти. LLM є закритою системою, яка спроможна генерувати складні послідовності, але вона не спроможна динамічна отримувати інформацію з навколишнього світу та актуалізуватися під час використання без додаткового складного процесу навчання.

Агенти були реалізовані на основі методу Chain-of-Thought, яка додавала до запиту користувача приклади проміжних міркувань, на основі яких модель ділила передані користувачем запити на проміжні міркування, задля покращення якості відповідей. Дана ідея стала основою для створення майбутніх агентів, які базуються на додаванні до ділення запиту на підзадачі додаткові інструменти, які модель зможе використовувати у довільному порядку (рисунок 2.26).

Наступним кроком вдосконалення Chain-of-Thought було додавання інструментів, які збільшують функціонал оригінальної моделі. Одним з головних прикладів агентів є ReAct. Даний агент базується на циклічній генерації наступних кроків:

– thought: крок на якому модель генерує внутрішнє міркування для наступних дій чи генерації фінальної відповіді користувачу;

– action: крок на якому модель обирає один з наявних інструментів та передає до нього необхідну інформацію;

– observation: крок на якому модель перевіряє результати попереднього кроку починає цикл заново (рисунок 2.27).

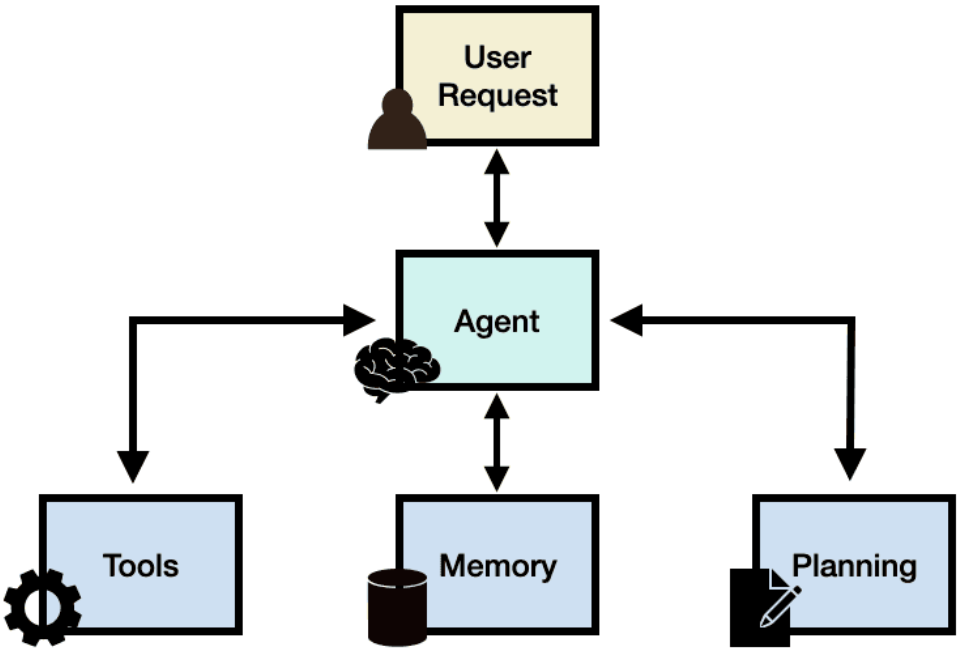

Рисунок 2.26 – Абстрактна структура агентів [37]

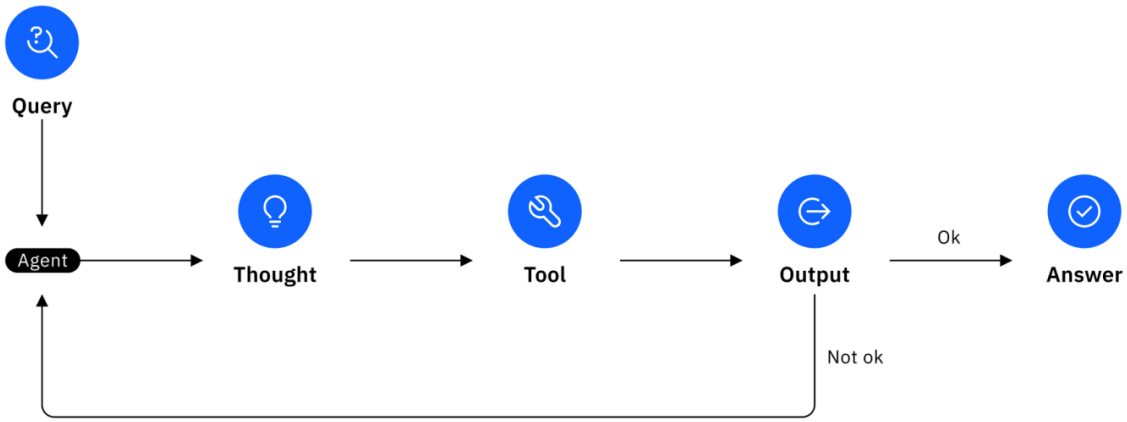

Рисунок 2.27 – Візуалізація потоку ReAct агента [37]

Використання ReAct агентів, як і інших видів має такі переваги:

– універсальність: використання агентів дає змогу LLM моделям використовувати різноманітні інструменти та виконувати різноманітні команди чітко розподіляючи послідовність дій;

– адаптивність: структура агентів не базується на самих інструментах, через що додавання агентів або ж зміна структури інструментів не потребує налаштування моделі;

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Ініціалізація додаткових моделей

Однією з головних частин розробки програмного застосунку є ініціалізація та налаштування моделей нейронних мереж для реалізації майбутніх інструментів.

Першою з ключових моделей ініціалізація та створення середи для використання моделі Llama-3.2-11B-Vision-Instruct. Дана модель є меншою варіацією моделі Llama Vision-Instruct. Причиною використання зменшеної версії є обмеження обчислювальних ресурсів. Не дивлячись на зменшену розмірність моделі, після детального тестування модель показала як високу якість генерації тексту так і високу якість обробки зображень та їх аналіз. Порівнюючи з іншими методами, даний алгоритм показує як високу якість так і доволі швидку генерації відповідей (рисунок 3.1).

```
bnb_config= BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16
)
```

Рисунок 3.1 – Ініціалізація квантування

Першим кроком для ініціалізації моделі є створення конфігурації квантування, яка буде базуватись на 4 бітному квантуванні з перетворенням даних до Float16. Використання звичайного Float16 не є кращою, головним типом даних для деквантування є Bfloat16, який можливо використовувати лише з спеціалізованими відеокартами по типу NVIDIA A100.

Для даного проекту використання Float16 є найкращою альтернативою через зберігання швидкості, потреби пам’яті та якості відповідей (рисунок 3.2).

```
template=PromptTemplate(
            input_variables=["query"],
            template="""<|begin_of_text|><|start_header_id|>system<|end_header_id|>
You are a useful assistant who should analyze images based on the user's request.
<|eot_id|><|start_header_id|>user<|end_header_id|>
User request:
{query}

User images
<|image|>

<|eot_id|><|start_header_id|>assistant_answer_final<|end_header_id|>"""
        )
```

Рисунок 3.2 – Конфігурація запиту моделі

Другою частиною ініціалізації моделі у дипломному проекті є створення конфігурації запиту моделі. Головною особливістю використання моделі Llama Vision-Instruct є специфікація запиту з додаванням спеціалізованого токену <|image|>, який необхідний для коректної обробки зображень та текстової інформації. Використання даної структури є необхідним для стабільного потоку даних між головною структурою Decoder-base моделі та VIT Encoder надбудови (рисунок 3.3).

```
class CustomLLM(LLM):
    model_name: str = Field(default="meta-llama/Llama-3.2-11B-Vision-Instruct")
    temperature: float = Field(default=0.7)
    max_tokens: int = Field(default=650)
    top_k: int = Field(default=50)
    top_p: float = Field(default=0.95)

    _processor: Any = PrivateAttr()
    _model: Any = PrivateAttr()

    def __init__(self, model_name: str = "meta-llama/Llama-3.2-11B-Vision-Instruct", device: str = "cuda:1", **kwargs):
        super().__init__(**kwargs)
        self._processor = AutoProcessor.from_pretrained(model_name, device_map=device, trust_remote_code=True, cache_dir=cache_dir)
        self._model = AutoModelForPreTraining.from_pretrained(model_name, device_map=device, trust_remote_code=True, quantization_config=bnb_config, cache_dir=cache_dir)
```

Рисунок 3.3 – Ініціалізація базових параметрів моделі

Після створення додаткових конфігурації, іде створення додаткового класу, який буде опрацьовувати запити користувача. На відміну від моделі ChatGPT-4o-mini та Stable-Diffusion, які можливо використовувати на основі додаткових фреймворків. Модель Llama Vision-Instruct не має на даний момент повноцінної підтримки через стрімкий розвиток та мале представлення мультимодільних моделей.

Для вирішення даної ситуації мною було обрано використовувати бібліотеку Pydantic, яка використовується для створення спеціалізованих класів та роботи з структурованими даними (рисунок 3.4).

```
def _call(self, prompt: str, stop: Optional[List[str]] = None, image: Optional[Image.Image] = None, **kwargs) -> str:
    inputs = self._processor(images=image, text=prompt, return_tensors="pt", padding=True) if image else self._processor(text=prompt, return_tensors="pt", padding=True)
    inputs = inputs.to(self._model.device)

    outputs = self._model.generate(
        **inputs,
        max_new_tokens=self.max_tokens,
        temperature=self.temperature,
        top_k=self.top_k,
        top_p=self.top_p
    )
    response_text = self._processor.decode(outputs[0], skip_special_tokens=True)
    final_output=self.parse_assistant_response(response_text)
    gc.collect()
    torch.cuda.empty_cache()
    return final_output
def parse_assistant_response(self,full_response: str) -> str:
    keyword = "assistant_answer_final"
    keyword_index = full_response.find(keyword)
    if keyword_index != -1:
        assistant_text = full_response[keyword_index + len(keyword):].strip()
        return assistant_text
    else:
        return full_response
```

Рисунок 3.4 – Створення функцій генерації та парсингу відповідей

Останнім кроком розробки є створення функцій для генерації та парсингу відповідей, для уникнення виводу системних аспектів моделі. Головною частиною обробки є підготовка даних на основі спеціалізованого процесору, який оброблює наявність зображень та аналізує структуру текстової інформації для стабільної обробки та виведення даних до користувача через візуальний додаток (рисунок 3.5).

```python
import torch
from diffusers import StableDiffusionPipeline
from transformers import BitsAndBytesConfig
import sys
import random
cache_dir = "D:/Diplom/New folder/models"
device = "cuda:0" if torch.cuda.is_available() else "cpu"
bnb_config = BitsAndBytesConfig(
    load_in_8bit=True,
)
def load_model(model_id):
    model = StableDiffusionPipeline.from_pretrained(
        model_id,
        torch_dtype=torch.float16,
        cache_dir= cache_dir,
    )
    model.to(device)
    return model
model_id="stabilityai/stable-diffusion-2-1"
model = load_model(model_id)
```

Рисунок 3.5 – Ініціалізація моделі Stable-Diffusion

Другою ключовою моделлю є ініціалізація моделі Stable-Diffusion, яка буде використовуватись для генерації зображень у агентній системі. Ініціалізація даної моделі базується на бібліотеці diffusers, яка є продовженням екосистемі Hugging Face, яка стала ключовою одним з ключових фреймворків для роботи зі великими нейронними мережами, через що дана модель не потребую такого тонкого налаштування як для моделі Llama Vision-Instruct.

3.2 Створення ключового агента

Другим ключовим кроком для розробки системи дипломного проекту є створення на налаштування агента, який буде базуватися на основі моделі ChatGPT-4o-mini, через використання Openai API. Не дивлячись на те, що

модель Llama Vision-Instruct є моделлю, на основі якої можливо побудувати структуру агентної системи, але обмеження обчислювальних ресурсів робить процес використання моделі Llama Vision-Instruct доволі складним, через наступні особливості.

– витрати пам’яті відеоадаптерів: створення агентної системи на основі Llama потребує створення додаткового екземпляру моделі, що є значно більш витратним ніж використання Openai API;

– якість обробки запитів користувача: для якісної обробки запитів користувачів, є необхідним використання моделей з кількістю параметрів, які наближаються до 100 мільярдів;

– швидкість роботи: опрацювання запитів користувачів потребує багаторазовий виклик агентної моделі, через що швидкість опрацювання запитів на основі локальної моделі буде займати занадто велику кількість часу.

3.2.1 Створення інструментів агентної системи

Головний аспект кожної агентної системи є інструменти, якими головна модель буде оперувати у ході аналізу запитів користувача. Даний дипломний проект базується на 3 головних інструментах (рисунок 3.6).

```
@tool
def generate_photo(prompt: str):
    """
    Call when you need to generate an image.
    Can generate only one image per call.
    """
    image = photo_model(prompt, num_inference_steps=150).images[0]
    return image
```

Рисунок 3.6 – Інструмент генерації зображень

Першим зі інструментів, якими оперує агентна система є агент, який викликає модель аналізу зображень. Структура кожного з інструментів для агентної системи базується як на основній функції так і на додаткових інструкцій, які знаходяться у спеціалізованому коментарі, який додається у серединну агентної функції. Даний коментар описує головній моделі, що виконує модель, як її використовувати, як передавати дані та може нести у собі інші додаткові коментарі, які будуть описувати особливості кожного з інструментів у системі (рисунок 3.7).

```python
@tool
def image_analysis(prompt:str):
    """
    Call when you need to analyze images sent by the user.
    You must pass the user's request and the path to the image that was sent to you.
    Query format in the tool:
        query: {query}
        image_path: {image_path}
    """
    data = json.loads(prompt)
    query=template.format(query=data["query"])
    answer=model._call(prompt=query,image=Image.open(data["image_path"]))
    return answer
```

Рисунок 3.7 – Інструмент аналізу зображень

Наступним інструментом у системі є інструмент для аналізу зображень, які користувач може передавати у систему. Головна особливість у використанні даного інструменту є автоматичне створення JSON формату, який базується на опису інструменту.

Дана особливість використання LLM моделей є ключовим фактором використання у агентних системах. Модель повинна не лише аналізувати запит користувача та аналізувати його а й виконувати перетворення даних у системі для обробки різноманітних функцій таких як створення додаткових

запитів у необхідному форматі до швидкого та правильного аналізу різноманітних форматів, які можуть використовуватись у різноманітних API інструментах (рисунок 3.8).

```python
@tool
def history(prompt:str):
    """
    This tool returns the last 3 pairs of the user's query/your answer.
    You should use it to get more context in the conversation with the user or the user can refer to their
    If the user's request is for images, use only the descriptions provided in the story.
    """
    documents = collection.find(
        {"role": {"$in": ["user", "assistant"]}, "content": {"$exists": True}}
    ).sort("_id", -1).limit(6)
    formatted_output = "Last 3 pairs user request/your answer\n"
    user_query = None
    for doc in reversed(documents.to_list()):
        if doc["role"] == "user":
            user_query = doc["content"]
        elif doc["role"] == "assistant" and user_query:
            formatted_output += f"User query: {user_query}\nYou answer: {doc['content']}\n\n"
            user_query = None
    return formatted_output
```

Рисунок 3.8 – Інструмент передачі історії

Останнім зі інструментів, які є у системі це інструмент для передачі історії чату з користувачем. Дана реалізація була необхідна для того, щоб модель не передавала непотрібну історичну інформацію з чату користувача, через можливу некоректну поведінку додаткових інструментів через перенасичення запиту додатковими даними, які не відносяться до запиту. Особливо дана реалізація була необхідна для коректної роботи інструменту генерації зображень, через знижену якість обробки текстової інформації.

3.2.2 Створення структури агента

Після створення та тестування інструментів агентної системи необхідно створити загальну структуру агентної системи. Дана структура

буде базуватись на бібліотеці Langgraph, яка є частиною фреймворку LangChain, який базується на роботі зі LLM моделями, а LangGraph конкретно використовується для створення агентів нового покоління на основі графової репрезентації, що дає більшу гнучкість, стабільність та можливість створювати складніші системи ефективніше (рисунок 3.9).

```
class AgentState(TypedDict):
    input: str
    agent_outcome: Union[AgentAction,AgentFinish,None]
    intermediate_steps: Annotated[list[tuple[AgentAction,BaseMessage]],operator.add]
```

Рисунок 3.9 – Створення структури потоку даних

Першим кроком для створення агентів є структуризація потоку даних у системі. У даному дипломному проекті потік даних складається з вхідної інформації, у якій зберігається як запит користувача так і проміжна аналітична інформація. Другою частиною є зберігання типу дії у агенті, де зберігаються кроки які модель може здійснити. Останньою частиною є зберігання проміжних кроків системі, дана структура необхідна через можливість зберігання інформації, яка буде потрібна для майбутніх кроків (рисунок 3.10).

```
def run_agent_reasoning(state:AgentState):
    agent_outcome=ReAct_agent.invoke(state,verbose=True)
    return {"agent_outcome":agent_outcome}

tool_executor=ToolExecutor(tools)

def execute_tools(state:AgentState):
    agent_action = state["agent_outcome"]
    output=tool_executor.invoke(agent_action)
    return {"intermediate_steps":[(agent_action,output)]}
```

Рисунок 3.10 – Формування вузлів у мережі

Наступним кроком є створення головних вузлів агента, які будуть формувати поведінку LLM моделі. Вузлова система агента базується на вузлі run_agent_reasoning, який буде аналізувати інформацію з стану моделі та формувати спеціалізовані запити для майбутнього виклику інструментів. Другим вузлом є execute_tools, який буде отримувати інформацію з попереднього вузла та викликати необхідні інструменти формувати структурну одинцю, яка буде додана до проміжних кроків виконання системи (рисунок 3.11).

```
AGENT_REASON="agent_reason"
ACT="act"

def sould_cont(state:AgentState):
    if isinstance(state["agent_outcome"],AgentFinish):
        return END
    else:
        return ACT

flow=StateGraph(AgentState)
flow.add_node(AGENT_REASON,run_agent_reasoning)
flow.set_entry_point(AGENT_REASON)

flow.add_node(ACT,execute_tools)
flow.add_conditional_edges(
    AGENT_REASON,
    sould_cont,
)
flow.add_edge(ACT,AGENT_REASON)

agent=flow.compile()
```

Рисунок 3.11 – Формування графа агента

Останнім кроком розробки агента є створення графа, який буде контролювати поведінку системи та описувати її. Графова структура дає можливість робити більш гнучкі системи, які можуть виконувати дії не лишу у лінійному виді, як було до створення додаткової бібліотеки LangChain. Особливість графової структури полягає у створені вузлів та

умовних зав’язків, де модель аналізує вихідну інформацію з моделі та зупиняє чи продовжує опрацювання запиту користувача (рисунок 3.12).

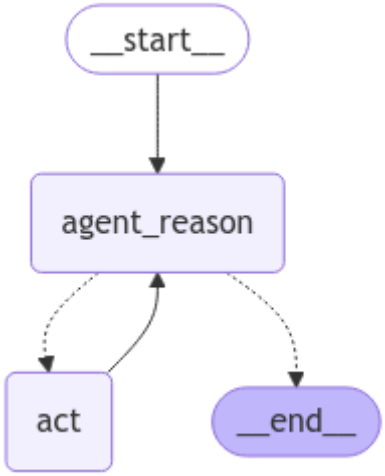

Рисунок 3.12 – Візуалізація структури агента

Структура ReAct агента базується на покроковому виконанні операцій з аналізом вихідної інформації зі інструментів, де модель перевіряє чи можливо з створити фінальну відповідь на запит користувача. ReAct хоч і є одним з найпопулярніших типів агентів на сьогоднішній день, його використання може бути неефективним у ситуаціях де є жорстке обмеження у викликах моделей а також при опрацюванні дуже складних запитів за наявністю великої кількості додаткових інструментів. У даному випадку буде більш доцільним використання Plan-based agent, який попередньо створює план виконання запиту. Але дана система може бути неефективна при роботі зі системами, які являються доволі динамічними.

3.3 Створення додаткової оболонки

Останнім кроком створення дипломного проекту є створення візуальної оболонки, яка буде поєднувати серверну частину та взаємодію з

користувачем. Візуальна оболонка даного проекту базується на streamlit, бібліотеці для швидкого створення візуальної оболонки проектів, які основані на нейронних мережах (рисунок 3.13).

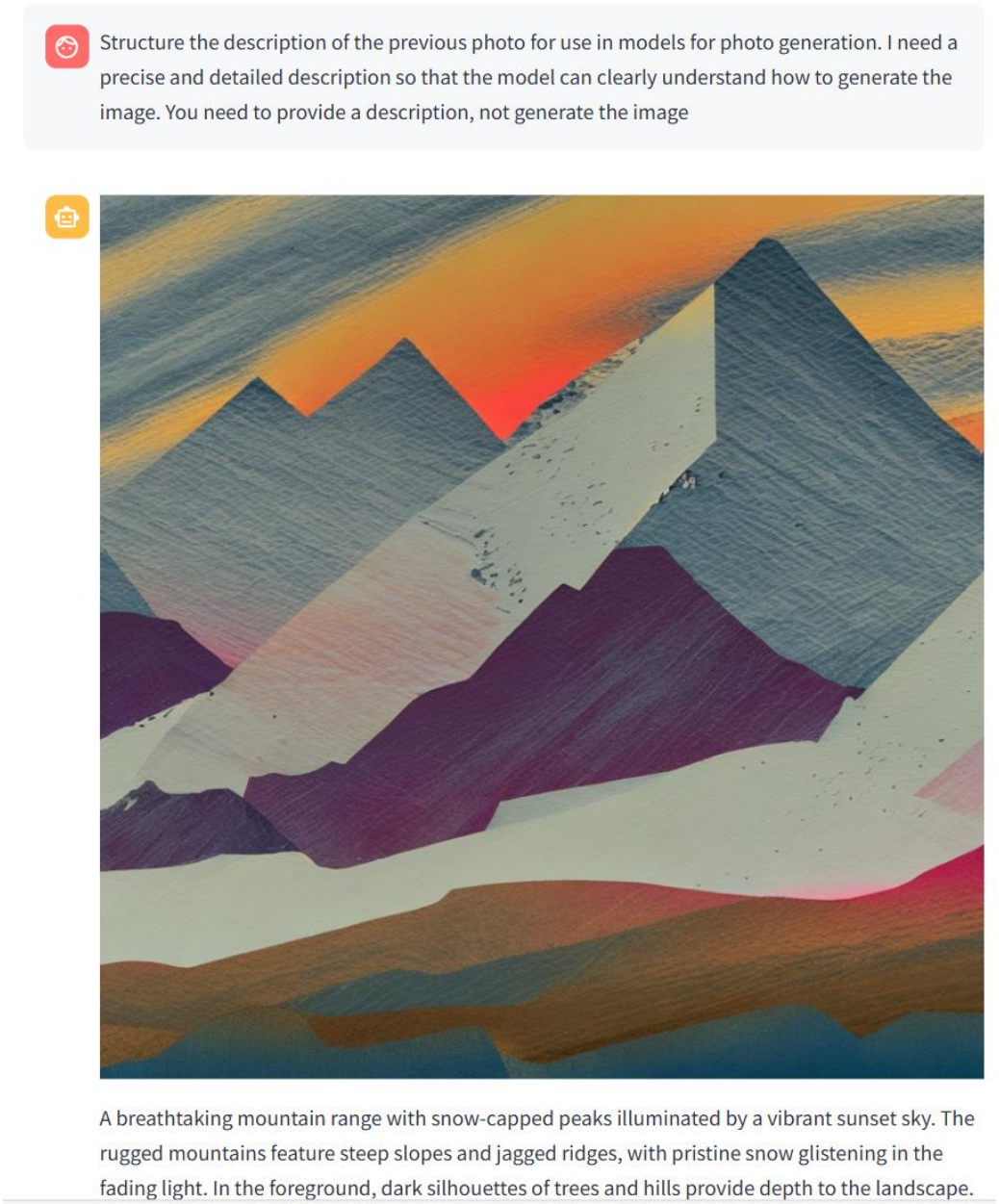

Рисунок 3.13 – Приклад візуального відображення дипломного проекту

Останнім кроком для створення системи є налаштування бази даних, яка буде зберігати чат з користувачем та передавати її як у графічну

оболонку моделі, так і у інструмент, який буде обробляти історію у агентній системі (рисунок 3.14).

```
def save_message(role, content=None, image=None):
    message = {"role": role, "content": content, "timestamp": datetime.utcnow()}
    if image:
        buffered = BytesIO()
        image.save(buffered, format="PNG")
        img_str = base64.b64encode(buffered.getvalue()).decode("utf-8")
        message["image"] = img_str
    collection.insert_one(message)


def fetch_chat_history(max_records=10):
    messages = list(collection.find().sort("timestamp", -1).limit(max_records * 2))
    messages.reverse()
    history = "\n".join([f"{msg['role']}: {msg['content']}" for msg in messages if 'content' in msg])
    return history
```

Рисунок 3.14 – Налаштування бази даних

Для виконання даного дипломного проекту мною було обрано використовувати NoSQL базу даних Mongodb, яка не має у собі чітку структуру даних. Даний тип баз даний є більш доцільним тому, що як запити користувача так і відповіді асистента мають динамічну структуру та їх опис у класичному реляційному стилі є складним та не таким ефективним ніж використання NoSQL баз даних JSON формату.

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи було досягнуто мету розробки мультимодальної агентної системи для створення єдиної екосистеми для обробки та генерації інформації. У даній кваліфікаційній роботі було описано повний цикл розробки агентної системи, починаючи з аналізу предметної області та аналізом теоретичних особливостей, які стали основою даних застосунків, до програмної реалізації, яка була створена на основі інформації на основі попередніх пунктів на базі Python з використанням LangChain, LangGraph та екосистеми Hugging Face. У рамках проекту реалізований механізм прийому та одночасного аналізу текстових, графічних інтерфейсів реалізований за допомогою Streamlit, а збереження історії чат-сесій та логів обробки забезпечено через MongoDB.

Одним з головних етапів створення застосунку був аналіз предметної галузі та сучасної ситуації у розвитку мультимодальних. Аналіз існуючих тенденцій та варіантів реалізації показав переваги та недоліки різноманітних підходів у створенні мультимодальних екосистем та їх інтеграції у сучасні застосунки.

Таким чином, розроблена мультимодальна система для автономного аналізу та генерації відповідає сучасним вимогам та потребам користувачів, забезпечує швидкий процес аналізу та генерації інформації з спрощеним методом передачі даних між різними модальностями. Це робить даний інструмент гарним прикладом реалізації мультимодальних систем та автоматизації їх роботи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) Estimating the strength of soil stabilized with cement and lime at optimal compaction using ensemble-based multiple machine learning. Scientific Reports. *Nature*. URL: https://www.nature.com/articles/s41598-024-66295-4 (date of access: 09.06.2025).

2) Support Vector Machines (SVM) In Machine Learning Made Simple & How To Tutorial. *Spot Intelligence*. URL: https://spotintelligence.com/2024/05/06/support-vector-machines-svm/ (date of access: 09.06.2025).

3) Singh R. Recurrent Neural Network (RNN). *Medium*. URL: https://medium.com/@RobuRishabh/recurrent-neural-network-rnn-8412b9abd755 (date of access: 19.05.2025).

4) Efficient Estimation of Word Representations in Vector Space. *arXiv.org*. URL: https://arxiv.org/abs/1301.3781 (date of access: 19.05.2025).

5) Chiusano F. A Brief Timeline of NLP. *Medium*. URL: https://medium.com/nlplanet/a-brief-timeline-of-nlp-bc45b640f07d (date of access: 19.05.2025).

6) Vatsal. Word2Vec Explained. *Towards Data Science*. URL: https://towardsdatascience.com/word2vec-explained-49c52b4ccb71/ (date of access: 19.05.2025).

7) Poudel S. Recurrent Neural Network (RNN) Architecture Explained. *Medium*. URL: https://medium.com/@poudelsushmita878/recurrent-neural-network-rnn-architecture-explained-1d69560541ef (date of access: 19.05.2025).

8) Attention Is All You Need. *arXiv.org*. URL: https://arxiv.org/abs/1706.03762 (date of access: 19.05.2025).

9) Visualizing and Understanding Convolutional Networks. *arXiv.org*. URL: https://arxiv.org/abs/1311.2901 (date of access: 19.05.2025).

10) Transformer Neural Networks: A Step-by-Step Breakdown | *Built In*. *Built In*. URL: https://builtin.com/artificial-intelligence/transformer-neural-network (date of access: 19.05.2025).

11) Amanatullah. Transformer Architecture explained. *Medium*. URL: https://medium.com/@amanatulla1606/transformer-architecture-explained-2c49e2257b4c (date of access: 19.05.2025).

12) Stryker C., Bergmann D. What is a Transformer Model? | *IBM*. *IBM - United States*. URL: https://www.ibm.com/think/topics/transformer-model (date of access: 19.05.2025).

13) Understanding Scaled Dot-Product Attention in Transformer Models. *Medium*. URL: https://medium.com/@saraswatp/understanding-scaled-dot-product-attention-in-transformer-models-5fe02b0f150c (date of access: 19.05.2025).

14) Deep Residual Learning for Image Recognition. *arXiv.org*. URL: https://arxiv.org/abs/1512.03385 (date of access: 19.05.2025).

15) Transformer Feed-Forward Layers Are Key-Value Memories. *arXiv.org*. URL: https://arxiv.org/abs/2012.14913 (date of access: 19.05.2025).

16) François K. Deep dive in embeddings. *Medium*. URL: https://medium.com/neoxia/deep-dive-in-embeddings-888d4acc1d0a (date of access: 09.06.2025).

17) Wong W. What is Residual Connection. *Medium*. URL: https://medium.com/data-science/what-is-residual-connection-efb07cab0d55 (date of access: 09.06.2025).

18) Deep Learning Series 20 :- Masked Multi Head Attention. *Medium*. URL: https://medium.com/@yashwanths_29644/deep-learning-series-20-masked-multi-head-attention-8b364d01032b (date of access: 09.06.2025).

19) Clark B. What is Quantization? | *IBM*. *IBM - United States*. URL: https://www.ibm.com/think/topics/quantization (date of access: 19.05.2025).

20) Devanand N. What is Quantization in LLM. *Medium*. URL: https://medium.com/@techresearchspace/what-is-quantization-in-llm-01ba61968a51 (date of access: 19.05.2025).

21) Neves M. C. What are Quantized LLMs?. *TensorOps*. URL: https://www.tensorops.ai/post/what-are-quantized-llms (date of access: 19.05.2025).

22) Tensor Quantization: The Untold Story | Towards Data Science. *Towards Data Science*. URL: https://towardsdatascience.com/tensor-quantization-the-untold-story-d798c30e7646/ (date of access: 09.06.2025).

24) Amanatullah. Fine-Tuning the Model: What, Why, and How. *Medium*. URL: https://medium.com/@amanatulla1606/fine-tuning-the-model-what-why-and-how-e7fa52bc8ddf (date of access: 19.05.2025).

25) Barreto S., Simic M. What Is Fine-Tuning in Neural Networks?. *Baeldung*. URL: https://www.baeldung.com/cs/fine-tuning-nn (date of access: 19.05.2025).

26) LoRA: Low-Rank Adaptation of Large Language Models. *arXiv.org*. URL: https://arxiv.org/abs/2106.09685 (date of access: 19.05.2025).

27) Osmulski R. How to fine-tune a Transformer (pt. 2, LoRA). *Radek Osmulski*. URL: https://radekosmulski.com/how-to-fine-tune-a-tranformer-pt-2/ (date of access: 19.05.2025).

28) Zilliz. LoRA Explained: Low-Rank Adaptation for Fine-Tuning LLMs. *Medium*. URL: https://medium.com/@zilliz_learn/lora-explained-low-rank-adaptation-for-fine-tuning-llms-066c9bdd0b32 (date of access: 19.05.2025).

29) Jawade B. Understanding LoRA - Low Rank Adaptation For Finetuning Large Models | *Towards Data Science. Towards Data Science*. URL: https://towardsdatascience.com/understanding-lora-low-rank-adaptation-for-finetuning-large-models-936bce1a07c6/ (date of access: 19.05.2025).

30) Cretu C. How Does ChatGPT Actually Work? An ML Engineer Explains | *Scalable Path. Scalable Path*. URL:

https://www.scalablepath.com/machine-learning/chatgpt-architecture-explained (date of access: 19.05.2025).

31) Sharma A. J. ChatGPT Architecture: Exploring the Inner Workings of the Language Model. *Medium*. URL: https://medium.com/@ashish.sharma1981/chatgpt-architecture-exploring-the-inner-workings-of-the-language-model-41731fc05483 (date of access: 19.05.2025).

32) Vaj T. The key difference between Transformers architecture and the GPT architecture. *Medium*. URL: https://vtiya.medium.com/the-key-difference-between-transformers-architecture-and-the-gpt-architecture-e47378eca0e8 (date of access: 19.05.2025).

33) Qi J. Inside Multimodal LLaMA 3.2: Understanding Meta's Vision-Language Model Architecture. *Medium*. URL: https://j-qi.medium.com/inside-mllama-3-2-understanding-metas-vision-language-model-architecture-ae12ad24dcbf (date of access: 19.05.2025).

34) Transformers Explained Visually (Part 3): Multi-head Attention, deep dive *Towards Data Science. Towards Data Science*. URL: https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853/ (date of access: 19.05.2025).

35) Ryu H. InstructGPT: Training language models to follow instructions with human feedback. *Medium*. URL: https://medium.com/@uhanho/instructgpt-training-language-models-to-follow-instructions-with-human-feedback-b71a2beafd46 (date of access: 19.05.2025).

36) Singh R. Vision Transformer (ViT). *Medium*. URL: https://medium.com/@RobuRishabh/vision-transformer-vit-39f627d04b2a (date of access: 19.05.2025).

37) Feature Pyramid Networks for Object Detection. *arXiv.org*. URL: https://arxiv.org/abs/1612.03144 (date of access: 19.05.2025).