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 другий (магістерський)

Банківський асистент використовуючи великі мовні моделі (LLM)

(тема)

Виконав:
здобувач другого року навчання,
групи ДСм-23-1
Гребінник Є.В.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Науки про дані (Data Science)
(повна назва спеціалізації)
Керівник проф. Бодянський Є.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ О.В. Золотухін
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

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

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Науки про дані (Data Science)
(повна назва)

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Гребіннику Єгору Вячеславовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Банківський асистент використовуючи великі мовні моделі (LLM)

затверджена наказом університету від 22 листопада 20 24 р. № 1238Ст
2. Термін подання студентом роботи до екзаменаційної комісії 23 січня 20 25 р.
3. Вихідні дані до роботи Набір даних для навчання великої мовної моделі (LLM) з банківськими транзакціями

4. Перелік питань, що потрібно опрацювати в роботі
1) Дослідження та аналіз роботи нейронних мереж
2) Дослідження та аналіз мовних моделей
3) Реалізація додатку банківського асистенту

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	25.11.2024	виконано
2	Аналіз проблеми та огляд існуючих рішень	28.11.2024	виконано
3	Вибір технології розробки та інструментальних	30.11.2024	виконано
	засобів		
4	Тренування асистенту	14.12.2024	виконано
5	Тестування асистенту	17.12.2024	виконано
6	Створення інтерфейсу	20.12.2024	виконано
7	Написання пояснювальної записки	13.01.2025	виконано
8	Перевірка на академічний плагіат	13.01.2025	виконано
9	Нормоконтроль	13.01.2025	виконано
10	Підготовка презентації та доповіді	14.01.2025	виконано
11	Попередній захист	16.01.2025	виконано
12	Рецензування	20.01.2025	виконано
13	Захист перед ЕК	23.01.2025	

Дата видачі завдання 25 листопада 2024 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Бодянський Є.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 63 с., 37 рис., 1 табл., 10 джерел.

AI, BANKING, BANK-ASSISTANT, CLASSIFICATION, FINE-TUNING, FRAUD-DETECTION, HUGGINGFACE, JUPYTER-LAB, KAGGLE, LLM, LORA, NEURAL NETWORKS, NUMPY, PANDAS, PEFT, PROMPT, PYTHON, QLORA, QUANTIZATION, TEXT-GENERATION, TRANSFORMERS.

Об’єкт дослідження – задача автоматизації задач банківських співробітників.

Предмет дослідження – використання великих мовних моделей для автоматизації задач банківських співробітників.

Мета роботи – створення банківського асистенту для класифікації банківських транзакцій на класи: Зловмисна, Легальна.

Методи дослідження – теоретичний (збір та структуризація теоретичного матеріалу), експериментальний (програмна реалізація великої мовної моделі та її навчання). Методи розробки базуються на технологіях Python з фреймворками Pytorch, Transformers.

У результаті було проведено аналіз архітектур великих мовних моделей та архітектури трансформер відповідно, досліджени методи навчання та роботи таких шарів як: self-attention, cross-attention, multi-head attention та блоків: encoder та decoder й використані існуючі набори даних, що складаються приблизно з 16000 банківських транзакцій. Аналогічні дані були використані для розрахунку метрики точності.

ABSTRACT

Master’s thesis contains: 63 pp., 37 fig., 1 tabl., 10 references.

AI, BANKING, BANK-ASSISTANT, CLASSIFICATION, FINE-TUNING, FRAUD-DETECTION, HUGGINGFACE, JUPYTER-LAB, KAGGLE, LLM, LORA, NEURAL NETWORKS, NUMPY, PANDAS, PEFT, PROMPT, PYTHON, QLORA, QUANTIZATION, TEXT-GENERATION, TRANSFORMERS.

The object of research is the task of automating the tasks of bank employees.

The subject of research is the use of large language models to automate the tasks of bank employees.

Purpose – to create a banking assistant for classifying bank transactions into classes: Malicious, Legal.

Research methods: theoretical (collection and structuring of theoretical material), experimental (software implementation of a large language model and its training). The development methods are based on Python technologies with Pytorch and Transformers frameworks.

As a result, we analyzed the architectures of large language models and transformer architectures, respectively, studied the methods of training and operation of such layers as self-attention, cross-attention, multi-head attention and blocks: encoder and decoder, and used existing datasets consisting of approximately 16,000 bank transactions. Similar data was used to calculate the accuracy metric.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект;

AI – Artificial Intelligence – штучний інтелект;

Fine-Tuning – покращення моделі під вузько направлену задачу;

LLM – Large Language Model – велика мовна модель;

PeFT – Parameter Efficient Fine-Tuning – покращення моделі під вузько направлену задачу використовуючи певний обсяг тренувальних параметрів моделі.

ВСТУП

Нейронні мережі в сучасному світі здобули велику популярність і стали одними з найбільш важливих технологій в галузі штучного інтелекту та машинного навчання. Вони відкривають широкі можливості для аналізу даних, розпізнавання образів, обробки природної мови та розв’язання складних завдань, які раніше вважались недосяжними.

Нейронні мережі, іноді також називані штучними нейронними мережами або глибокими нейронними мережами, імітують роботу людського мозку. Вони складаються з великої кількості з’єднаних штучних нейронів, які співпрацюють для виконання завдань обробки інформації. Кожен нейрон приймає вхідні сигнали, обробляє їх і передає результати наступним нейронам у мережі. Завдяки такому масивному паралелізму та здатності до самонавчання нейронні мережі можуть досягати виняткових результатів у різних сферах.

Однією з ключових переваг нейронних мереж є здатність до автоматичного визначення складних залежностей і патернів у великих обсягах даних. Вони можуть виявити невидимі для людського спостереження закономірності та зробити прогнози на основі цих знань. Це особливо корисно у таких галузях, як медицина, фінанси, технології безпеки, маркетинг та багато інших.

Найбільш популярними та актуальними є моделі типу Transformers або вдосконалений варіант LLM. Це моделі які займаються обробкою тексту, ці моделі створюють контекст вхідного тексту та генерують текст враховуючи цей контекст.

Типові задачі цих моделей: text classification (класифікація тексту), machine translation (машинний переклад), token classification (класифікація токена (слова або частини слова (найпростішої частини токенайзеру))), text summary (визначення тези тексту) та аналіз семантичних зв’язків речення. З виходом великих мовних моделей це все можна виділити в одну задачу, а

саме генерація тексту (text generation), саме виконання такої задачі як text generation відкриває можливості до майже повної автоматизації процесів у більшості галузей (банкінг, медицина, фінанси тощо) у вигляді чат-бота наприклад, який здатний відповідати на найскладніші питання та відповідати на питання використовуючи певну базу знань, або ж чат-бот радник який здатен давати пораду у певній галузі аргументуючи свою відповідь.

Найбільша перевага великих мовних моделей полягає саме у їхній універсальності, модель може виконувати створення будь-якого тексту майже на будь-яке запитання, але важливо пам’ятати що модель ніколи не приймає рішення за людину а лише дає їй пораду та те що будь-яка мовна модель може галюцинувати: генерувати текст, який не має жодного сенсу (у більшості випадків серію повторювань літер та їхніх комбінацій).

У цій роботі було розглянуто результат генерації тексту LLM, проведено експерименти з різними запитами до моделі (prompt) та експерименти з обсягом датасету та розподілом його класів. Також використано модель Mistral-7B [6] для створення банківського асистенту.

1 ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОБОТИ НЕЙРОННИХ МЕРЕЖ

1.1 Аналіз структури та роботи нейронних мереж

Deep Feedforward Networks, також відома як Feedforward Neural Networks, є базою для усіх глибоких моделей. Ціль Feedforward Network зробити апроксимацію певної функції f^*. Наприклад для класифікатора $y = f^*(x)$ зв'язує вхідні дані x зі класом y. Feedforward Network представляє відображення $y = f(x; \theta)$ й вивчає значення параметрів θ, й вивчає так що результат це найкраща апроксимація Моделі feedforward названий так, тому що інформацію завжди йде уперед до результату y. У цих моделей нема зворотних зв'язків, у яких результат моделі входить у модель. Якщо ж такі зв'язки існують, то цей тип моделей називається Recurrent Neural Networks.

Feedforward Neural Networks вносять великий вклад у нейронні мережі для новачків, так як вони формують базис для багатьох інших типів нейронних мереж, як наприклад Convolutional Networks які використовують для Object Detection це особливий випадок Feedforward Neural Networks.

Feedforward Neural Networks називається мережою тому що вона уявляються комбінацією або мережою функцій. Таку модель можна уявити у вигляді направленого нециклічного графу. Наприклад маємо три функції: f 1 , f 2 , f 3 , зв'язані поступово одна з одною так, що формують f(x) = f 3 (f 2 f 1 (x))). Такі поступове формування функцій й найбільш частіше використовується у нейронних мережах. Перша функцію називають першим шаром, тоді другу – другим шаром й так далі. На цьому моменті можна визначити глибину моделі як кількість таких функцій. Фінальний шар такої мережі називається вихідним. Другий шар такої мережі називається прихованим, так як на практиці людина з ним ніяк не контактує окрім як навчання на відмінну від вхідного та вихідного. Прихованих може бути скільки завгодно, нема чіткого правила яке б затверджувало що саме така кількість прихованих шарів правильна.

На рисунку 1.1 можна побачити приклад структури Feedforward Neural Network, самої простої моделі – Perceptron [1]. Зв’язки що з’єднують шарі різних рівнів мають певні ваги, так що наступний нейрон має зважену суму.

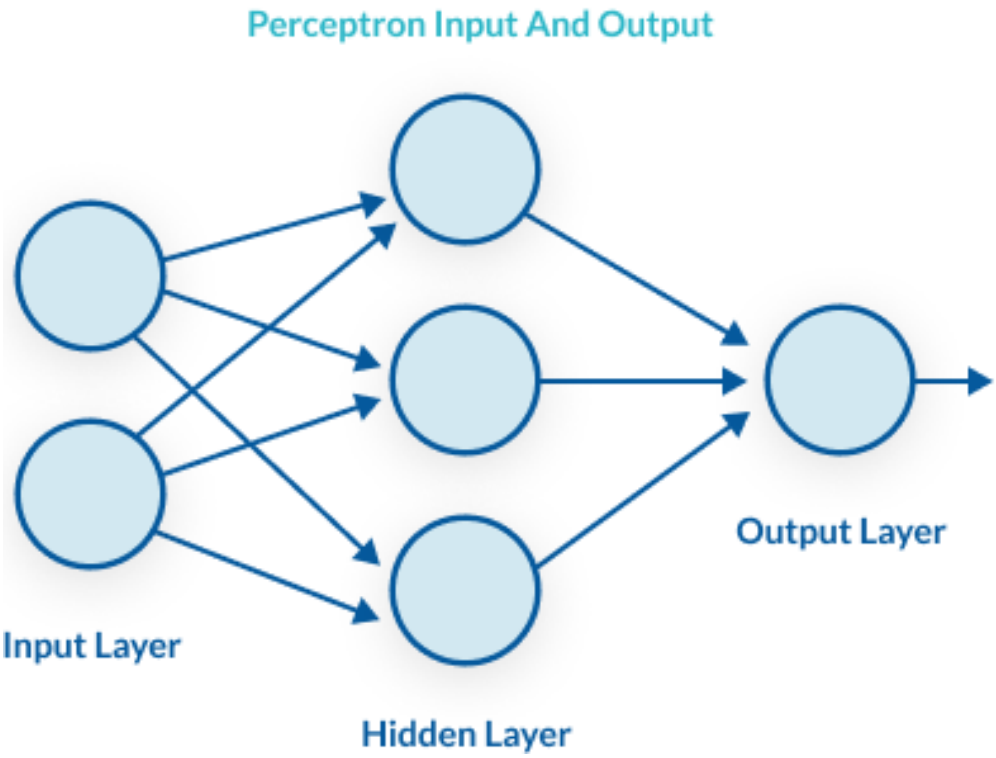

Рисунок 1.1 – Демонстрація структури моделі типу Feedforward neural network, а саме Perceptron

Функція активації [2] виконує водночас декілька функцій:

– нелінійність даних. Завдяки тому що функції активації нелінійні, а до цього нейронна мережа уявлялася як серія лінійних функцій викликаних одне за одною. Функція активації дозволяє розширити варіанти значень які можуть прийняти нейрони в мережі (рисунок 1.2);

– нормалізація даних або відображення даних на інтервал (-1; 1). Переклад на «формат» даних на якому працює нейронна мережа;

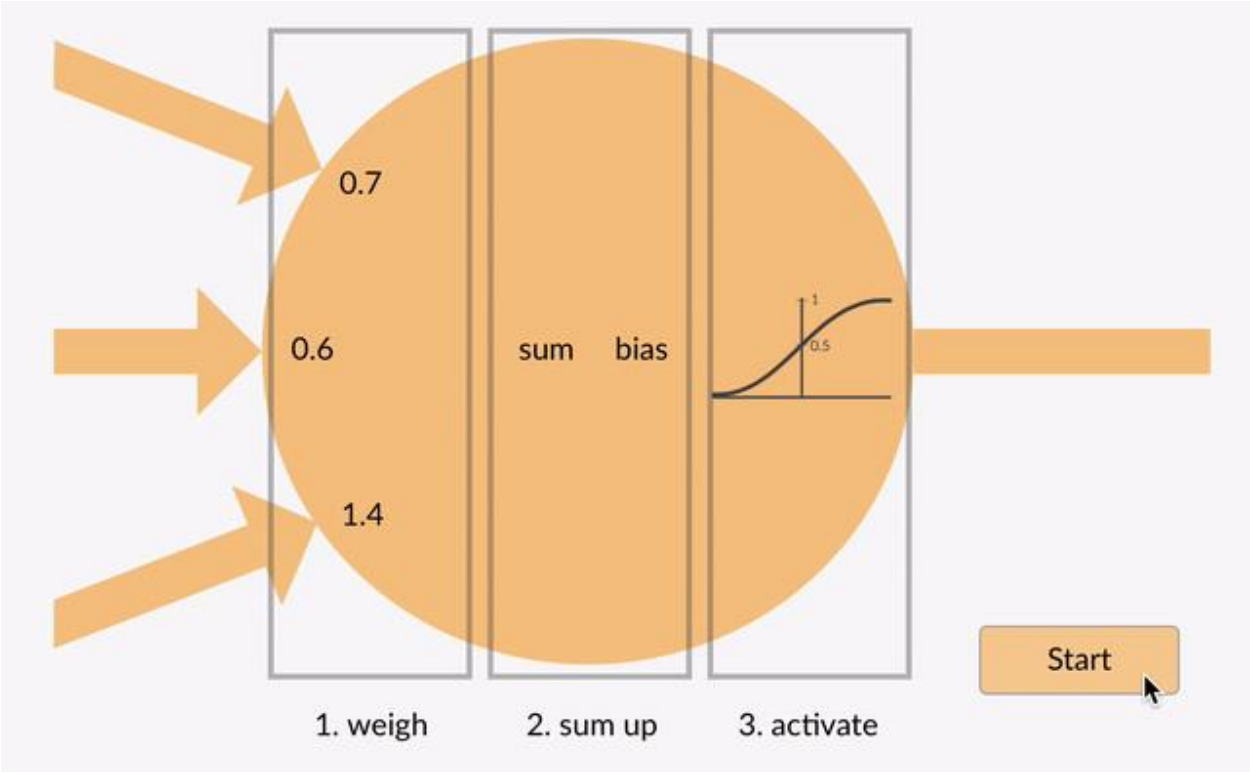

Рисунок 1.2 – Демонстрація отриманням нейрону завдяки взваженої суми нейронів попереднього шару

– гладка функція з монотонною похідною. Можливість знайти часткову похідну (градієнт моделі) на будь-якій ділянці функції, до того ж цей градієнт буде завжди більше ніж нуль;

– наближення до тотожної функції f(x) = x. Цей факт допомагає в апромксимації функції активації, тобто спрощує знаходження похідної на деяких участках функції;

– монотонність.

Також функції активації брали натхнення з науки о вивчені мозку.

Існує різні функції активації, кожна має свої недоліки та переваги.

Sigmoid (рисунок 1.3). $S(x) = \frac{1}{1+e^{-x}}$, де x – вхідне значення нейрона. Одна з причин через яку сигмоїда використовується в нейронних мережах – це простий вираз її похідної через саму функцію (що дозволило істотно скоротити обчислювальну складність. Не менш важливою причиною

введення нелінійності є математично доведена можливість отримати як завгодно точне наближення будь-якої неперервної функції багатьох змінних, використовуючи операції додавання та множення на число.

Як правило використовується на вихідному шарі для підрахунку вирогідності належності до певного класу. Недолік цієї функції у тому, що на початку та кінці функція має насиття, це позначає що градієнт на цих ділянках градієнт буде максимально малим, чому саме це погано буде пояснено у розділі про аналіз навчання нейронних мереж.

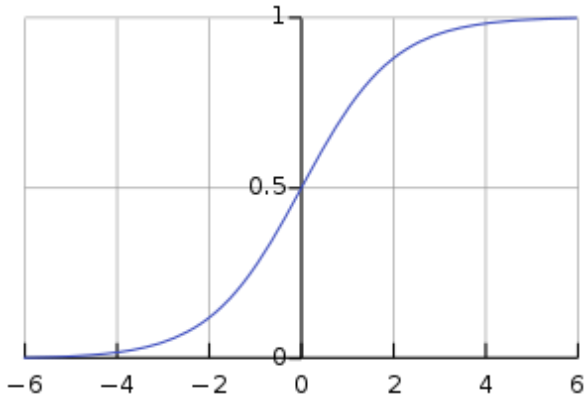

Рисунок 1.3 – Графік функції Sigmoid

ReLU (рисунок 1.4). ReLU(x) = max(0, x), де x – вхідне значення нейрона, найбільш частіше використовується при глибинному навчані в задачах з комп’ютерного зору та розпізнавання мовлення. Перевагами цієї функції є її простота та простота обчислення її градієнта, дуже рідко з’являється проблема обчислення нульових градієнтів. До недоліків можна віднести мертву зону у яких нейрони просто не активується, її необмеженість, не має похідної у нулі. Як правило цю функцію активації використають у прихованих шарах нейронної мережі, так як прихованих шарів може бути необмежена кількість завдяки що рахування цієї функції та її градієнта дуже просте так спрощують використання нейронної мережі (менше витрачається часу на повний прохід (inference) та ресурсів на рахування градієнтів (backpropogation)).

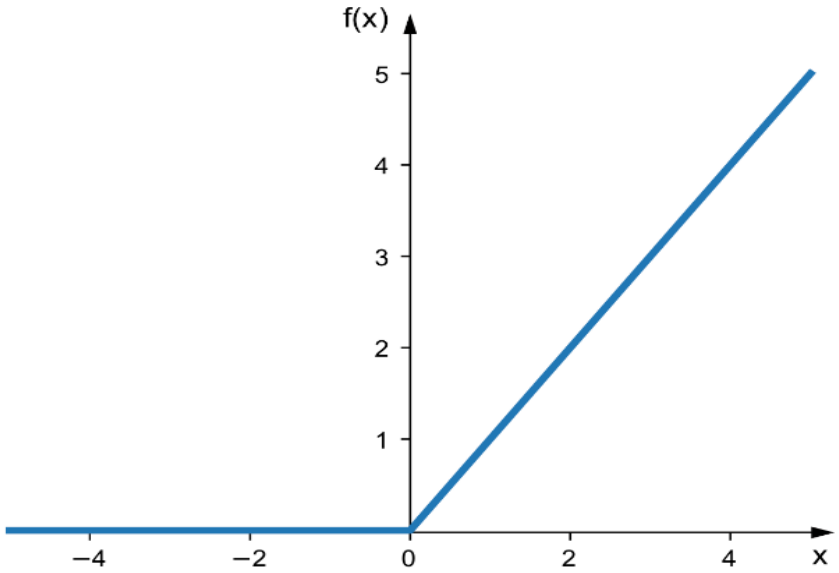

Рисунок 1.4 – Графік функції ReLU

Th (гіперболічний тангенс, рисунок 1.5). Th(x) = $\frac{e^x + e^{-x}}{e^x - e^{-x}}$, де x – вхідне значення нейрона, дуже на сігмоїду і навіть таку схожу S-образну форму, але вихідні значення цієї функції лежать у інтервалі (-1; 1), як правило використовується в задачах класифікації між двома класами.

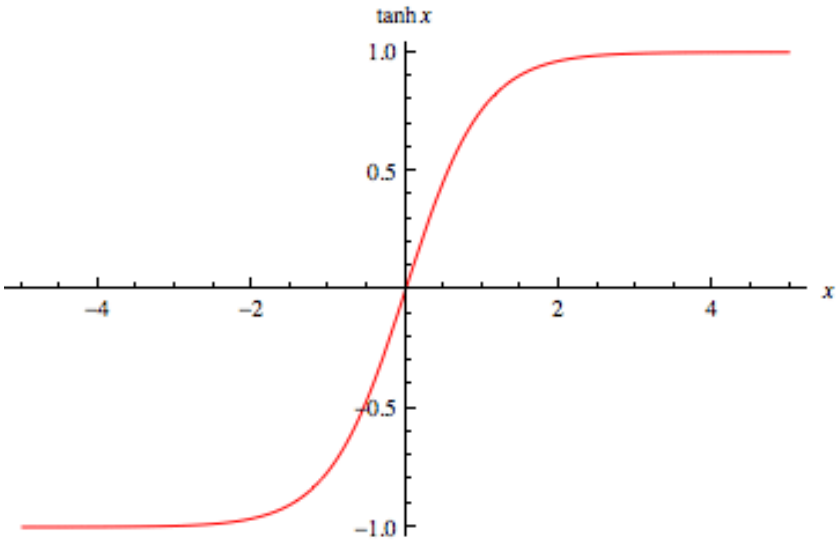

Рисунок 1.5 – Графік функції Th

1.2 Аналіз навчання нейронних мереж

Одну з найважливіших тем є навчання нейронних мереж. Навчання – це етап коригування вагів нейронної мережі. Існує декілька методів навчання: Supervised, Unsupervised, Reinforcement leaning.

Reinforcement learning – це галузь машинного навчання, що вивчає питання про те, які дії повинні виконувати програмні агенти в певному середовищі задля максимізації деякого уявлення про сукупну винагороду, агент у цьому випадку алгоритм який дослідник обрав для навчання, мета Reinforcement learning навчати алгоритм виконувати дії заплановані дослідником у певному просторі, завдяки нагороджуванню за правильну дію.

Unsupervised leaning – один зі способів машинного навчання, при вирішенні яких випробувана система спонтанно навчається виконувати поставлене завдання без втручання дослідника.

Supervised learning – навчання з учителем, контрольоване або кероване навчання, тип навчання в якому алгоритм навчається на прикладі наданим дослідником. Було проаналізовано що для поставленої задачі підходить Supervised leaning.

Як вже було описано Supervised learning – це метод навчання алгоритму коли дослідник додає приклад правильного результату, наприклад розглянемо класифікатор, для того щоб навчити модель відносити певний об’єкт до правильного класу потрібно створити набори даних:[x1, y1], [x2, y2], [x3, y3] … [xn, yn], де xi – набір вхідних даних, yi – клас для вхідних даних xi. Для того щоб подалі описати кроки навчання нейронних мереж, визначимо функцію втрат.

Функція втрат, також відома як функція помилки або функція цілі, визначає міру розбіжності між фактичними та прогнозованими значеннями нейронної мережі. Її використовують для оцінки якості роботи моделі та керування процесом навчання.

Обирання відповідної функції втрат залежить від характеру задачі, яку розв'язує нейронна мережа. Найвідоміші функції втрат:

– середньоквадратична помилка (Mean Squared Error, MSE): ця функція втрат вимірює середнє квадратичне відхилення між фактичними та прогнозованими значеннями. Вона підходить для задач регресії і добре враховує величину помилки;

– категоріальна кросс-ентропія (Categorical Cross-Entropy): ця функція втрат використовується у задачах класифікації з багатьма категоріями. Вона порівнює розподіл ймовірностей фактичних та прогнозованих класів, і чим більше розбіжність, тим вища втрата;

– бінарна кросс-ентропія (Binary Cross-Entropy): ця функція використовується у задачах бінарної класифікації, де потрібно визначити між двома можливими класами. Вона оцінює розбіжність між фактичними та прогнозованими ймовірностями класів;

– середня абсолютна помилка (Mean Absolute Error, MAE): ця функція втрат також використовується для задач регресії і вимірює середню абсолютну різницю між фактичними та прогнозованими значеннями. Вона менш чутлива до великих викидів, оскільки використовує абсолютне значення помилки.

Також існують функції втрат які є модифікацією існуючих як наприклад FocalLoss, це модифікація categorical cross-entropy яка завдяки додатковим гіперпараметрам робить фокус на класи яких менше у датасеті. Вибір функції втрат важливий для успішного навчання нейронної мережі, оскільки вона впливає на оптимізацію параметрів моделі. При виборі функції втрат потрібно враховувати характер задачі, обсяг даних та інші особливості конкретної ситуації.

Далі треба визначити оптимізатор. Оптимізатор в контексті нейронних мереж є алгоритмом, який використовується для налаштування ваг та зміщень моделі з метою зменшення функції втрат і покращення її продуктивності. Оптимізатори грають ключову роль у процесі навчання

нейронних мереж і допомагають знаходити оптимальні значення параметрів.

Основна ідея оптимізаторів полягає в тому, щоб рухатись в напрямку, що мінімізує функцію втрат. Вони використовують градієнтну інформацію, яка вказує напрямок найшвидшого зменшення функції втрат, і здійснюють ітерації для оновлення параметрів моделі.

Існує багато алгоритмів навчання нейронних мереж такі як: Adam, AdaGRAD, RMSprop та ін., але у цьому розділі ми розглянемо алгоритм навчання SGD, що розшифровується як Stochastic Gradient Descent, це найбільш популярний та найбільш простий алгоритм навчання. Для того щоб навчити який можна описати двома формулами:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} L_i(w), \tag{1.1}$$

де L(w) – загальна функція втрат на весь набір даних;

L_i(w) – функція втрат для одного набору даних;

$$w := w - \alpha \nabla \mathrm{L}(\mathrm{w}) = w - \frac{\alpha}{n} \sum_{i=1}^{n} \nabla L_i\,(w), \tag{1.2}$$

де α – гіперпараметр який називається learning rate;

∇ – модуль набла, вектор найскорішого росту.

Слід зауважити що алгоритм шукає локальний мінімум або локальний максимум, тому може бути ситуація коли оптимізаційний алгоритм потрапляє у локальний мінімум й не може з нього вийти, або оптимізаційний алгоритм його просто «переступає» й не може попасти у глобальний мінімум, для того щоб уникати ситуації описані вище у формулі використовується гіперпараметр learning rate, по факту learning rate це відсоток градієнта який враховується у зміни вагів нейронів мережі, якщо learning rate занадто малий то навчання моделі може бути набагато довшим

ніж могло бути або навчання зовсім не завершиться, або оптимізатор знайде локальний мінімум та не зможе вийти з нього, якщо learning rate занадто великий то оптимізатор може дуже довго обходити колами глобальний мінімум, тому дуже важливо обрати правильний для задачі learning rate, як правило learning rate ставиться від 0.01 до 0.001. Є деякі правила які допомагають встановити правильний learning rate.

Learning rate треба ставити вище ніж зазвичай у випадках:

– модель занадто глибока з-за занадто низького learning rate, градієнти можуть просто не доходити до початкових шарів, як наслідок ваги у нейронах цих шарів майже не змінюються, тому результат сходження моделі може бути дуже поганим;

– дані на яких навчається модель збалансовані, дуже важливо щоб дані на яких навчається модель мали рівномірний розподіл або максимально схожий розподіл до рівномірного, якщо розподіл даних не є так, може виникнути проблеми зі сходженням моделі.

Окрім learning rate є ще гіперпараметри які дозволяють прискорити сходження моделі:

– Weight decay – один зі способів прискорення сходження моделі. Weight decay додає до фінальної функції втрат суму квадратів вагів нейронній мережі помноженою на коефіцієнт weight decay. Ідея у тому щоб штрафувати алгоритм за надання занадто великих вагів нейронам. Великі ваги дуже погано для нейронної мережі так як на градієнти становляться більше на початку й як у наслідок маємо проблему у вигляді великого кроку для оптимізатора яка була описана вище;

– Momentum – один зі способів прискорення сходження моделі. Momentum робить експоненціальну середню з попередніх градієнтів, кількість градієнтів залежить від параметра β, як правило його значення дорівнює 0.9, на практиці застосування цього методу дозволяє оптимізатору набагато швидше знайти глобальний мінімум (рисунок 1.6);

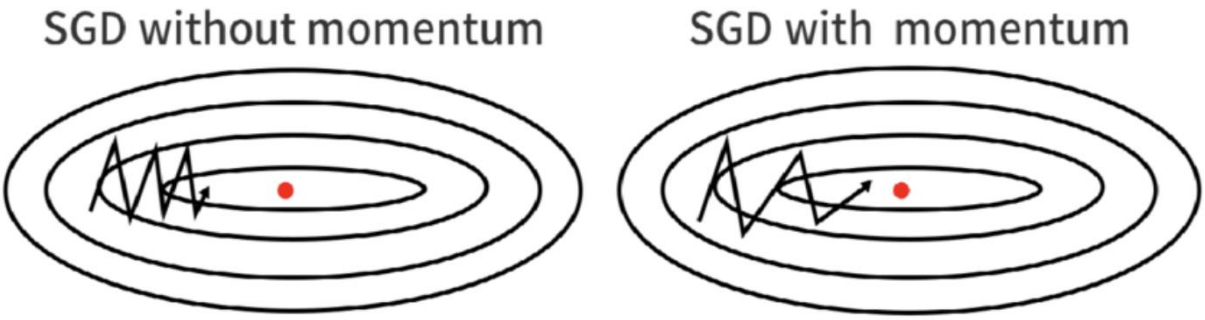

Рисунок 1.6 – Демонстрація застосування метода momentum

– використання batch size – одна з найвідоміших технік, ця техніка не так прискорює навчання моделі скільки робить його зручнішим. Від самого початку планувалося що підрахунки градієнтів будуть відразу для усього набору даних, але на практиці це майже неможливо виконати, так як усі підрахунки градієнтів не вмістяться у пам’ять. Тому використовують репрезентативну вибірку набору даних розміру batch size для навчання моделі.

Остання техніка не стосується покращення навчання, а саме рахування градієнтів на практиці. На практиці щоб підрахувати усі градієнті моделі, використовують метод зворотного поширення помилок або Backpropogation. Важно розуміти, що backpropogation це лише спосіб обчислення градієнтів згенерованих алгоритмом SGD. Backpropogation використовує правило знаходження часткової похідної методом ланцюга, тобто спочатку знаходиться часткова похідна від останнього шара, потім знаючий цей градієнт знаходиться передостанній методом ланцюга складної похідної.

Так йде до останнього шара (тобто початкового), якщо при звичайному використанні моделі дані йдуть спочатку до кінця, то в backpropogation градієнти йдуть від кінця до початку й потім застосовуються зміни до вагів за формулою:

$$w := w - \alpha \frac{\partial L}{\partial w}, \quad (1.3)$$

де L – функція втрат;

α – learning rate.

2 ДОСЛІДЖЕННЯ ТА АНАЛІЗ МОВНИХ МОДЕЛЕЙ

2.1 Препроцесінг мовних моделей

Мовні моделі це ті моделі які використовують текст для вивчення паттернів даних та використання їх у певних задачах. Проте, сам текст модель сприйняти не може так як принцип навчання мовних моделей такий самий як у інших моделей: побудування функції втрат та робити зміни у вагах відповідно зі значенням функції втрат. Для того щоб перевести текст у числові значення мовні моделі використовують таку структуру даних як tokenizer. Токенайзер не тільки перетворює текст на число та розбиває його на менші одиниці які й називаються токенами. Для того щоб токенайзер почав працювати потрібно «навчити» його на корпусі (повний обсяг слів датасету).

Існують такі методи токенизації зі своїми плюсами та мінусами:

– проста токенизація (рисунок 2.1) – метод, у якому використовуються регулярні вирази (правила за якими розбивається текст на групи та правила за відбувається пошук у тексті) на підставі пробілів і розділових знаків, тобто токен – це й є слово. Це дуже простий у використанні метод, проте головний недолік цього методу – надмірне використання пам’яті, так як кожне слово має свій власний токен, навіть слова, які відтворені від одного слова мають різні токени (у англійській мові це word building, в українській – відмінювання слова);

– токенизація на основі пробілів (рисунок 2.2) – метод, який розбиває текст на токени, використовуючи пробіли як роздільники, при цьому на відміну від попереднього може не враховувати знаки пунктуації. Недоліки такі самі як у попереднього методу, навіть недолік попереднього методу розкривається ще більше, так як знаки пунктуації не враховуються й можуть з’явитися дублі, наприклад, з різницею у кількості знаків окликання

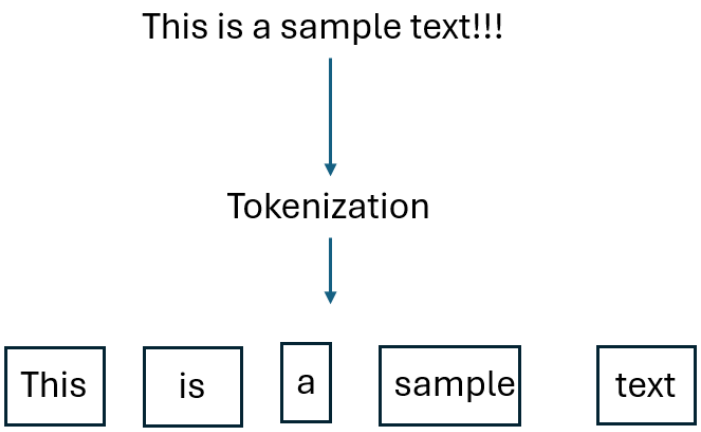

Рисунок 2.1 – Демонстрація простої токенизації

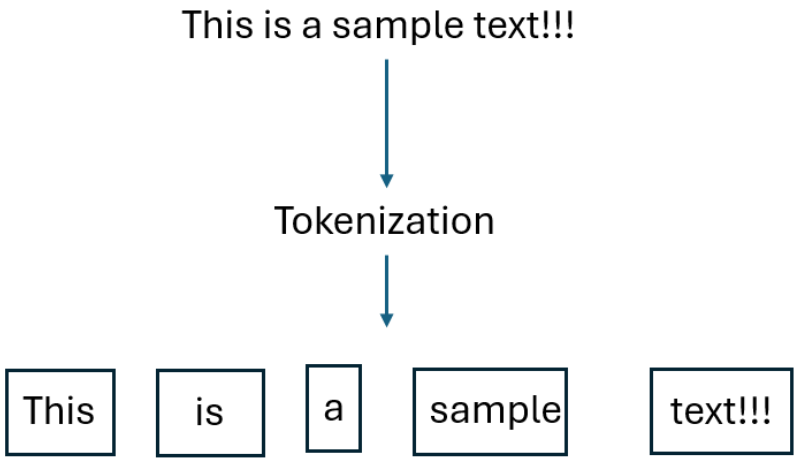

Рисунок 2.2 – Демонстрація токенизації на основі пробілів

– токенизація на основі конкатенації – метод, у якому слова розглядаються як послідовності символів, розбиті певними знаками, такими як дефіси, апострофи тощо. Цей спосіб робить використання пам’яті меншим, так як токенайзер перетворює літери на числа, проте робить вхідну послідовність чисел для нейронної мережі занадто довгою, що приводить до незручностей у роботі з моделі та погіршення результатів, так як шари які дивляться на контекст будуть дивитися на послідовність літер а не послідовність слів;

– морфологічна токенизація – метод, у якому слова розбиваються згідно морфологічного аналізу на корені, префікси та суфікси, використовується для мов з великою кількістю морфологічних правил;

– токенизація зі службовими токенами – метод, який обробляє спеціальні символи або терміни, наприклад URL адреси або адреса електронної пошти, для того щоб зберегти важливу інформацію;

– контекстна токенизація – використовує моделі нейронних мереж для токенизації використовуючи контекст слів, метод потребує багато ресурсів проте є дуже корисним коли повні слова в залежності від контексту мають декілька значень;

– Byte Pair Encoding – метод, який діє шляхом послідовного об’єднання найбільш частих пар символів, створюючи нові токени, найчастіше використовується так як має здатність зменшити розмір словника токенайзеру й при цьому не робить вхідну послідовність занадто великою.

Проте використання токенайзеру не дає вирішення проблеми контексту, оскільки дуже важливо за яким номер стоїть токен та біля яких він стоїть. Так як у всіх задачах NLP використання контексту є дуже важливою частиною навчання моделі. На відмінну від моделей типу seq2seq, моделі які не використовують шари self-attention [3], cross-attention тощо, мають реалізацію врахування контексту. У мовних моделях типу трансформер для створення контексту використовується шар positional-embedding. Тобто замість того щоб зробити реалізацію врахування позиції токену та його сусідніх токенів, до вектору з токенами додаються значення які залежать від позиції токену у послідовності. Позиційні значення можуть бути створені за допомоги різних функцій, проте як правило прийнято використовувати синусоїдальні функції. Приклади функцій:

$$PE(pos, 2i) = \sin\left(pos/10000\text{^}(2i/d)\right), \qquad (2.1)$$

$$PE(pos, 2i + 1) = \cos(pos/10000\hat{}(2i/d)), \quad (2.2)$$

де d – розмірність вектору;

i – позиція токену у цьому векторі.

Після перетворення слів на токени та додавання до них позиційного вектору цей вектор подається до моделі як вхідні дані.

2.2 Архітектура мовних моделей

Мовна модель складається в основному з двох блоків, а саме encoder та decoder. Encoder (рисунок 2.3) – частина нейронної мережі задача якої створити контекстні вектори які б описували зміст речення та взаємозв'язок між різними токенами

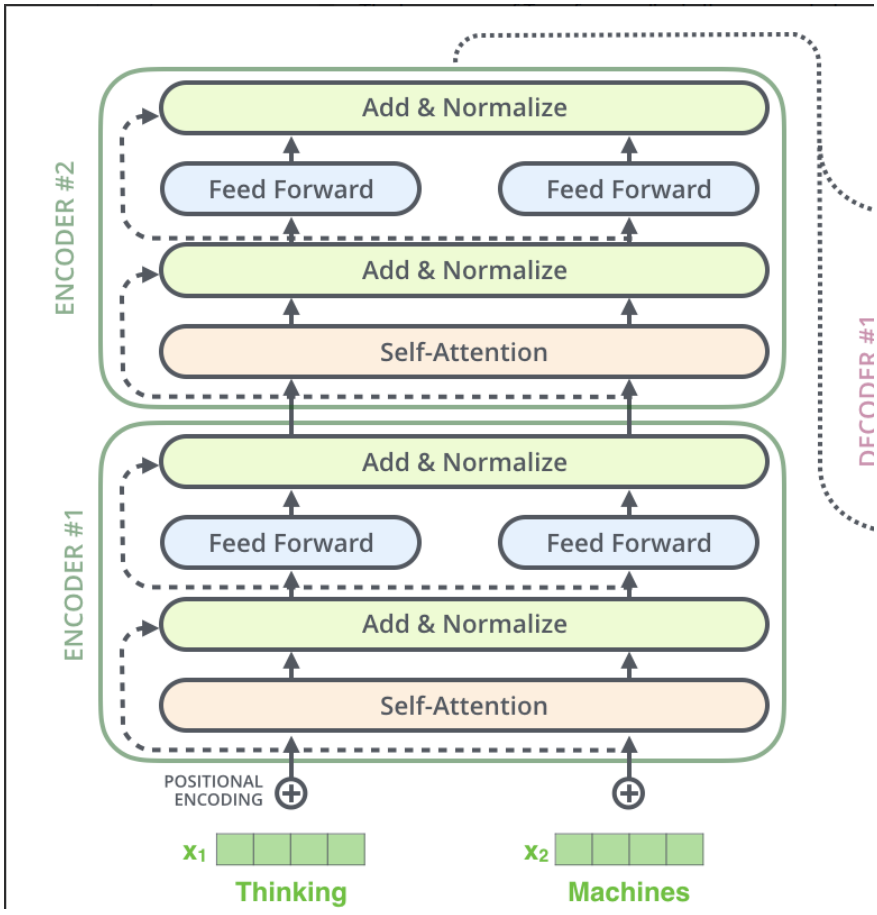

Рисунок 2.3 – Архітектура моделі transformer блоку encoder

Encoder блок – це набір residual блоків (блок який оброблює вхідні дані та робить конкатенацію їх з вхідними даними), у кожного residual блоку є feed forward шар мета якого змінити розмірність вхідного вектору та знайти складні паттерни у векторах. Найбільш цікаве у цих блоках шар – self-attention, механізм який створює контекстні вектор, які дозволяють сфокусуватися на більш значущих токенах. На рисунках 2.4–2.10 наведена робота self-attention [3], [4].

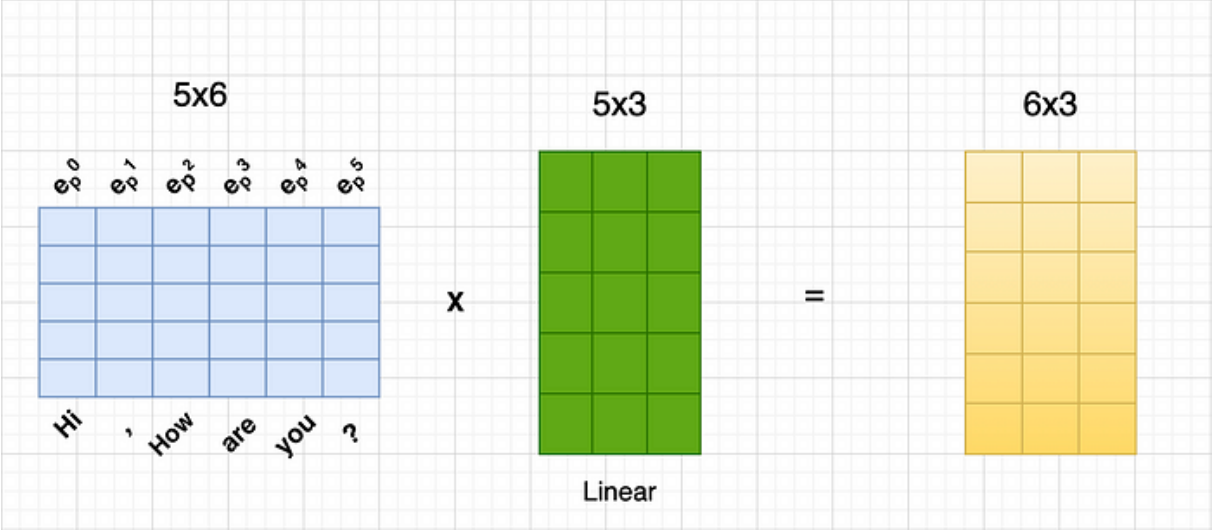

Рисунок 2.4 – Демонстрація роботи self-attention

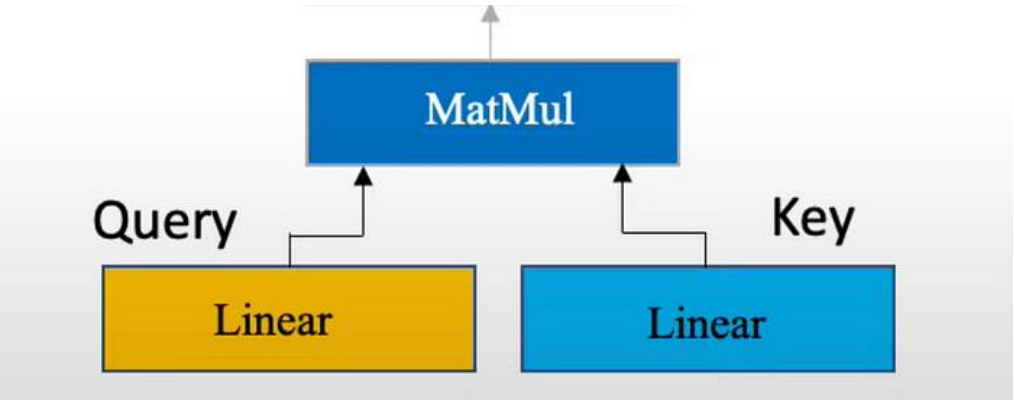

Рисунок 2.5 – Демонстрація роботи self-attention

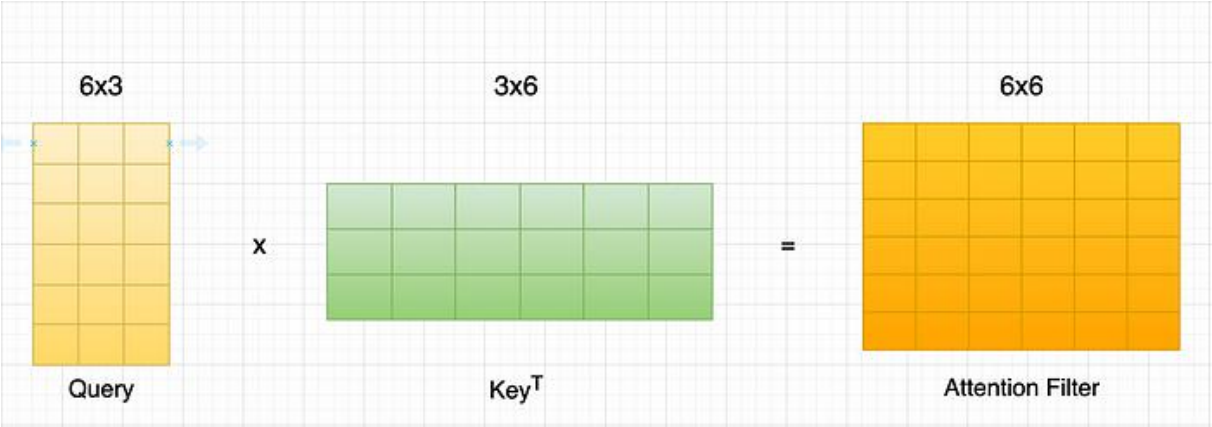

Рисунок 2.6 – Демонстрація роботи self-attention

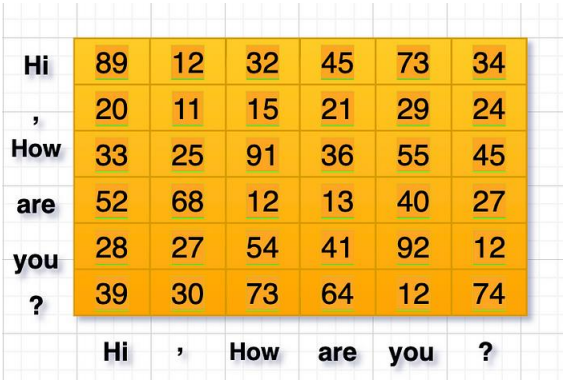

Рисунок 2.7 – Демонстрація роботи self-attention

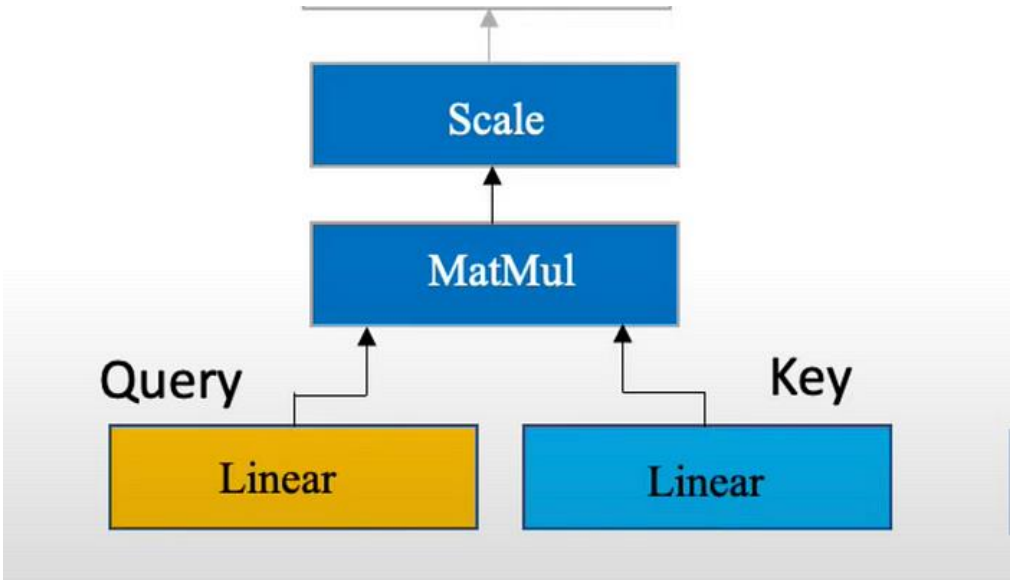

Рисунок 2.8 – Демонстрація роботи self-attention

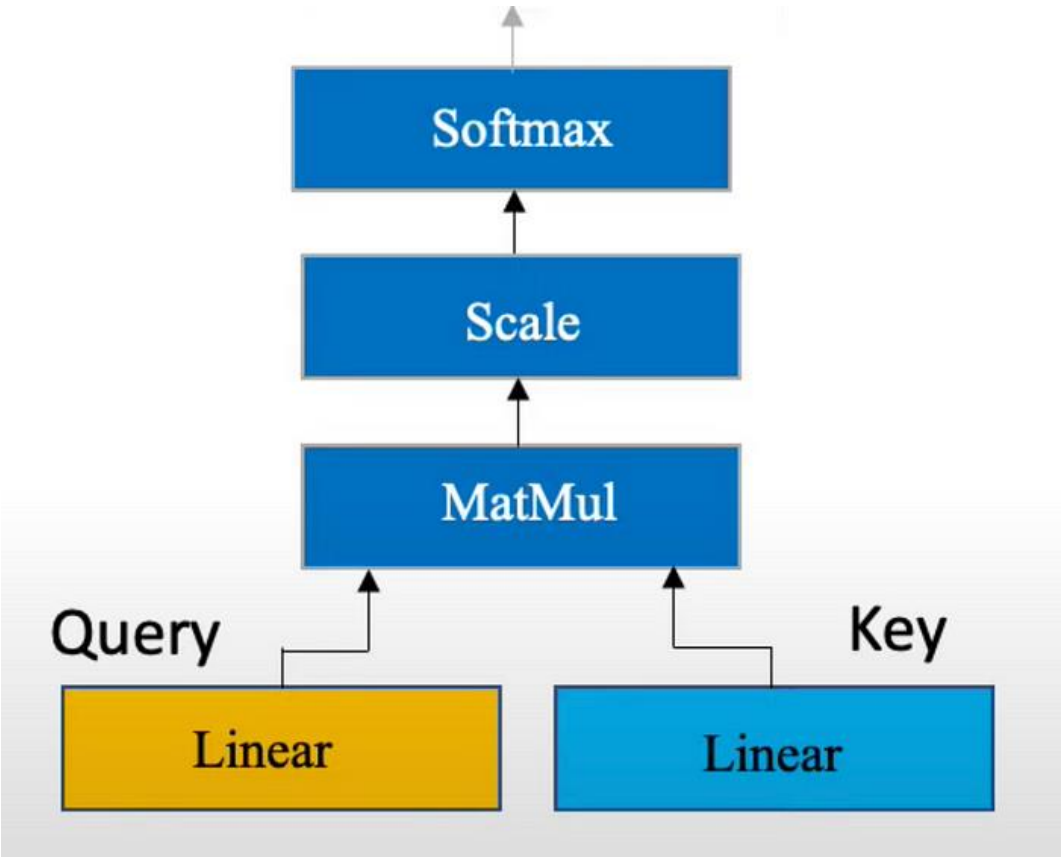

Рисунок 2.9 – Демонстрація роботи self-attention

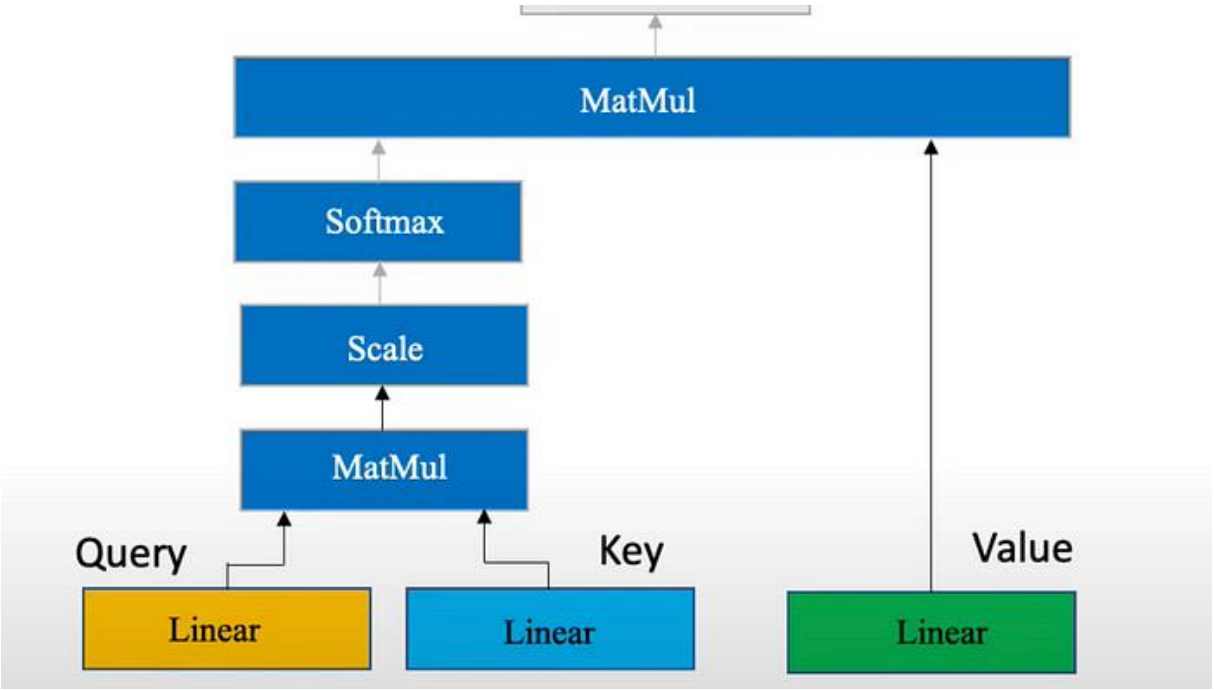

Рисунок 2.10 – Демонстрація повної архітектури self-attention

Encoder роздроблює вектори токенів на набори векторів Query (Запит), Key (Ключ), Value (Значення), ці вектори будуються за допомогою певних матриць ваг: W_Q, W_K, W_V. Спочатку шар робить обчислення коефіцієнтів уваги (attention scores), робиться це за допомогою скалярного добутку між матрицею запитів та матрицею ключів:

$$Q * \frac{K^T}{sqrt(dk)}, \tag{2.3}$$

де dk – розмірність ключів, що дозволяє нормалізувати значення для запобігання великих значень при активації функції softmax, softmax по факту дає ймовірності так згідно зі властивістю softmax сума значень вектору дорівнює одиниці.

$$\text{Softmax}(Q * \frac{K^T}{sqrt(dk)}). \tag{2.4}$$

Потім ці ймовірності використовуються як ваги для зваженої суми

Також для покращення роботи self-attention та cross-attention було розроблено модифікацію Multi-Head Attention (рисунок 2.11). Основна ідея полягає у тому щоб використовувати декілька шарів attention паралельно, для того щоб захопити фокус на різні аспекти речення, тобто чим більше паралельних шарів, тим більше захоплюється варіантів контексту. Результати кожної голови об’єднаються і потім проходять лінійний шар для змінення розмірності вектори та захоплення найважливіших паттернів, як вже було описано вище це дозволяє моделі вивчати різні представлення контексту, враховуючи різні аспекти вхідних даних.

Після роботи енкодеру результат його передається до блоку decoder (рисунок 2.12), який використовує residual блоки використовуючи контекстні вектори для створення вихідної послідовності слів.

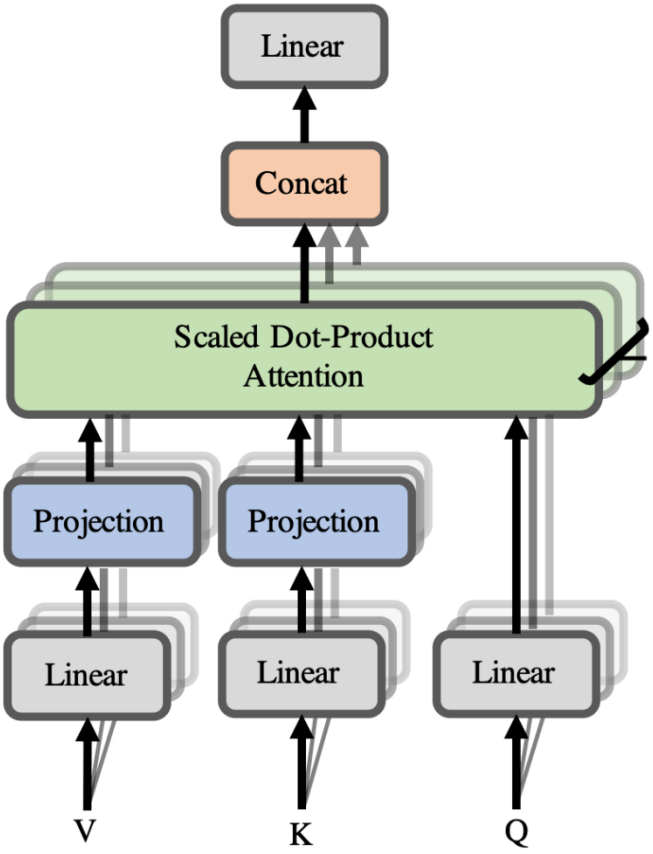

Рисунок 2.11 – Демонстрація структури multi-head attention [4]

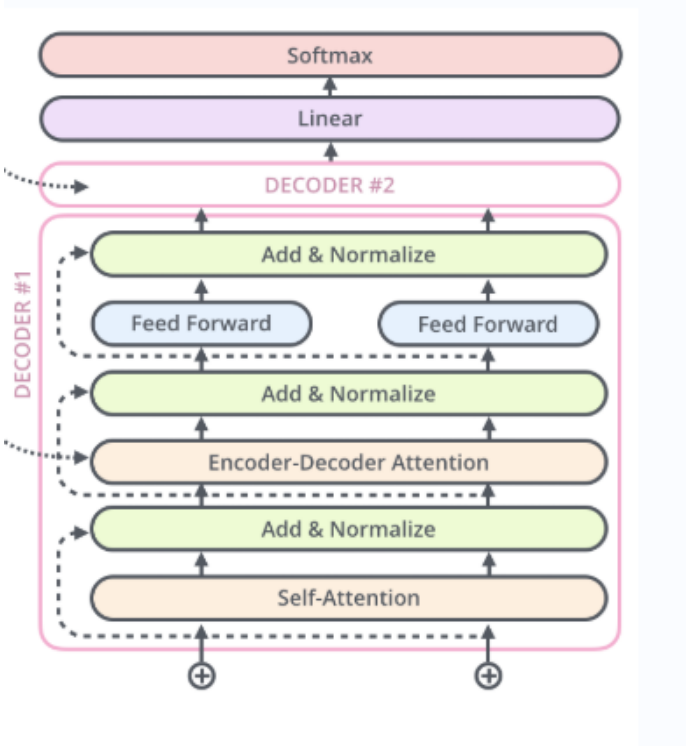

Рисунок 2.12 – Демонстрація архітектури decoder [4]

Якщо точніше, у першому кроці decoder використовує self-attention шар, коли у наступних кроках використовується шар cross-attention або encoder-decoder attention (рисунок 2.13), який використовує контекстні вектори як набір векторів Keys, Values. У випадку створення тексту модель бере контекстні вектори з encoder блоку та спеціальний стартовий токен: [s] наприклад, кожен раз додає вихід моделі до цього токену аж поки не отримає стоп токен: [/s] наприклад й не отримає усе речення або текст.

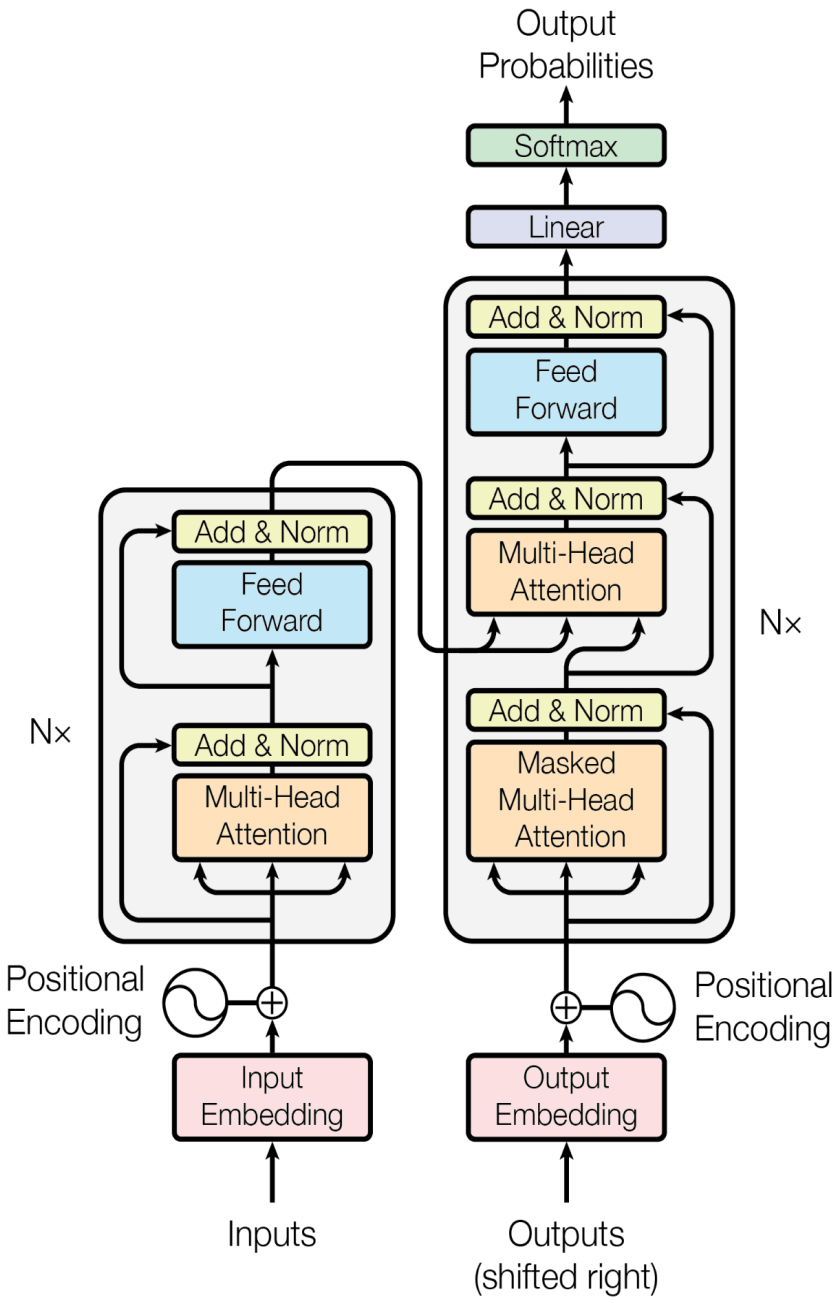

Рисунок 2.13 – Структура моделі типу transformer [5]

Тренування моделі проходить за такими самими функціями втрат, як й не у мовних моделях, завдяки останньому шару мовної моделі та функції втрат, будується задача для нейронна мережа й тренується. Так наприклад коли задача полягає у token classification, тобто коли потрібно класифікувати кожний токен до нейронної мережі додається feed forward шар який змінює розмірність з dim1 до dim2, де dim2 – потрібна кількість класів, або ж задача машинного перекладу де feed forward шар змінює розмірність на корпус слів певної мови. У випадку text classification використовується шар який бере перше сховане значення (hidden states) для класифікації усього тексту.

Для покращення результатів моделі transformer деякі реалізації роблять претренувальний етап який робить тренування моделі на таких задачах як Mask language model (рисунок 2.14) – навчання під час якого деякі токени за певними правилами маскуються, у випадку моделі BERT замінюється 15% відсотків токенів. Модель має передбачити оригінальні токени на основі немаскованих токенів, це завдання значно покращує здатність моделі приділяти увагу контексту.

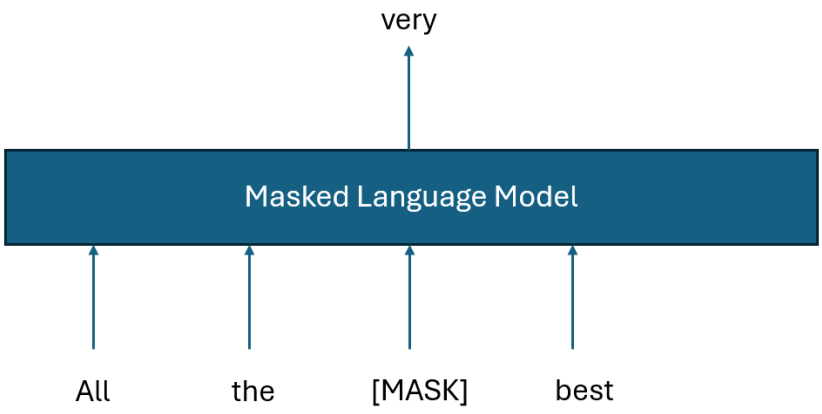

Рисунок 2.14 – Демонстрація претренування на задачі masked language

Класифікація наступного речення (Next sentence prediction) – тренування, мета якого полягає у тому щоб класифікувати чи є подане речення наступним для вхідного речення, для цього модель отримує пари речень, де 50% з низ – справжні пари а інші 50% є випадковими. Це

тренування значно покращує здатність моделі до розуміння зв'язків між реченнями. Після тренування ці ваги зберігаються та використовуються для тренування на певних задачах NLP (рисунок 2.15).

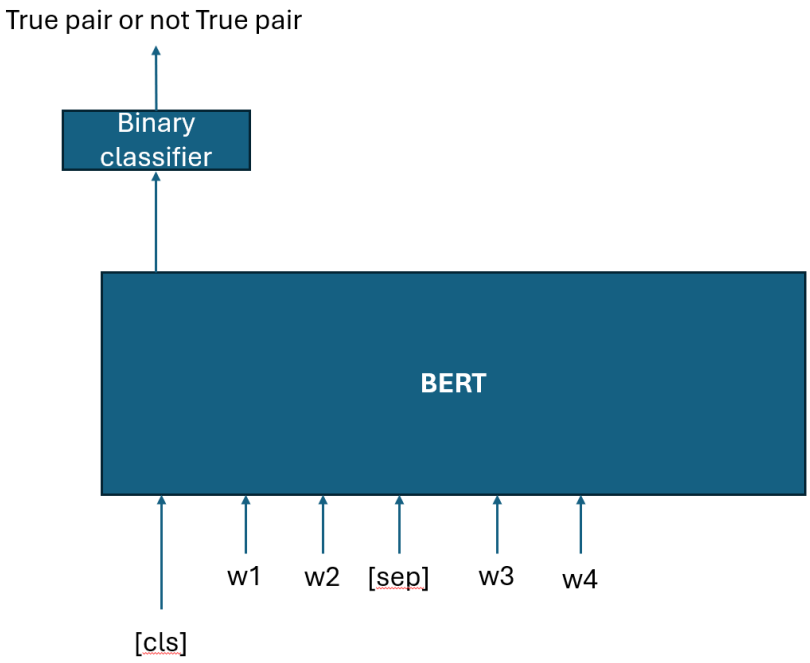

Рисунок 2.15 – Демонстрація претренування на задачі Next sentence prediction

В залежності від реалізацій моделі transformer виконуються різні претренувальні етапи, так наприклад покращена модель BERT – RoBERTa, окрім вище описаних використовує Dynamic masking як претренувальний етап, де маскування проходить випадково кожну епоху навчання, що значно підвищує різноманітність, даних для навчання і допомагає моделі бути більш узагальненою.

2.3 Архітектура моделі Mistral 7B та fine-tuning LLM

З виходом LLM моделей з'явилась можливість змінити архітектуру transformer так як LLM завжди виконує задачу next token prediction, тобто бере стартовий токен додає до нього запит (prompt) та видає найбільш

ймовірне наступне слово й продовжує це того моменту коли модель видає end token (рисунок 2.16), LLM модель Mistral 7B [6] – це LLM модель структура якої така сама як у архітектури transformer за винятком того що модель використовує лише decoder блок.

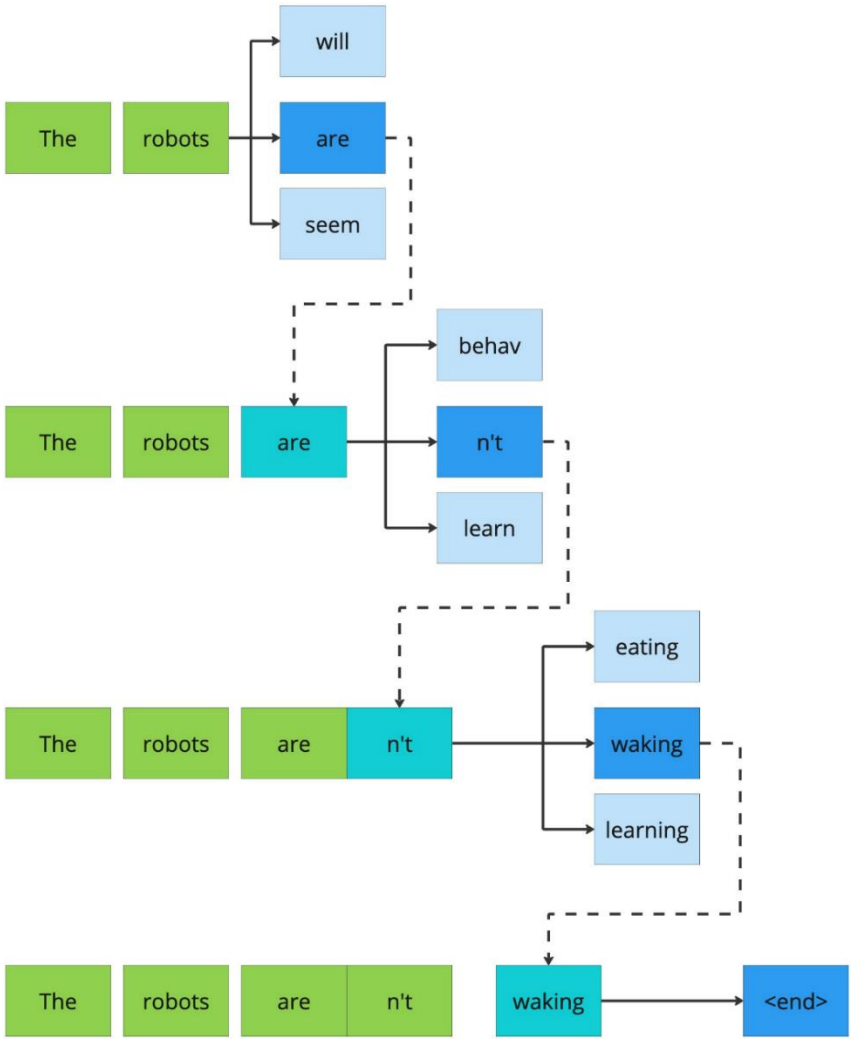

Рисунок 2.16 – Демонстрація задачі next token prediction

Так як модель «дивиться» завжди зліва направо у зв’язку з типом задачі next token prediction, потреба у створенні контексту з двох сторін зникає тому й саме encoder блок прибирається, проте архітектура LLM може відрізнятися, деякі – decoder-only, а інші encoder-only та encoder-decoder. Mistral 7B [6] показує вражаючі результати які порівнюються з сімейством моделей GPT та сімейством моделей LLaMA (рисунок 2.17).

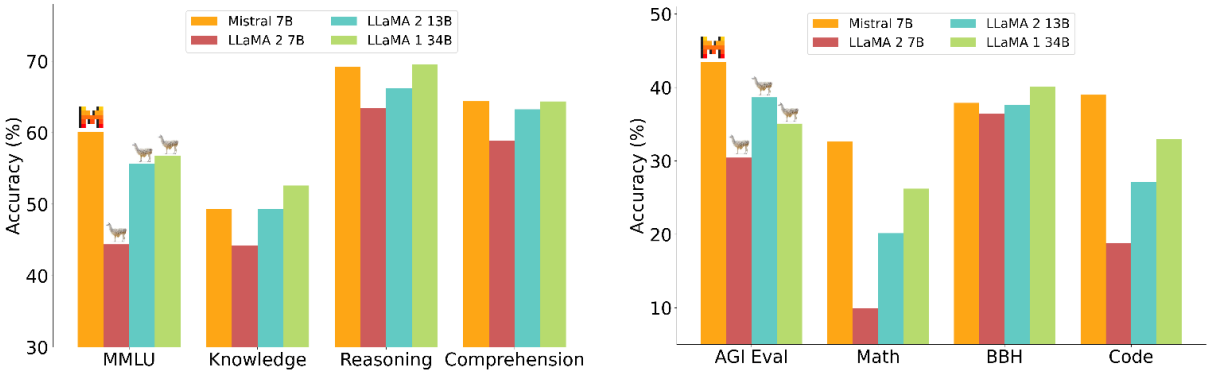

Рисунок 2.17 – Порівняння моделі Mistral 7B з сімейством LLaMa у бенчамарках [6]

Головна перевага Mistral 7B [6] перед іншими моделями у тому що вона значно менше, що дозволяє її вмістити на відеокарти з меншим обсягом відео пам’яті.

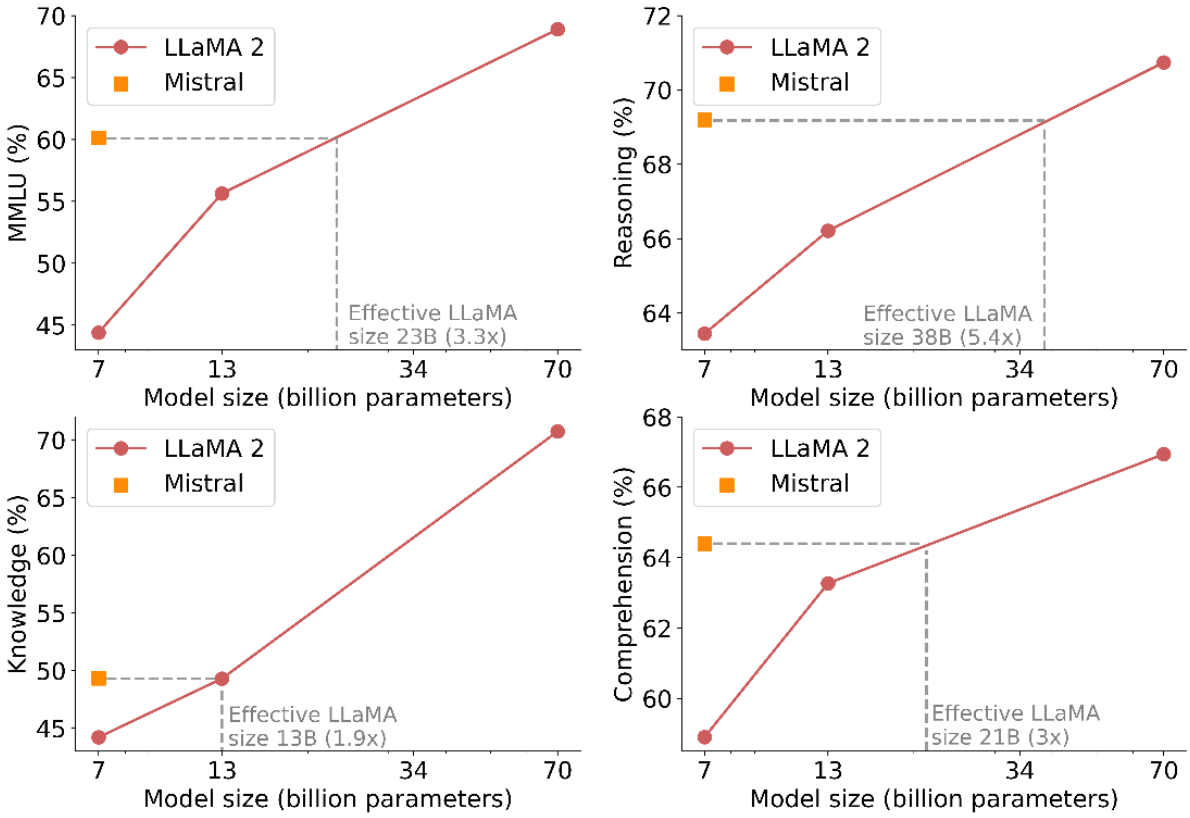

Рисунок 2.18 – Порівняння розмірів моделей [6]

Також Mistral 7B [6] має деякі архітектурні змінення окрім видалення encoder блоку:

– внесені змінення у шар self-attention [3] (рисунок 2.17), а саме windowed self-attention[3]. Шар приймає параметр window_size який показує відстань на яку шар бере токени для створення контекстів векторів, сам по собі такий шар зменшує якість моделі у наданні найбільш підходящого слова, проте так як модель використовує multi-head attention, тому в даному випадку це можна оцінити як ансамбль слабких self-attention шарів;

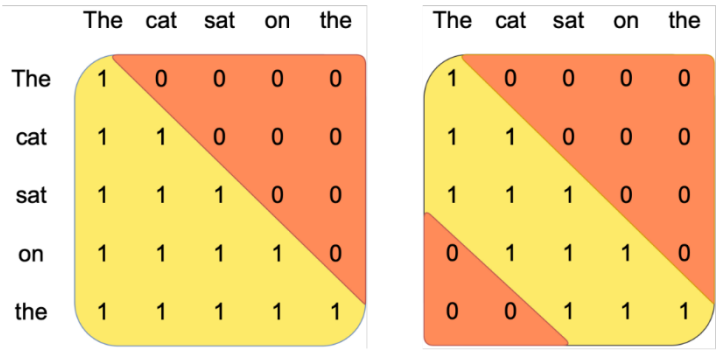

Рисунок 2.17 – Демонстрація звичайного attention (зліва) та sliding window attention (справа)

– використання Key Value кеш пам’ять фіксованої множини з принципом циклічного буферу (коли усі комірки зайняти, модель перезаписує значення в комірки за принципом i mod W, де i – це індекс токена, W – розмір кешу) для прискорення навчання та звичайної роботи моделі (рисунок 2.18).

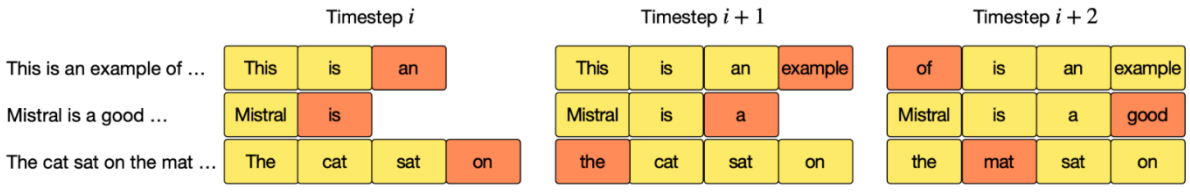

Рисунок 2.18 – Кеш пам’ять моделі Mistral 7B

Проте навіть таку ефективну модель дуже складно помістити на відеокарти с великим batch size та також не кожна відеокарта зможе прорахувати градієнти для 7 мільярдів тренувальних параметрів або це займе дуже багато часу. Для того щоб зменшити обсяг необхідної відео пам’яті та зменшити час на тренування моделі використовують такі техніки як квантизація (quantization) [10] та Peft [9] (parameter-efficient fine-tuning).

Peft [9] – техніка при якому модель тренують на певному обсязі параметрів, з точки зору ресурсів – зменшення тренувальних параметрів в декілька разів, що значно зменшує навантаження на відео чіп та відео пам’ять так як прораховується менша кількість градієнтів та використовується менше пам’яті для їх збереження.

Один з таким методів – LoRA [7]. LoRA (low rank adaptation) метод у якому матриця вагів (тренувальних параметрів) розбиваються на дві матриці меншого розміру добуток яких дають ту саму кількість параметрів (рисунок 2.19).

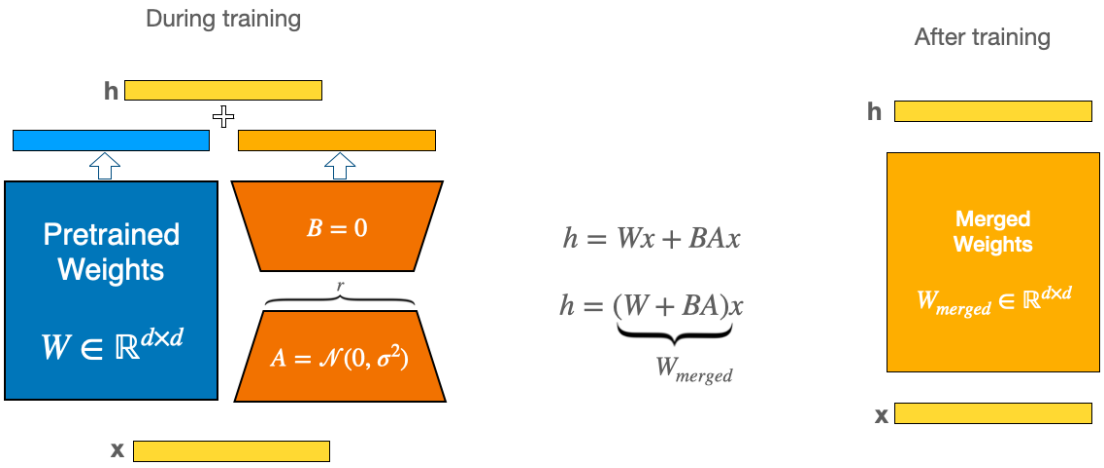

Рисунок 2.19 – Демонстрація методу LoRA

В принципі, LoRA можна застосувати до будь-якої підмножини вагових матриць у нейронній мережі, щоб зменшити кількість параметрів, які можна навчити. Однак для простоти та додаткової ефективності параметрів у трансформаторних моделях LoRA зазвичай застосовується

лише до блоків уваги. Отримана кількість параметрів, які можна навчити, у моделі LoRA залежить від розміру матриць оновлення низького рангу, який визначається в основному рангом ri формою вихідної вагової матриці.

Квантування [10] – це техніка, яка дозволяє зменшити витрати на обчислення та пам’ять для виконання висновків шляхом представлення вагових коефіцієнтів і активацій за допомогою типів даних із низькою точністю, таких як 8-бітне ціле число (int8) замість звичайного 32-бітного числа з плаваючою комою (float32).

Зменшення кількості бітів означає, що отримана модель потребує менше пам’яті, споживає менше енергії (теоретично), а такі операції, як множення матриці, можна виконувати набагато швидше за допомогою цілочисельної арифметики. Це також дозволяє запускати моделі на вбудованих пристроях, які іноді підтримують лише цілі типи даних.

Основна ідея квантування досить проста: перехід від високоточного представлення (зазвичай 32-бітного числа з плаваючою точкою) для вагових коефіцієнтів і активацій до нижчого типу даних. Тип даних накопичення вказує тип результату накопичення (додавання, множення тощо) значень відповідного типу даних.

Двома найпоширенішими випадками квантування є з float32 на float16і та з float32 -> на int8.Завдяки квантуванню модель займає набагато менше місця та потребує значно менше ресурсів для прорахування градієнтів так як квантування замінює тип даних на менший де це не впливає на точність моделі.

Квантування буває симетричне та афінне, симетричне –квантування яке описується інтервалом [-a; a], афінне – квантування яке описується інтервалом [-a; b]. Для визначення значень a, b використовується калібрування.

Калібрування – це етап квантування, на якому float32 обчислюються діапазони. Для ваг це досить легко, оскільки фактичний діапазон відомий

під час квантування. Але для активацій це менш зрозуміло, і існують різні підходи:

– динамічне квантування – після навчання: діапазон для кожної активації обчислюється на льоту під час виконання. Хоча це дає чудові результати без зайвих зусиль, воно може бути трохи повільнішим, ніж статичне квантування, через накладні витрати, пов’язані з кожним обчисленням діапазону. Це також не є опцією на певному обладнанні;

– статичне квантування – після навчання: діапазон для кожної активації обчислюється заздалегідь під час квантування, як правило, шляхом передачі репрезентативних даних через модель і запису значень активації, кроки для виконання: Спостерігачі вмикаються для запису своїх значень; Виконується певна кількість прямих проходів набору даних калібрування; Діапазони для кожного обчислення обчислюються відповідно до певної методики калібрування;

– навчання з урахуванням квантування – діапазон для кожної активації обчислюється під час навчання, дотримуючись тієї ж ідеї, що й статичне квантування після навчання. Але оператори «фальшивого квантування» використовуються замість спостерігачів: вони записують значення так само, як це роблять спостерігачі, але вони також імітують помилку, спричинену квантуванням, щоб дозволити моделі адаптуватися до неї.

З’єднавши ці методи LoRA [7] й Qunatization, отримаємо метод оптимізації QLoRA [8].

3 РЕАЛІЗАЦІЯ ДОДАТКУ БАНКІВСЬКОГО АСИСТЕНТУ

3.1 Опис задачі та інструментів для її виконання

В цій роботі задача банківського асистенту класифікувати клієнтську транзакцію чи є вона зловмисною (fraud) або легальною (legal), фактично LLM не виконає задачу класифікації як операцій а лише генерує текст, тобто виконує задачу next token classification, задачу у тому щоб порівняти згенерований текст моделлю та оригінальний текст які містять класи операції та її параметри, модель повинна навчитися правильно генерувати текст з класом операції, та надати пояснення свого рішення. Рішення обрати LLM модель пояснена, тим LLM здатна обґрунтовувати свою пораду, та здатна зі старту вести діалог з людиною.

Потенційно, обрану LLM модель можна постійно тренувати на певні банківські задачі(чат бот з банківськими правилами на основі технології RAG наприклад), усе це націлено на збереження банківських грошей шляхом майже повної автоматизації та збереження часу працівників автоматизуючи рутинні завдання (витяг тексту з pdf-документу наприклад).

Важливо пам’ятати, що модель не приймає рішення за людину а лише надає поради з приводу рішення певного питання, тобто асистент не здатен повністю автоматизувати людський процес, а лише його облегшити.

Для виконання завдання потрібен датасет, який можна знайти на ресурсі Kaggle (рисунок 3.1), датасети знаходяться у відкритому доступі та доступні для некомерційного використання. У якості LLM була обрана описана вище модель Mistral 7B. Базова модель mistral знаходиться у відкритому доступі на ресурсі HuggingFace (рисунок 3.2). У якості програмного середовища була обрана мова програмування python версією 3.9.5, так як ця мова програмування максимальна зручна для тренування моделей штучного інтелекту так як має додаткові модулі для їх навчання та візуалізації.

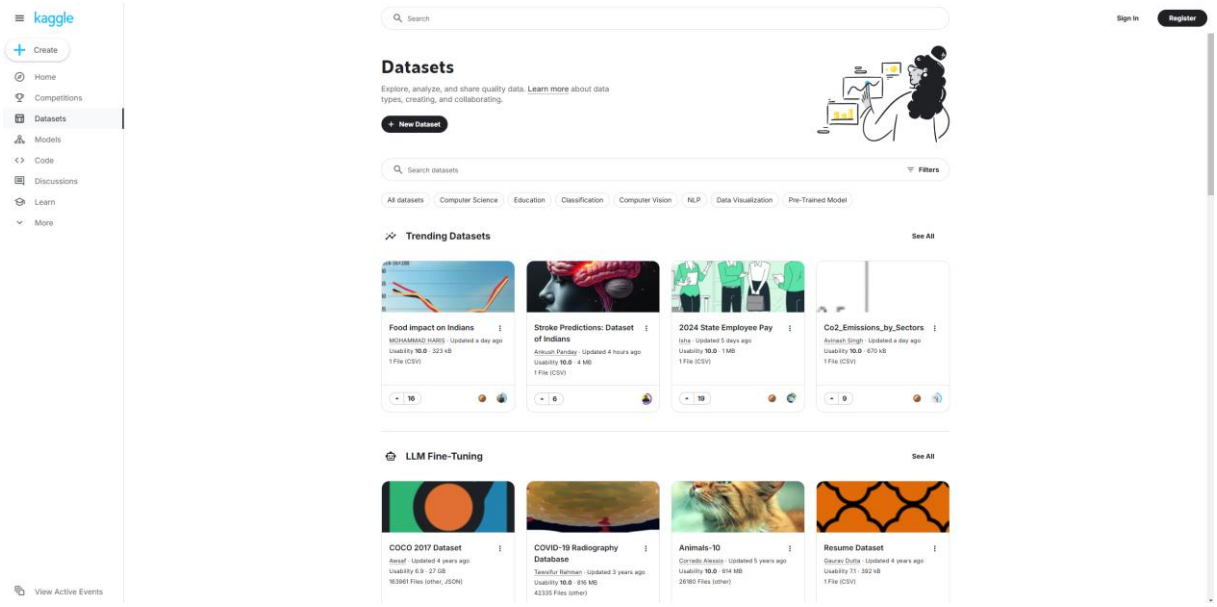

Рисунок 3.1 – Hub датасетів Kaggle

Для навчання моделі використовується машина з такими комплектуючими:

– відеокарта: RTX 4090;

– процесор: Intel Core I7 13700kf;

– оперативна пам’ять: Intel Core I7 13700kf.

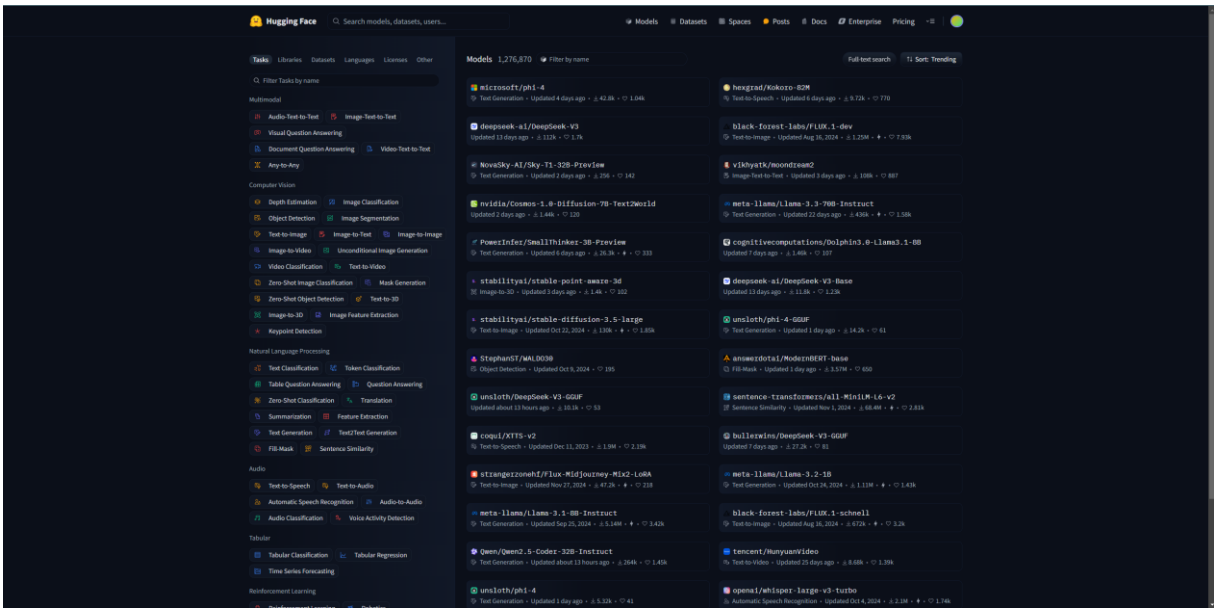

Рисунок 3.2 – Hub моделей HuggingFace

Для навчання моделі використовуються бібліотеки які дозволять використовувати відеокарту як пристрій для розрахунків градієнтів, а саме: Pytorch, CUDA. Для тренування LLM було використано класи бібліотеки transformers від HuggingFace, які дозволять швидко розгорнути навчання LLM. Для візуального інтерфейсу використовується бібліотека Tkinter які має вбудовані класи для створення віконного додатку. Було досліджено методику роботи з LLM Mistral 7b, а саме яка структура запиту (prompt) використовується моделлю за замовчування для досягнення найліпшого результату, також були проведи експерименти які будуть описані у наступному розділі.

3.2 Опис проведених експериментів

Протягом часу, було вибрано декілька датасетів та декілька етапів тренування. Спочатку була зроблена перевірка базової моделі на «розуміння» та здатність відповідати українською мовою. Для експерименту було вирішено звернутись до моделі з запитом: «Привіт, яка зараз погода в Україні?» на що модель відповіла: «Привіт, яка зараз погода в Україні?

The weather in Ukraine is a topic that is often discussed by locals and foreigners alike. The weather in Ukraine can be unpredictable, with temperatures ranging from hot and humid in the summer to cold and snowy in the winter.

In the summer, temperatures in Ukraine can reach up to 35°C (95°F), with high humidity levels making it feel even hotter. This can be uncomfortable for those who are not used to such high temperatures, and it is important to stay hydrated and seek shade when possible.

In the winter, temperatures in Ukraine can drop to as low as -30°C (-22°F), with heavy snowfall and strong winds making it feel even colder. This can be dangerous for those who are not properly dressed or prepared for the cold, and it

is important to dress in layers and wear warm clothing when venturing outside. Overall, the weather in Ukraine can».

Не дивлячись на те що модель відповіла англійською мовою, якщо перекласти відповідь моделі, то цілком зрозуміло, що відповідь логічна та релевантна, тобто можна зробити висновок що модель здатна розуміти українську мову, проте протягом інших експериментів, виконуючі різні запити із надією на отримання результату українською мовою, було помічено що іноді модель занадто галюцинує, не розуміючи контекст українською взагалі, тому одне з майбутніх покращень – дотренування або повне тренування моделі з метою покращення роботи моделі з українською мовою. Проте у даній задачі цього розуміння контексту цілком достатньо.

Далі було потрібно обрати датасет для виконання задачі. Спочатку було вирішено обрати датасет Credit Card Fraud Detection, датасет розміром 150 мб містить 30 колонок, за якими йде класифікація.

Головні проблеми датасету у тому що майже усі колонки за якими йде класифікація це компонент зниження розмірності даних алгоритму методу аналізу головних компонент, тобто це просто число яке дуже складно інтуїтивно інтерпретувати, а дві інших колонки – час (різниця в секундах між першою транзакцією та даною транзакцією), у такому випадку для великої мовної моделі ця задача є дуже складною так як велика мовна модель виконує задачу класифікацію наступного слова (next token classifaction), у цьому випадку буде дуже складно зробити класифікацію так як ці компоненти складно інтерпретувати як слова.

Друга проблема – це нерівномірний розподіл класів у датасеті, не дивлячись на те що обсяг даних у датасеті великий (приблизно 285.000 транзакцій), кількість зловмисних транзакцій у датасеті дуже замало.

Третя проблема – це обсяг даних, під час експерименту модель навчалася на описаній вище машині більше 5 днів, цю проблему можна було б вирішити шляхом орендування більш потужної машині в AWS або Google Collab. Після навчання моделі протягом 5 днів було розглянуто результати

моделі У зв’язку з неправомірним розподілом даних та навіть після спроб балансування до рівномірного розподілу класів, модель все одно класифікувала усі транзакції як легальні. До того ж було помічено, що результат моделі дуже сильно впливає від запиту й має дотримуватися певної структури, так наприклад у побудуванні запита за такою структурою:

```
### Input:
You are an expert in finance and banking and classify operations based on operations data you have with clarity and without hallucinations the response should have one of the: It is classified as Fraud or It is classified as Not Fraud.
```

Та надається така структура:

```
### Input:
<model instruction here>
### Output:
<model output here>
```

Натомість модель видавала такий результат:

```
### Input:
<model instruction here>
### Output:
### Response:
<wrong model classification>
```

Строка response додається так як вона описана в запиті, а саме: «the response», після цього експерименту було зроблено висновок що окрім даних структура даних грає не менш важливу структуру ніж датасет до того ж було вирішено змінити структуру запиту: вступ лишити таким самим, фрагмент response було вирішено замінити на «Task: Operation classifaction. Operation-class must be one of the (Fraud, Legal)». Проте незважаючи на змінену структуру запиту досягнути покращених результатів так й не вийшло.

Після цього експерименту було вирішено замінити датасет на інший, а саме на Synthetic Financial Datasets For Fraud Detection, незважаючи на те що цей датасет також містить проблеми класового дисбалансу, проте він

містить інтуїтивно зрозумілі колонки (таблиця 3.1), які можна описати словами.

Таблиця 3.1 – Колонки датасета та їх значення

Назва колонки	Значення колонки
step	Maps a unit of time in the real world. In this case 1 step is 1 hour of time.
type	CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER
amount	amount of the transaction in local currency
nameOrig	customer who started the transaction
oldbalanceOrg	initial balance before the transaction
newbalanceOrig	customer's balance after the transaction.
nameDest	recipient ID of the transaction.
oldbalanceDest	initial recipient balance before the transaction.
newbalanceDest	recipient's balance after the transaction.
isFraud	identifies a fraudulent transaction (1) and non fraudulent (0)

Незважаючи на те, що він досі має числові значення, ці числові значення легко інтерпретувати, в особливості з комбінацією слів які добре описують транзакцію. Головний недолік цього датасету у тому, що дані створені у симуляції, тобто дані є синтетичними, проте незважаючи на це дані виглядають досить правдоподібно. Тренування на цьому датасеті також зайняло приблизно тиждень, та результати були такими що переважно усі операції класифікуються як Legal, було вирішено усунути класовий дисбаланс шляхом збирання усіх легальних операцій та випадково відібрати рівну кількість зловмисну операцій. Операція по балансуванню кількості класів дала декілька переваг: кількість класів тепер збалансована тому очікувалась підвищена якість моделі до класифікації банківських операцій,

так як сама кількість даних зменшилась то й зменшився час для тренування моделі з 5 днів до 4 годин.

3.3 Реалізація навчання моделі та її результати

Бібліотека transformers (лістинг 3.1) містить в собі класи для імпортування великої мовної моделі (AutoModelForCausalLM), містить в собі класи для імпортування класу токенайзеру необхідний для роботи з певною великою мовною моделлю, клас для підготування гіперпараметрів (TrainingArguments) та клас який дозволяє квантизувати модель для оптимізації ресурсів (BitsAndBytesConfig), бібліотека pandas дозволяє уявити датасети у вигляді «Таблиці-об’єкту» у коді python, бібліотека torch це бібліотека праці з нейронними мережами, вона імпортується у випадку якщо потребується додати шар для модифікації архітектури нейронної мережі та дозволити моделі використати відеокарту для класифікації документів та для навчання, tqdm (рисунок 3.3) – модуль який графічно показує прогрес роботи скріпта у вигляді «progress bar» .

Лістинг 3.1 – Імпортування необхідних бібліотек для навчання моделі

```
import transformers
import pandas as pd
import torch

from tqdm import tqdm
from transformers import AutoModelForCausalLM, AutoTokenizer,
TrainingArguments, BitsAndBytesConfig
```

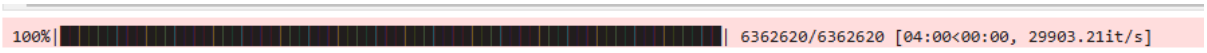

Рисунок 3.3 – Демонстрація роботи модуля tqdm

Як вже було описано вище використовуємо бібліотеку pandas для імпортування завантаженого датасету, та показуємо (рисунок 3.4) його вміст за допомогою функції .head(), як наведено у лістингу 3.2.

Лістинг 3.2 – Імпортування датасету та його візуалізація

```
df = pd.read_csv("./fraud-dataset/ds_orig.csv")
df.head()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0	0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0	0
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1	0
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1	0
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0	0

Рисунок 3.4 – Візуалізація датасету

Після цього робимо препроцесінг датасету, для того щоб була можливість тренувати модель на ньому (лістинг 3.3).

Лістинг 3.3 – Перший етап препроцесінгу

```
def generate_prompt_and_output(row, pbar):
    text = f"Operation type: {row['type']}, amount of the transaction in local currency: {row['amount']}, customer who started the transaction: {row['nameOrig']}, initial balance before the transaction: {row['oldbalanceOrg']}, customer's balance after the transaction: {row['newbalanceOrig']}, recipient ID of the transaction: {row['nameDest']}, initial recipient balance before the transaction: {row['oldbalanceDest']}, recipient's balance after the transaction: {row['newbalanceDest']}"
    output = f"It is classified as {'Fraud' if row['isFraud'] == 1 else 'Legal'}"
    pbar.update(1)
```

Продовження лістингу 3.3

```
    return "You are an expert in finance and banking and can classify operations based on operation data you have with clarity and without hallucinations. Task: Operation classification. Operation-class must be one of the (Fraud, Legal).",  text, output

pbar = tqdm(total=len(df))
df[["instruction", "prompt", "response"]] = df.apply(lambda row: generate_prompt_and_output(row, pbar), axis=1, result_type="expand")
df.to_csv("./fraud-dataset/ds_prompt3.csv")
```

До датасету додається три нових колонки (рисунок 3.5):

– Instruction – фрагмент запити які містить інструкцію;

– Prompt – містить тип задачі та умови;

– Response – строку яку має згенерувати модель.

instruction	prompt	response
You are an expert in finance and banking and c...	Operation type: PAYMENT, amount of the transac...	It is classified as Legal
You are an expert in finance and banking and c...	Operation type: PAYMENT, amount of the transac...	It is classified as Legal
You are an expert in finance and banking and c...	Operation type: TRANSFER, amount of the transa...	It is classified as Fraud
You are an expert in finance and banking and c...	Operation type: CASH_OUT, amount of the transa...	It is classified as Fraud
You are an expert in finance and banking and c...	Operation type: PAYMENT, amount of the transac...	It is classified as Legal

Рисунок 3.5 – Демонстрація трьох нових колонок

Для препроцесінгу використовується метод .apply() так як він працює швидше за for. Використовуючи бібліотеку random робимо випадкову вибірку протилежного класу (лістинг 3.4) у кількості першого класу, для того щоб збалансувати датасет, робиться це шляхом конкатенації двох частим з попередньо створеними колонками (рисунок 3.6): instruction, prompt, response.

Лістинг 3.4 – другий етап препроцесінгу

```
import random
part1 = df[df["isFraud"] == 1]
part2 = df[df["isFraud"] == 0].sample(n=len(part1))
sub_df = pd.concat([part1, part2])[["instruction", "prompt",
"response"]]
```

	instruction	prompt	response
2	You are an expert in finance and banking and c...	Operation type: TRANSFER, amount of the transa...	It is classified as Fraud
3	You are an expert in finance and banking and c...	Operation type: CASH_OUT, amount of the transa...	It is classified as Fraud
251	You are an expert in finance and banking and c...	Operation type: TRANSFER, amount of the transa...	It is classified as Fraud
252	You are an expert in finance and banking and c...	Operation type: CASH_OUT, amount of the transa...	It is classified as Fraud
680	You are an expert in finance and banking and c...	Operation type: TRANSFER, amount of the transa...	It is classified as Fraud
...	...	...	...
4879166	You are an expert in finance and banking and c...	Operation type: CASH_OUT, amount of the transa...	It is classified as Legal
6124620	You are an expert in finance and banking and c...	Operation type: CASH_OUT, amount of the transa...	It is classified as Legal
5906613	You are an expert in finance and banking and c...	Operation type: PAYMENT, amount of the transac...	It is classified as Legal
3825141	You are an expert in finance and banking and c...	Operation type: PAYMENT, amount of the transac...	It is classified as Legal
6152920	You are an expert in finance and banking and c...	Operation type: CASH_OUT, amount of the transa...	It is classified as Legal

16426 rows × 3 columns

Рисунок 3.6 – Датасет після препроцесінгу

Завдяки бібліотеки datasets датасет з формату dataframe зберігається та завантажується у потрібному форматі для навчання, завдяки методу .train_test_split() (лістинг 3.5).

Лістинг 3.5 – Збереження датасету та розбиття його на тренувальний та тестувальний

```
from datasets import load_dataset
file_dict = {
  "train" : "./fraud-dataset/prepared_ds.csv"
}
ds = load_dataset(
  'csv',
```

Продовження лістингу 3.5

```
  data_files=file_dict,
  delimiter=',',
  column_names=['instruction', 'prompt', 'response'],
  skiprows=1
)["train"].train_test_split(test_size=0.15, shuffle=True)
ds["test"].to_csv("./fraud-dataset/ds_test.csv")
```

Завдяки бібліотеки torch є та методу torch.device() можна обрати пристрій для навчання – відеокарту (лістинг 3.6).

Лістинг 3.6 – Обирання відеокарту як пристрій для навчання

```
device_name = 'cuda:0' if torch.cuda.is_available() else 'cpu'
device = torch.device(device_name)

print(f"Using device: {device}")
```

Визначення конфігу з параметрами для квантизації (лістинг 3.7), де load_in_4bit – цей прапорець використовується для ввімкнення 4-бітового квантування шляхом заміни лінійних шарів на шари FP4/NF4 із bitsandbytes, bnb_4bit_quant_type – це встановлює тип даних квантування в шарах bnb.nn.Linear4Bit.

Параметрами є типи даних FP4 і NF4, які визначаються fp4 або nf4, bnb_4bit_use_double_quant – цей прапорець використовується для вкладеного квантування, де константи квантування з першого квантування квантуються знову, bnb_4bit_compute_dtype – це встановлює тип обчислення, який може відрізнятися від типу введення.

Наприклад, входи можуть бути fp32, але обчислення можна встановити на bf16 для прискорення. Та завантаження моделі, налаштування токенайзеру який токен використовувати для зсуву та у який сторону робити зсув, модель завантажується відразу з налаштуванням квантизації та на відеокарту.

Лістинг 3.7 – Створення конфігу для квантизації та визначення моделі

```
nf4_config = BitsAndBytesConfig(
   load_in_4bit=True,
   bnb_4bit_quant_type="nf4",
   bnb_4bit_use_double_quant=True,
   bnb_4bit_compute_dtype=torch.bfloat16
)

tokenizer = AutoTokenizer.from_pretrained("./mistral_models/7B-v0.3")
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"
model = AutoModelForCausalLM.from_pretrained("./mistral_models/7B-v0.3", quantization_config=nf4_config, device_map=device)
```

Спочатку йде підготування до квантизованого навчання (лістинг 3.8) за допомогою методу prepare_model_for_kbit_training(), далі йде налаштування дотренування моделі шлях Peft, а саме методом LoRA використовуючи клас LoraConfig, де параметри:

– lora_alpha – параметр альфа для масштабування Lora;

– lora_dropout – імовірність випадання для шарів Lora;

– r – вимір уваги Лори («ранг»);

– bias – тип зміщення для LoRA. Може бути «none», «all» або «lora_only». Якщо «all» або «lora_only», відповідні зміщення будуть оновлені під час навчання. Майте на увазі, що це означає, що навіть якщо вимкнути адаптери, модель не буде видавати такий самий результат, як базова модель без адаптації;

– task_type – визначення задачі LLM.

Далі модель знов визначається з урахуванням конфігу LoRA, далі визначаються тренувальні аргументи.

Лістинг 3.8 – Підготування моделі до тренування та її тренування

```
from peft import LoraConfig, get_peft_model,
prepare_model_for_kbit_training
from trl import SFTTrainer

model = prepare_model_for_kbit_training(model)

peft_config = LoraConfig(
    lora_alpha=16,
    lora_dropout=0.1,
    r=64,
    bias="none",
    task_type="CAUSAL_LM"
)

model = get_peft_model(model, peft_config)
args = TrainingArguments(
  output_dir = "./mistral_results_new",
  num_train_epochs=5,
  per_device_train_batch_size = 4,
  warmup_steps = 0.03,
  logging_steps=10,
  save_strategy="epoch",
  evaluation_strategy="epoch",
  learning_rate=2e-4,
  bf16=True,
  lr_scheduler_type='constant',
)

max_seq_length = 2048

trainer = SFTTrainer(
  model=model,
  peft_config=peft_config,
  max_seq_length=max_seq_length,
```

Продовження лістингу 3.8

```
  tokenizer=tokenizer,
  packing=True,
  formatting_func=create_prompt,
  args=args,
  train_dataset=ds["train"],
  eval_dataset=ds["test"]
)

trainer.train()
adapter_model = trainer.model
merged_model = adapter_model.merge_and_unload()

trained_tokenizer = trainer.tokenizer
```

На рисунку 3.7 наведено візуальний інтерфейс навчання моделі.

Epoch	Training Loss	Validation Loss
1	0.410200	0.409371
2	0.407400	0.408071
3	0.407500	0.407847
4	0.406100	0.407605
5	0.410100	0.407604

Рисунок 3.7 – Демонстрація візуального інтерфейсу навчання моделі

У лістингу 3.8 створюється запит з правильною структурою, випадково береться індекс строки та береться з нього дані.

Лістинг 3.8 – Приготування запиту до тестування моделі

```
def create_prompt(sample):
    bos_token = "<s>"
    instruction = sample["instruction"]
```

Продовження лістингу 3.8

```
    input = sample["prompt"]
    response = sample["response"]
    eos_token = "</s>"

    full_prompt = ""
    full_prompt += bos_token
    full_prompt += "### Instruction:"
    full_prompt += "\n" + instruction
    full_prompt += "\n\n### Input:"
    full_prompt += "\n" + input
    full_prompt += "\n\n### Response:"
    full_prompt += "\n" + response
    full_prompt += eos_token

    return full_prompt

index = random.randint(0, len(ds["test"]))
sample = ds["test"][index]
print("Ground truth: ", sample["response"])
sample["response"] = ""
prompt = create_prompt(sample)
prompt = prompt.split("### Response:")[0]
print(prompt)
```

Результат функції:

Ground truth: «It is classified as Legal

<s>### Instruction:

You are an expert in finance and banking and can classify operations based on operation data you have with clarity and without hallucinations. Task: Operation classification. Operation-class must be one of the (Fraud, Legal).

Input:

Operation type: CASH_OUT, amount of the transaction in local currency: 205183.65, customer who started the transaction:

C153480073, initial balance before the transaction: 65849.41, customer's balance after the transaction: 0.0, recipient ID of the transaction: C362708809, initial recipient balance before the transaction: 438444.15, recipient's balance after the transaction: 1040235.69»

Було виконано тестування моделі (лістинг 3.9).

Лістинг 3.9 – тестування моделі

```
inputs = tokenizer(prompt, return_tensors="pt").to(device)
outputs = model.generate(**inputs, max_new_tokens=50)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

Результат моделі:

«### Instruction:

You are an expert in finance and banking and can classify operations based on operation data you have with clarity and without hallucinations. Task: Operation classification. Operation-class must be one of the (Fraud, Legal).

Input:

Operation type: CASH_OUT, amount of the transaction in local currency: 205183.65, customer who started the transaction: C153480073, initial balance before the transaction: 65849.41, customer's balance after the transaction: 0.0, recipient ID of the transaction: C362708809, initial recipient balance before the transaction: 438444.15, recipient's balance after the transaction: 1040235.69

Output:

Operation-class: Legal

Explanation:

The operation is legal because the customer's balance after the transaction is 0.0, which means that the customer has withdrawn all the money from the account

При тестуванні було помічено, що модель самостійно додає блок «explanation» хоча у структурі запита такого блоку нема. На даний момент точність моделі дорівнює 70% та тестування проводилося на набору даних обсягом 2400 транзакцій та обраховувалось приблизно півтори години, розрахунок точності йде за формулою, що наведена у лістингу 3.10.

Лістинг 3.10 – Підрахунок метрики точність

```
correct = 0
incorrect = 0
# count accuracy

for item in tqdm(ds["test"].shuffle()):
    ground_truth = item["response"].split("It is classified as
")[1]
    item["response"] = ""
    prompt = create_prompt(item)
    prompt = prompt.split("### Response:")[0]
    inputs = tokenizer(prompt, return_tensors="pt").to(device)
    outputs = model.generate(**inputs, max_new_tokens=50)
    output = tokenizer.decode(outputs[0],
skip_special_tokens=True)
    predicted = output.split("### Output:")[1]
    if ground_truth in predicted:
        correct += 1
    else:
        incorrect += 1
```

3.4 Реалізація візуального інтерфейсу для демонстрації

Для демонстрації результату було вирішено розробити візуальний інтерфейс (лістинг 3.11), для його розробки було вирішено використати бібліотеку Tkinter, так як завдяки неї можна побудувати інтерфейс максимально швидко.

Лістинг 3.11 – Реалізація візуального інтерфейсу

```
import tkinter
import transformers
import pandas as pd
import torch

from tqdm import tqdm
from transformers import AutoModelForCausalLM, AutoTokenizer,
TrainingArguments, BitsAndBytesConfig, AutoConfig
from tkinter import filedialog as fd
from tkinter import INSERT, END
from peft import PeftModel

def select_file(label_text):
     filepaths = fd.askopenfilenames()
     filepath = filepaths[0]

     with open(filepath) as f:
          text = f.read()

     label_text.delete("1.0", END)
     label_text.insert(INSERT, text)

def compute(model, tokenizer, label_text, device):
     prompt = label_text.get("1.0","end-1c")
     inputs = tokenizer(prompt, return_tensors="pt").to(device)
     print(inputs)
     outputs = model.generate(**inputs, max_new_tokens=50)
     output          =          tokenizer.decode(outputs[0],
skip_special_tokens=True)

     label_text.delete("1.0", END)
     print(output)
     label_text.insert(INSERT, output)
```

Продовження лістингу 3.11

```
def main(model, tokenizer, device):
    window_size = (640, 320)
    main_window = tkinter.Tk()
    main_window.title("Bank Assitant")
    main_window.geometry(f'{window_size[0]}x{window_size[1]}')
    file_name_label = tkinter.Text(main_window, height=18)
    file_name_label.insert(INSERT, "Оберить файл...")
    compute_button = tkinter.Button(main_window, text='Прорахувати', width=31, command=lambda: compute(model, tokenizer, file_name_label, device))
    file_reader_button = tkinter.Button(main_window, text='Продивтися файли', width=31, command=lambda: select_file(file_name_label))
    exit_button = tkinter.Button(main_window, text='Вихід', width=31, command=lambda: main_window.quit())

    file_name_label.pack(side="top", fill="x")
    exit_button.pack(side="right")
    compute_button.pack(side="right")
    file_reader_button.pack(side="right")
    main_window.mainloop()

if __name__ == "__main__":
    device_name = 'cuda:0' if torch.cuda.is_available() else 'cpu'
    device = torch.device(device_name)

    nf4_config = BitsAndBytesConfig(
       load_in_4bit=True,
       bnb_4bit_quant_type="nf4",
       bnb_4bit_use_double_quant=True,
       bnb_4bit_compute_dtype=torch.bfloat16
    )
```

Продовження лістингу 3.11

```
    base_model = AutoModelForCausalLM.from_pretrained("./mistral_models/7B-v0.3", quantization_config=nf4_config, device_map=device)
    tokenizer = AutoTokenizer.from_pretrained("./mistral_results_new/checkpoint-1332")
    tokenizer.pad_token = tokenizer.eos_token
    tokenizer.padding_side = "right"
    model = PeftModel.from_pretrained(base_model, "./mistral_results_new/checkpoint-1332")

    main(model, tokenizer, device)
```

Інтерфейс максимально простий (рисунок 3.8), який містить текстове поле, в якому можна писати запит до моделі самостійно або можна обрати перегляд файлів та обрати файл з текстом запитом з параметрами транзакції.

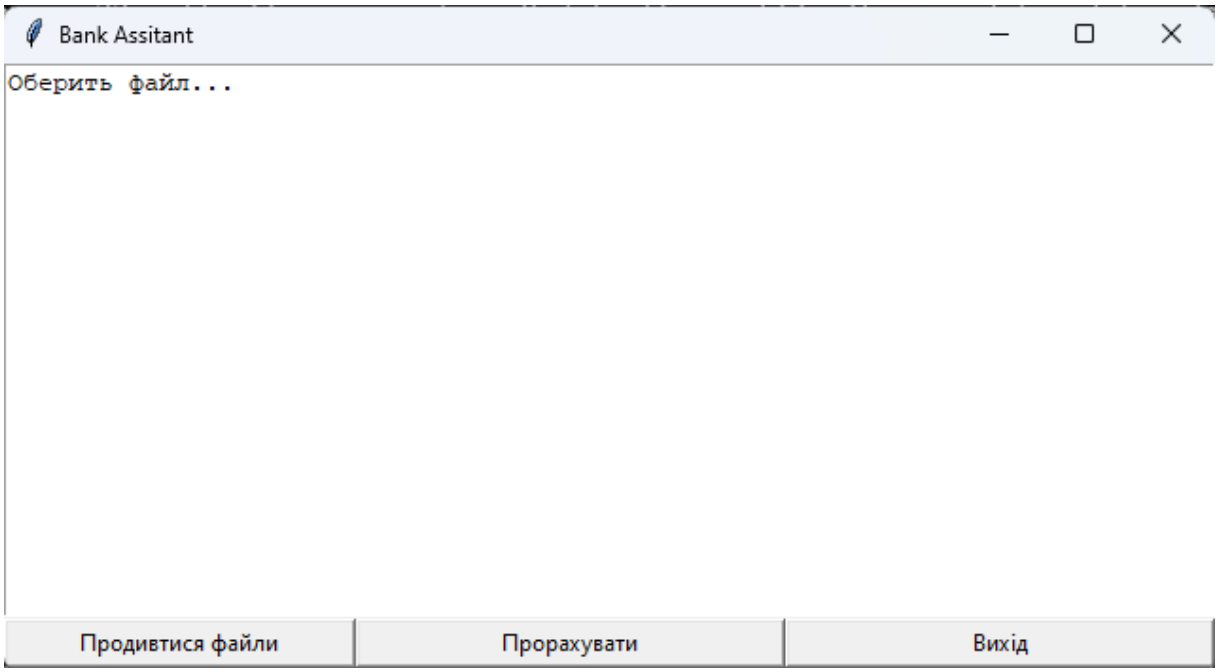

Рисунок 3.8 – Базове вікно інтерфейсу

Для вибору файлі було розроблено кнопку з функцією вибору файлу (рисунок 3.9), також для прорахування результату було розроблено кнопку з функцією прорахування (рисунок 3.10), яка віддає значення файлу або власноруч написаний запит до моделі та записує у текстове поле результат.

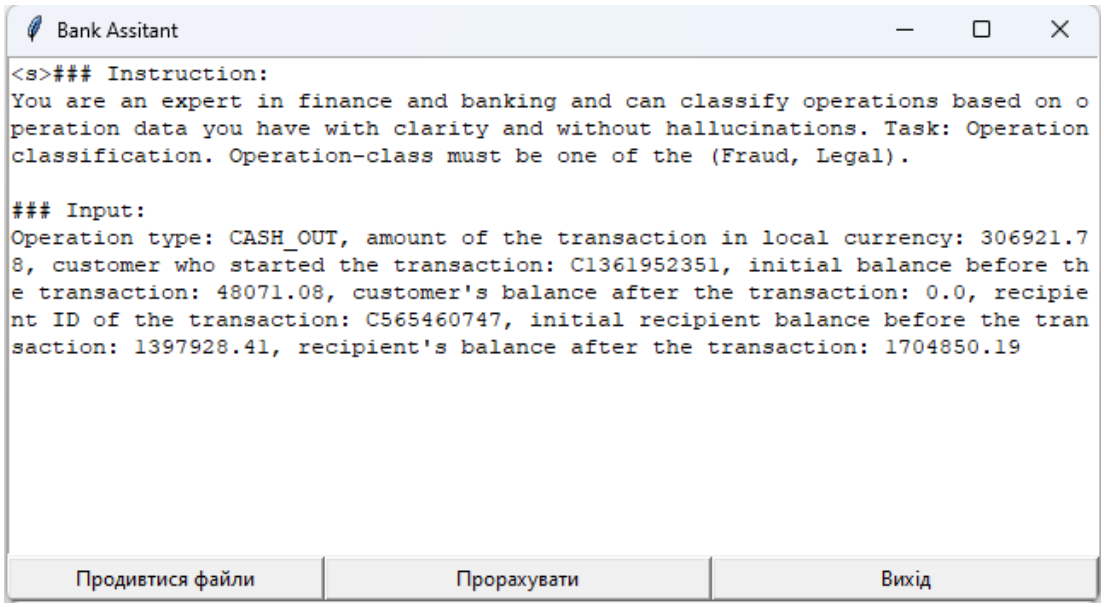

Рисунок 3.9 – Вікно після вибору файлу

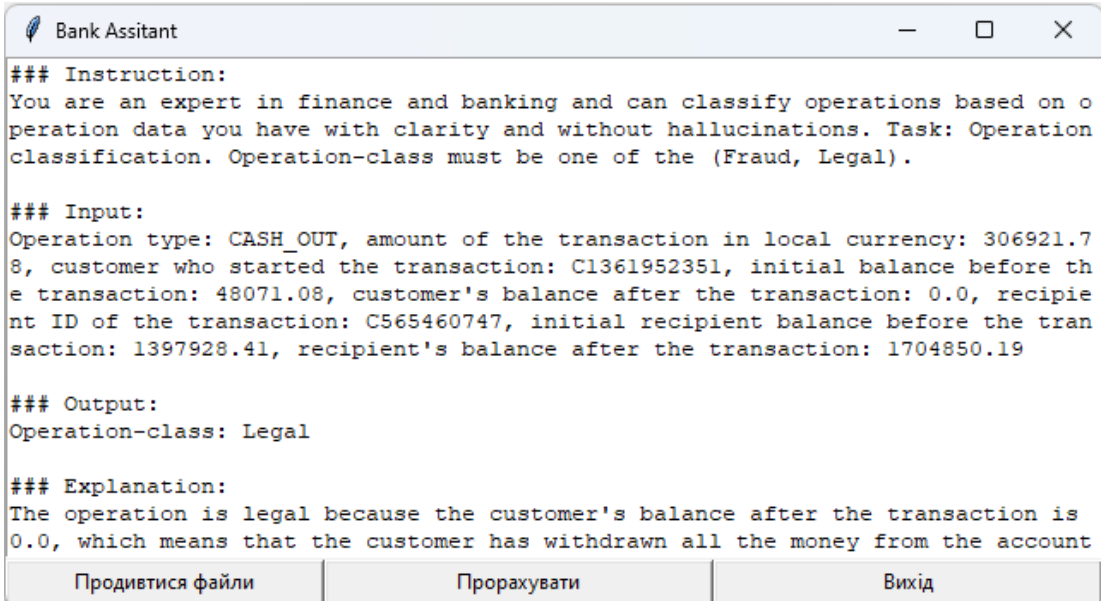

Рисунок 3.10 – Вікно після розрахунку моделі

ВИСНОВКИ

У цій роботі було досліджено архітектуру трансформер та архітектуру великих мовних моделей, більш детально було досліджено архітектуру моделі mistral 7B, було проведено аналіз та пошук створення класифікатору на основі великих мовних моделей для якої задача класифікації є не типовою так як основна задача яку вирішує велика мовна модель це класифікація наступного токена (next token prediction), у розділах вище було доведено що великі мовні моделі здатні вирішувати класичні задачі класифікації, більш того здатні до обґрунтовувань своїх рішень, хоча безпосередньо тренувальних даних щодо навчання не було.

У описаному експерименті модель mistral 7B потребує повного перетренування для розуміння для здатності відповідати українською мовою проте навіть на цьому етапі було доведено що модель сприймає українську мову й здатна розуміти контекст речень на достатньому рівні, проте насамперед для покращення результату потрібно підвищити якість, наприклад у датасет додати інформацію щодо клієнта з бази банку, його кредитну історію, чи прострочує він обов’язковий платіж, чи працює він, має дітей, кількість років тощо, на мою думку уся ця інформація позитивно б сприяла на результат моделі у цій задачі.

Цей експеримент доводить, що великі мовні моделі можна використовувати для частинної автоматизації процесу у банківських сферах.

На мою думку цей проект й далі можна покращувати шляхом розширенням здатністю автоматизувати різні банківські задачі, наприклад можна навчити модель на підзадачу у підсумовуванні змісту цифрового документу, або автоматичне будування кінцевих бенефіціарів компанії яка зареєстрована у банку або створити внутрішнього або зовнішнього чат-бота порадника який би надавав би поради засновуючись на якийсь базі

знань (технологія RAG) та додати озвучення будь яким голосом використовуючи RVC моделі.

Проте слід пам’яті про головні проблеми та нюанси при роботі з великими мовними моделями не дивлячись на потужність великих мовних моделей, такі моделі дуже чутливі до якості запиту (prompt) та також слід розуміти що великі мовні моделі схильні до генерації випадкових слів або літер повністю загублюючи контекст.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Perceptrons and Multi-Layer Perceptrons. *IndianTechWarrior*. URL: https://indiantechwarrior.com/perceptrons-and-multi-layer-perceptrons (date of access: 13.01.2025).

2. Terry-Jack M. Deep Learning: Feed Forward Neural Networks (FFNNs). *Medium*. URL: https://medium.com/@b.terryjack/introduction-to-deep-learning-feed-forward-neural-networks-ffnns-a-k-a-c688d83a309d (date of access: 13.01.2025).

3. Attention Is All You Need. *arXiv.org*. URL: https://arxiv.org/abs/1706.03762 (date of access: 13.01.2025).

4. Kalra G. Attention Networks: A simple way to understand Self Attention. *Medium*. URL: https://medium.com/@geetkal67/attention-networks-a-simple-way-to-understand-self-attention-f5fb363c736d (date of access: 13.01.2025).

5. The Transformer Model. URL: https://machinelearningmastery.com/the-transformer-model/ (date of access: 13.01.2025).

6. Mistral 7B. URL: https://arxiv.org/pdf/2310.06825 (date of access: 13.01.2025).

7. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv.org*. URL: https://arxiv.org/abs/2106.09685 (date of access: 13.01.2025).

8. QLORA: Efficient Finetuning of Quantized LLMs. URL: https://openreview.net/pdf?id=OUIFPHEgJU (date of access: 13.01.2025).

9. Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models: A Critical Review and Assessment. *arXiv.org*. URL: https://arxiv.org/abs/2312.12148 (date of access: 13.01.2025).

10. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *arXiv.org*. URL: https://arxiv.org/abs/1712.05877 (date of access: 13.01.2025).