МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК І ТЕХНОЛОГІЙ

(повне найменування факультету)

Кафедра «Системний аналіз та обчислювальна математика»
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему Файнтюнінг великих мовних моделей для спеціалізованих чат- ботів

(назва теми)

Виконав(ла): студент(ка) _3_курсу, групи_КНТ-812сп
Спеціальності _124 – Системний аналіз
(код і найменування спеціальності)

Освітня програма (спеціалізація)
«Інтелектуальні технології та прийняття рішень
в складних системах»

_____УВАРОВ К.Ф_____
(ПРІЗВИЩЕ та ініціали)

Керівник _____ШИРОКОРАД Д. В._____
(ПРІЗВИЩЕ та ініціали)

Рецензент_____ДУМІН О. М._____
(ПРІЗВИЩЕ та ініціали)

2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет___124 — Системний аналіз___
Кафедра_____«Системний аналіз та обчислювальна математика»_____
Ступінь вищої освіти _____бакалавр_____
Спеціальність_____124 – Системний аналіз_____
(код і найменування)
Освітня програма (спеціалізація)____«Інтелектуальні технології та прийняття рішень в складних системах»____
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри___САОМ____
____ТЕРЕЩЕНКО Е. В._____
«__16___» ___червня_____20_25__року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

_____УВАРОВА Костянтина Федоровича_____
(ПРІЗВИЩЕ, ім'я, по батькові)
1. Тема проєкту (роботи)__Файнтюнінг великих мовних моделей для спеціалізованих чат ботів_____
керівник проєкту (роботи)____ШИРОКОРАД Дмитро Вікторович_____,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від «16»__травня_20_25_року №265
2. Строк подання студентом проєкту (роботи)____16 черня 2025_____
3. Вихідні дані до проєкту (роботи)__Дані з наукових публікацій, статей з відкритих джерел, дані для навчання мовної моделі з відкритих джерел які відносяться до задачі
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_В першому розділі розглядаються питання актуальності роботи, теоретична частина та проблеми. В другому розглянуто методи навчання мовних моделей. В третьому розділі було проведено навчання мовної моделі для задачі та аналізовано модель з двома методами навчання__
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
_____
_____

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1	Широкорад Д.В.	20.04.2025	04.05.2025
2	Широкорад Д.В.	05.05.2025	20.05.2025
3	Широкорад Д.В.	21.05.2025	01.06.2025
Нормоконтроль	Широкорад Д.В.	12.06.2025	12.06.2025

7. Дата видачі завдання «_16_» _квітня___2025_року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Формування мети та завдання роботи	01.04.2025-16.04.2025	
2	Опрацювання інформації з наукової літератури та публікацій	17.04.2025-10.05.2025	
3	Отримання даних для завдання	11.05.2025	
4	Розробка програмної реалізації	12.04.2025-31.05.2025	
5	Аналіз результатів та розрахунки	31.05.2025-03.06.2025	
6	Оформлення пояснювальної записки	21.05.2025-13.06.2025	
7	Передзахист дипломної роботи	04.06.2025	
8	Захист дипломної роботи	16.06.2025	

Студент(ка) _____ ___УВАРОВ К. Ф._____

(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ ____ШИРОКОРАД Д. В._____

(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 52 стор., 16 рис., 8 джерел

Об’єкт дослідження – файнтюнінг моделей та подальше використання їх в інтелектуальних системах.

Предмет дослідження – ефективність навчання та використання ресурсів різних методів файнтюнінгу. Вплив параметрів генерації моделей на ефективність.

Мета дипломної роботи полягає у задачо орієнтованому навчанні мовних моделей ШІ для подальшого використання в чат-ботах та інших інтелектуальних системах.

Результати роботи представлені навчанням моделі методами файнтюнінгу та тестування. Для визначення ефективності методів був проведен аналіз результатів тестування методів з різнимим параметрами.

Ключові слова: ШТУЧНИЙ ІНТЕЛЕКТ, МОВНА МОДЕЛЬ, ФАЙНТЮНІНГ, АНАЛІЗ ЕФЕКТИВНОСТІ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект

ВММ – велика мовна модель

ММ – мовна модель

Файнтюнінг – задачо орієнтоване навчання

regex – regular expression

ВСТУП

Актуальність роботи:

Стрімкий розвиток методів штучного інтелекту, зокрема технологій обробки природної мови (NLP), відкриває нові можливості для створення інтелектуальних систем, здатних з високою точністю і низькою похибкою допомагати користувачам, аналітикам, інженерам та іншим фахівцям у розв’язанні широкого кола задач. Сучасні мовні моделі демонструють вражаючі результати у генерації тексту, пошуку інформації, а також у вирішенні складних математичних і наукових проблем.

Однак ці моделі зазвичай вимагають значних обчислювальних ресурсів — як з точки зору апаратного забезпечення (GPU, TPU, NPU), так і енергоспоживання. Це обмежує їхнє застосування у пристроях із невеликою обчислювальною потужністю або в умовах обмежених ресурсів.

У зв’язку з цим зростає інтерес до використання малих мовних моделей, які можна запускати на звичайних користувацьких комп’ютерах або вбудованих системах. Зокрема, актуальним є дослідження ефективних методів тонкого налаштування (файнтюнінгу), які дозволяють адаптувати такі моделі до конкретних задач — наприклад, обробки частих запитань — без значної втрати якості розуміння мови.

Ця робота спрямована на вивчення підходів до оптимізації малих мовних моделей для прикладних NLP-завдань з урахуванням обмежень обчислювальних ресурсів, що є надзвичайно актуальним у контексті зростаючого попиту на локальні, автономні та енергоефективні рішення.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ АВТОРЕГРЕСИВНОЇ ВЕЛИКОЇ МОВНОЇ МОДЕЛІ

1.1 Основні теоретичні відомості

Велика мовна модель або велика модель мови (ВММ або LLM від англ. large language model) — це модель, що складається з нейронної мережі з багатьма параметрами (від десятків мільйонів до мільярдів), навчених на великій кількості тексту за допомогою самокерованого або напівкерованого навчання

Ці моделі загального призначення, які відмінно справляються з широким спектром завдань, на відміну від навчання для одного конкретного завдання (наприклад, аналіз настроїв, розпізнавання іменованих об'єктів або математичне міркування)[9]

Параметри мовної моделі – це ваги та зміщення всередині нейронної мережі, які були вивчені (або "налаштовані") під час процесу навчання. Саме ці параметри кодують знання, отримані моделлю з навчальних даних. Чим більше параметрів має модель, тим потенційно складніші закономірності вона може вивчити та тим краще вона може узагальнювати інформацію на нових, раніше не бачених даних, хоча це також збільшує обчислювальні витрати на навчання та використання моделі.

Токен – це одиниця мови яку модель може генерувати та приймати як вхід.

Речення можна розбити на слова, де слово – токен, але для більш кращої обробки та знаходження більше закономірностей, використовують алгоритми які розбивають речення на різні частини. Для прикладу, речення “Привіт, як справи?”, алгоритм може розбити на такі токени: “Пр”, “ивіт”, “,”, “як”, “спр”, “ави”

Авторегресивні моделі є класом ВММ, які генерують текст послідовно,

один токен за раз. Ключова характеристика авторегресивної моделі полягає в тому, що передбачення кожного наступного токена залежить від попередньо згенерованих токенів. Іншими словами, модель "регресує" на власні попередні виходи.

Формально, якщо ми маємо послідовність токенів $x_1, \dots x_{t-1}$, авторегресивна модель моделює ймовірність появи наступного токена x_t за умови всіх попередніх токенів $x_1, \dots x_{t-1}$:

$$P(x_t | x_1, x_2, \dots x_{t-1}) \ .$$

Для генерації цілої послідовності тексту модель починає з деякого початкового контексту (можливо, спеціального токена "початок послідовності" або вхідного запиту користувача) і потім послідовно генерує токени один за одним.

Вихідні токени називають контекстом моделі для наступного токена.

На кожному кроці модель обчислює розподіл ймовірностей для всіх можливих наступних токенів у своєму словнику та обирає один з них (наприклад, найбільш ймовірний або шляхом семплювання з цього розподілу). Цей новообраний токен потім додається до послідовності, і процес повторюється для генерації наступного токена.

Найбільш використаною архітектурою для побудови авторегресивних мовних моделей є трансформер [1].

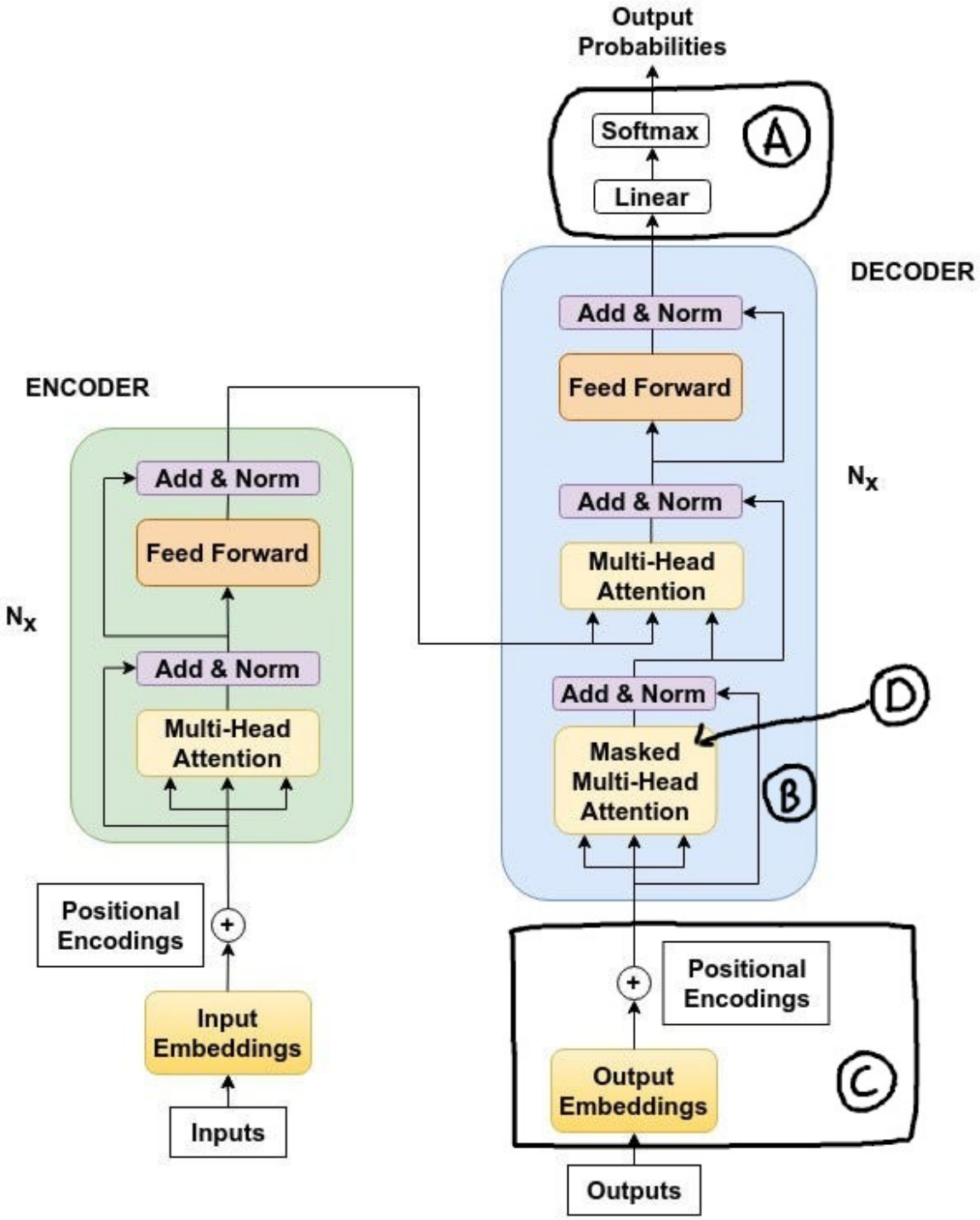

Рисунок 1.1 – Схема моделі трансформера

Для побудови сучасних великих мовних моделей використовується тільки права частина, а саме Decoder.

Спочатку, кожен токен перетворюється на багатовимірний вектор за допомогою словника токенів та нейронної мережі (рис. 1.1, C).

Після отримання багатовимірного вектору є позиційне кодування, для того щоб модель могла розрізняти однакові токени які знаходяться в різних місцях контексту.

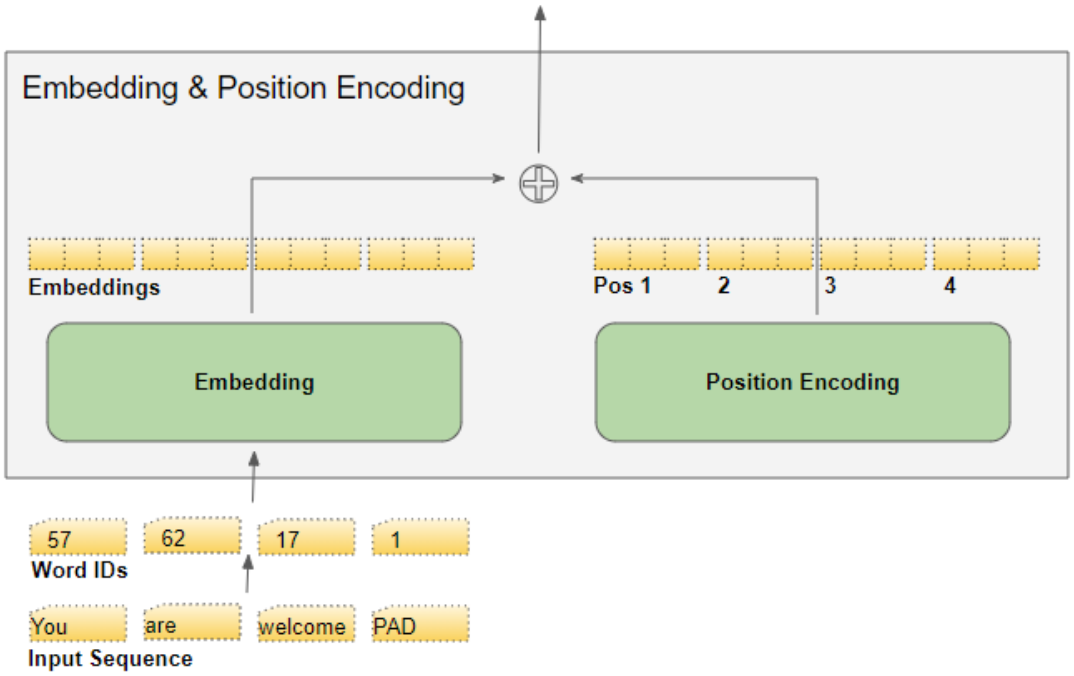

Рисунок 1.2 – Схема отримання кодованого контексту, багатовимірних векторів

1.2 Механізм уваги у великих мовних моделях на базі трансформер

Основною особливістю трансформера є механізм уваги (attention mechanism) (рис. 1.1, D).

Цей механізм знаходиться відразу після системи кодування токенів у вектори.

Attention mechanism, це механізм який полягає в знаходженні скалярних добутків між усіма токенами. Якщо скалярний добуток близький до нуля, то для моделі ці два токени не пов’язані між собою, але якщо близький до 1, то

для моделі вони пов’язані і матимуть вплив на наступний токен який генерується.

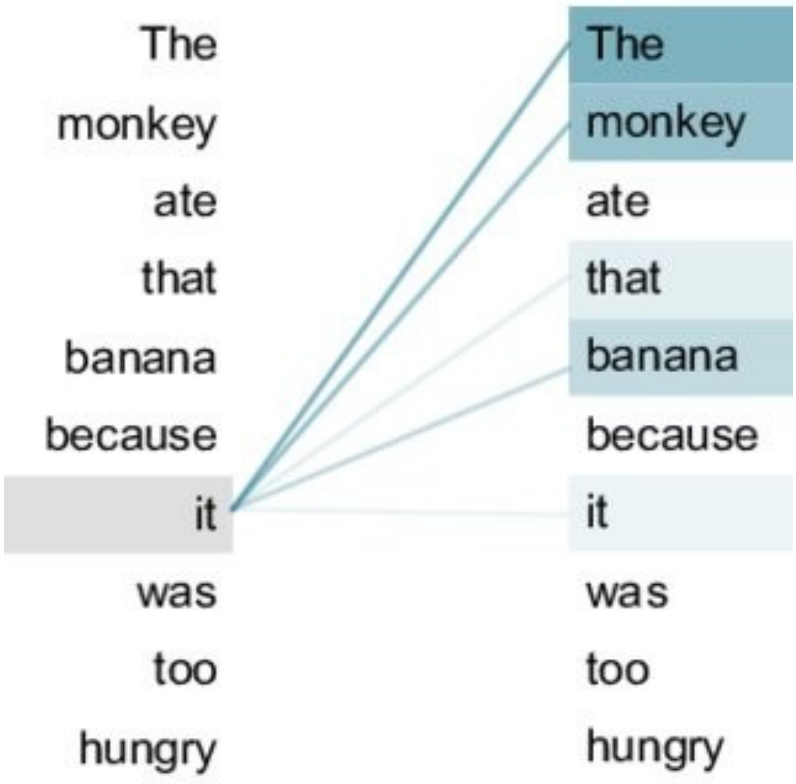

Рисунок 1.3 – Приклад механізму уваги

Як можна побачити з прикладу, токен 'it' пов’язаний більше з токенами 'the' та 'monkey'. Якщо це стосується першого шару уваги, то, найімовірніше, 'it' у даному випадку просто відноситься до суб’єкту 'monkey'. Однак, у більш глибоких шарах, взаємозв'язки між різними токенами стають набагато складнішими для інтерпретації.

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

де Q – матриця токенів, які були помножені на матрицю вагів W_q;

K – матриця ключів токенів, W_k;

V – матриця значень токенів, W_v.

Механізм уваги у більшості моделей має декілька шарів, які складаються з простих механізмів, тобто для кожного шару є окремі матрииці W_q, W_k та W_v.

Формально, механізм уваги працює таким чином:

в одному шарі є 3 * num_heads матриці $W_{iq(d_{model},d_k)}$, $W_{ik(d_{model},d_k)}$, $W_{iv(d_{model},d_k)}$

де:i – індекс механізму уваги в одному шарі;

d_k – розмір внутрішніх векторів, повинен бути d_model/num_heads;

d_model – розмір вхідного вектора токену;

num_heads – кількість механізмів уваги в одному шарі.

Для отриманих матриць векторів токенів з початку або з попереднього шару наступні операції:

$X_{(context_{size},d_{model})}$ – матриця векторів токенів.

$$Q_{i(context_{size},d_k)}=XW_{iq} \ , \quad K_{i(context_{size},d_k)}=XW_{ik} \ , \quad V_{i(context_{size},d_k)}=XW_{iv} \ .$$

Далі використовуючи формулу механізму уваги отримаємо:

$$Attention(Q_i,K_i,V_i)_{i(context_{size},d_k)}=softmax(\frac{Q_{i(context_{size},d_k)}K^T_{i(d_k,context_{size})}}{\sqrt{d_k}})V_{i(context_{size},d_k)} \ .$$

Тобто значення $V_{i(context_{size},d_k)}$, масштабується відповідно до скалярних добутків матриць K та Q.

Після отримання значень $V_{i(context_{size},d_k)}$ num_heads разів, отримаємо нову матрицю $V_{(context_{size},d_{model})}=\begin{bmatrix} Attention(Q_1,K_1,V_1)_{1(context_{size},d_k)}, \\ Attention(Q_2,K_2,V_2)_{2(context_{size},d_k)}, \\ \dots \\ Attention(Q_{num_{heads}},K_{num_{heads}},V_{num_{heads}})_{num_{heads}(context_{size},d_k)}, \end{bmatrix}$.

В кінці механізму уваги, потрібно зробити останнє перетворення за допомогою матриці $W_{o(d_{model},d_{model})}$.

$$X_{o(context_{size}, d_{model})} = V_{(context_{size}, d_{model})} W_{o(d_{model}, d_{model})}$$

Після чого нова матриця векторів $X_{o(context_{size}, d_{model})}$ переходить до наступного шару механізму уваги або в останній шар перетворення до тексту.

Таким чином, модель може мати різні концепти про значення векторів у різних шарах механізмів уваги, що посилює здатність моделі до вивчення складних текстів. Також незалежність механізмів уваги всередині одного шару надає можливість обробити токени паралельно, що робить модель більш швидкою при генерації у порівнянні з попередніми архітектурами мовних моделей.

При генерації токенів, вектори попередніх токенів зберігаються в KV кеші. Це необхідно, оскільки без кешування ми б отримували складність обчислення $O(n^2)$. При кешуванні ж складність обчислення знижується до $O(n)$, що є значно ефективнішим. Однак, варто зазначити, що такий підхід вимагає значного обсягу пам'яті.

В останньому шарі (рис. 1.1, А) йде система із лінійним перетворенням, звичайною нейронною мережою без активації на вихідних нейронах та функція softmax, яка перетворює вихідні вектори у ймовірності по всім токенам зі словника, після чого вибирається найімовірніший токен.

Функція softmax визначається так:

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \ .$$

1.3 Методи генерування токенів тексту

Є декілька методів вибіру токенів після отримання ймовірностей [2].

Greedy search.

При отриманні розподілення ймовірностей по всім токенам, вибирається найімовірніший. В такому випадку, модель є детерміноманою, тобто для того ж самого входу, отримаємо такий самий вихід, тобто однаковий токен

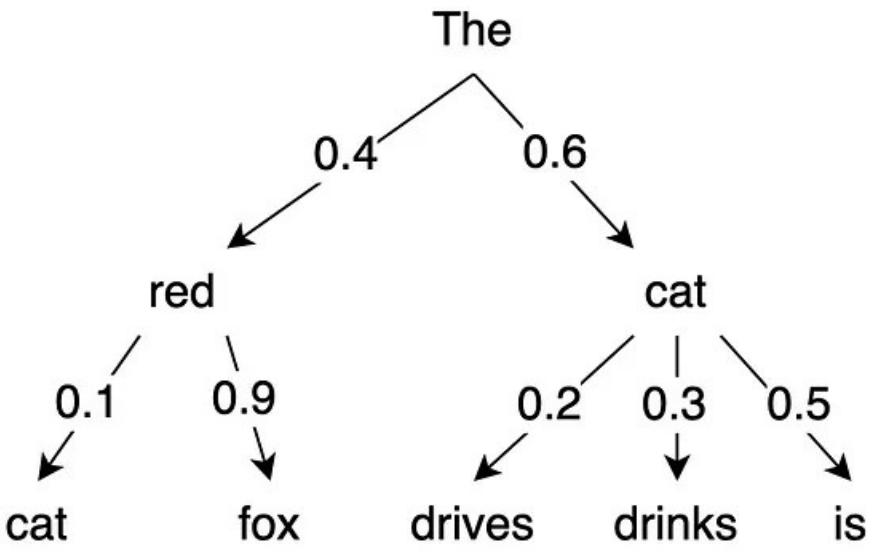

Рисунок 1.4 – Розподілення токенів для входу “The”

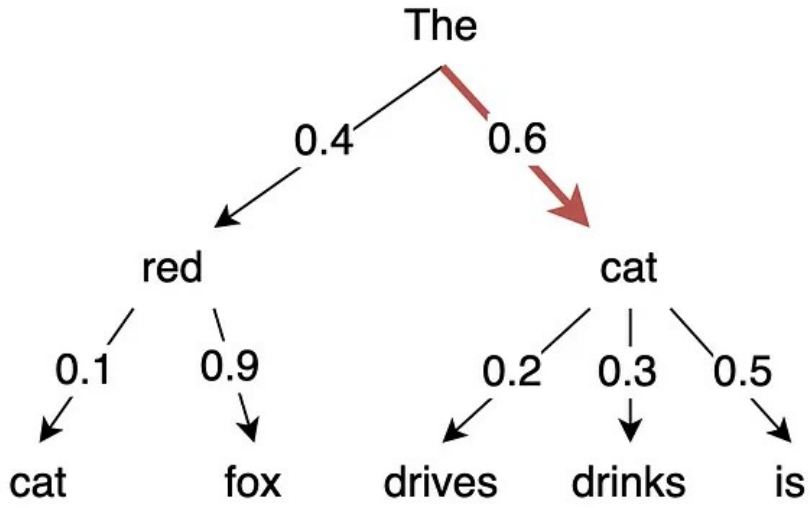

Рисунок 1.5 – Модель вибирає найімовірніший токен “cat”

Beam search.

В цьому методі, модель вибирає шлях з найбільшою імовірністю.

Параметр beam_num контролює кількість токенів моделів перевіряє, для прикладу, beam_num = 2.

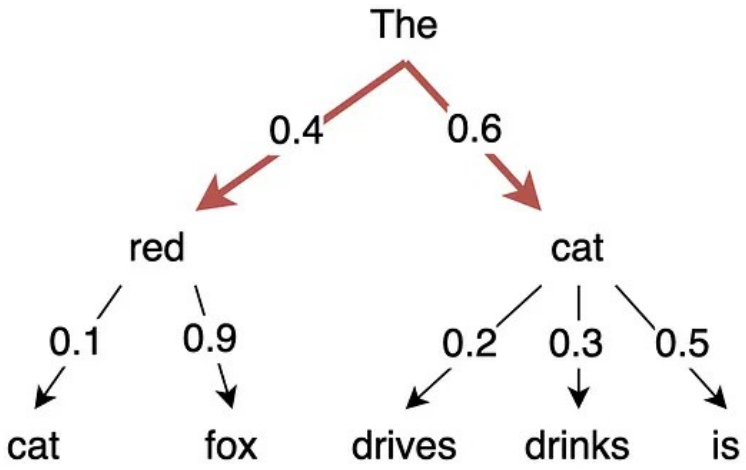

Рисунок 1.6 – Перевірка імовірності двох токенів

Як можна побачити, модель перевіряє спочатку “red” та “cat” токени

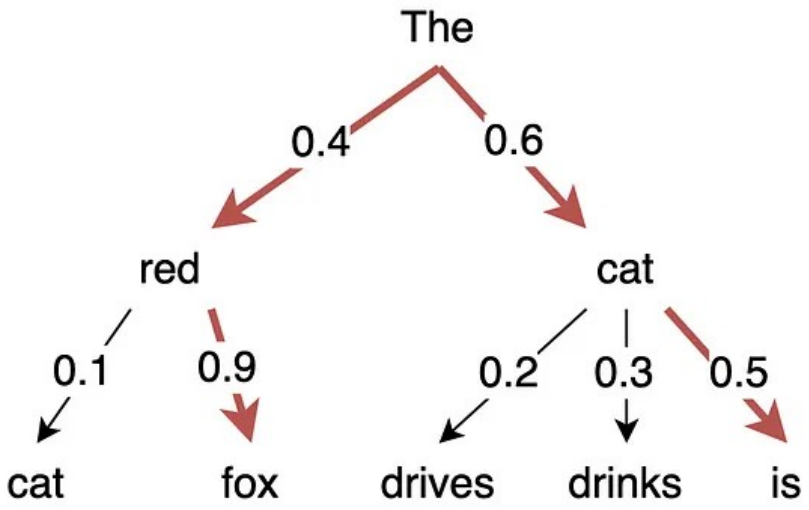

Рисунок 1.7 – Наступний крок beam search

На наступному кроці модель перевіряє токени “fox” та “is”. В даному прикладі, токени “the red fox” мають більшу імовірність, тому модель може зупинитись та повернути ці токени на виході.

Sampling.

Текст згенерований семплюванням(sampling) є більш природній ніж перші два методи, так як люди частіше очікують випадковість, але методи як beam search та greed search не є випадковими.

Базовий метод семплінг вибирає наступний токен з імовірністю, тобто токен з більшою імовірністю вибирається частіше, але у порівнянні з greed search інші токени з малою імовірністю також можуть бути обрані.

Для додаткової керованості генерування є декілька параметрів семплінгу: top-p, top-k, temperature.

Top-k: Цей параметр обмежує кількість токенів, з яких відбувається вибір.

Top-p: Цей параметр обмежує токени для вибору до тих, кумулятивна ймовірність яких не перевищує значення p.

Temperature: Цей параметр контролює розподіл ймовірностей між токенами. При значенні 0 семплінг не відрізняється від greedy search, тоді як при значеннях, близьких до 1, ймовірності токенів прагнуть до рівномірного

розподілу.

1.4 Проблеми великих мовних моделей та актуальність файнтюнінгу

Для отримання більш здатної моделі, потрібно мати більше параметрів, але при більшій кількості параметрів використовується більше ресурсів, повільна генерація тексту.

За проведеним тестуванням моделей з MMLU, були отримані результати в яких більші за розміром моделі отримали найвищу точність.

MMLU(Measuring Massive Multitask Language Understanding) - визначення масштабного багатозадачного розуміння мови, який оцінює здатність мовної моделі.

При цьому більшість з них є пропрієтарними і насправді кількість параметрів дізнатися не можна, але при екстраполяції даних про розміри попередніх великих моделей, а також дані тестування малих моделей, отримаємо результати на графіку (рис. 1.8).

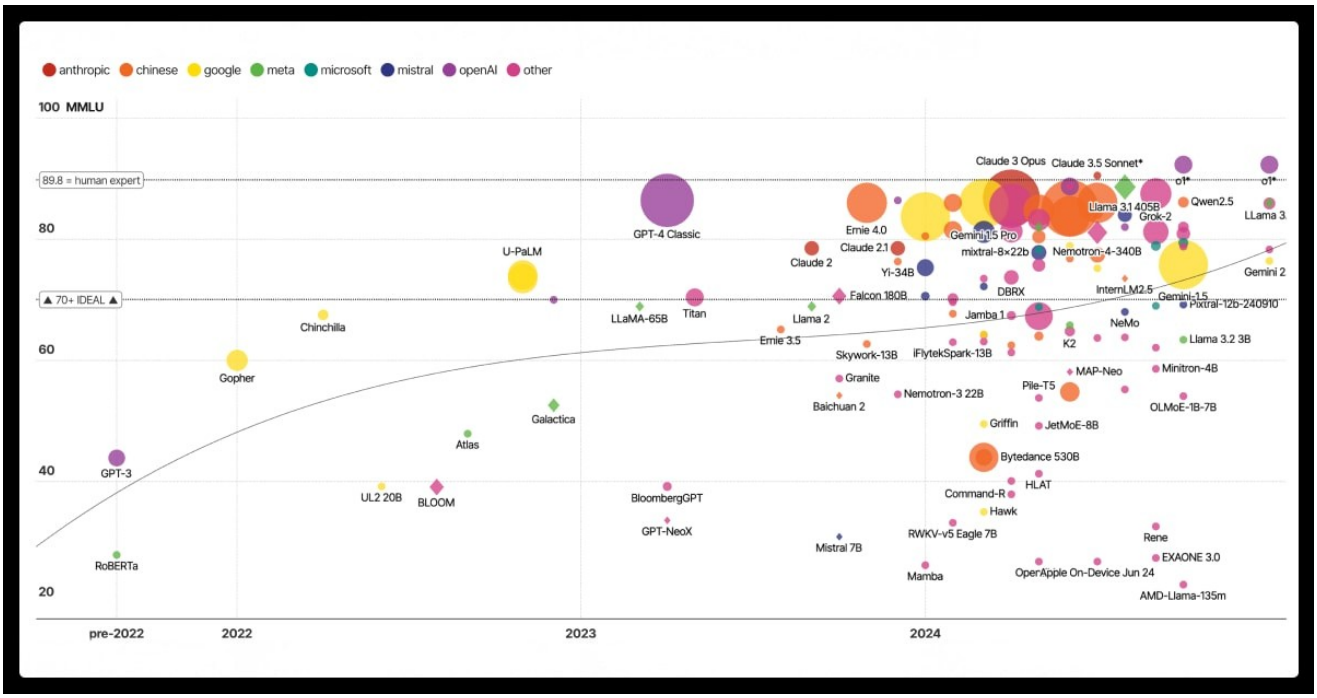

Рисунок 1.8 – Розміри моделей та їх точність у MMLU

Використовуючи відкриті мовні моделі, можна дізнатись про конфігурацію шарів, кількість їх параметрів та підрахувати скільки вони використовують пам’яті.

Наприклад, моделі Llama 65b або схожі за розміром, мають декілька основних параметрів.

Для Llama 65b, ці параметри мають такі значення:

d_model = 8192

d_k = 8192/64 = 128

num_heads = 64

кількість шарів – 80

Можна розрахувати скільки потрібно пам’яті для основних параметрів та збережених векторів (kv кеш) для Llama 65b.

Модель використовує точність вагів FP16, тобто 2 байти.

Отримаємо базовий розмір моделі не включаючи кеш та інші змінні: 65b * 2 = 65 000 000 000 * 2 байтів = 130 000 000 000 байтів = 126 953 125 кілобайтів = 123 977.~ мегабайтів.

Тобто модель llama 65b займає приблизно 124 gb пам’яті (1.1)

Таку модель не можна використовувати на домашніх пк, так як кількість пам’яті у більшості відеокарт менша ніж 30gb.

Є методи для зменшення використання пам’яті моделями, шляхом зменшення точності ваг, наприклад зменшення точності ваг до fp8 або навіть fp4 зменшує розмір моделі у 2, 4 рази відповідно, тобто модель буде мати розмір 62gb, 31gb відповідно. Але такі методи погано впливають на здатність моделі. Цей процес називається квантизація моделі або параметрів моделі.

Окрім квантизації, можливо запускати моделі не тільки на відеокарті, але й розподілити різні модулі, шари між процессором та відеокартою. При цьому, отримаємо можливість запускати більш великі моделі, але й швидкість генерації буде швидшою ніж просто на процессорі.

РОЗДІЛ 2 НАВЧАННЯ МОДЕЛІ ТРАНСФОРМЕР ТА МЕТОДИ ФАЙНТЮНІНГУ

2.1 Основні задачі мовних моделей

Для того щоб модель була корисною в задачах, її потрібно навчити на даних, які відображають суть задач які модель буде вирішувати. Є декілька видів задач для ВММ:

- генерація тексту – це найбільш загальна задача мовних моделей. При достатній кількості параметрів моделі та даних на яких навчається модель, така модель може замінити інші види моделей, але буде потребувати набагато більше ресурсів ніж задачо-орієнтовані мовні моделі.

- Модель для підведення підсумків – при отриманні тексту, генерує зменшений опис ключових деталей тексту.

- Переклад – перекладає текст з однієї мови на іншу, модель трансформер спершу з’явилася саме для цієї задачі.

- Запитанна-Відповідь – задачо орієнтована модель, яка відповідає на запитання які стосуються конкретного предмету.

- Класифікація тексту – генерує вектор або текст, який класифікує текст запиту та інші.

Сучасні мовні моделі зазвичай є загальними моделями які генерують текст та охоплюють всі інші задачі, чим більша модель, тим краще вона може їх вирішувати. Але для більших моделей потрібно більше ресурсів. Тому якщо потрібно щоб модель могла вирішити одну задачу та не має достатньо ресурсів для запуску великої моделі, потрібно використовувати методи файнтюнінгу маленьких моделей які справляться із задачею.

2.2 Навчання мовної моделі

На першому етапі навчання модель вивчає мову, послідовність токенів та інше.

Для навчання мовних моделей використовують багато текстових даних.

Так як багато даних береться з відкритих джерел, шляхом скрейпу та парсингу сайтів, в даних є елементи, які не є важливими, наприклад посилання, елементи форматування та інше, то перед використанням таких даних, вони очищуються та обробляються

Формально, навчання можна описати як оптимізацію функції

$$min_{\theta}(L(f_{\theta}(x), y)) \ ,$$

де L – функція втрати;

f_{θ} – мовна модель з параметрами θ ;

y – цільове значення яке відповідає входу x.

В мовних моделях, функцією втрати зазвичай використовують Cross-entropy loss.

Параметри моделі оновлюються за допомогою модифікованого методу градієнтного спуску – Adam:

$$W_t = W_{t-1} - \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \alpha$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \frac{\delta L}{\delta W_t}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left(\frac{\delta L}{\delta W_t} \right)^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1}$$

$$\hat{v}_t = \frac{m_t}{1-\beta_2}$$

де: W_t - параметри моделі на кроку t;

β_1, β_2 - параметри швидкості розпаду;

L - функція втрат.

Можна описати кроки навчання моделі.

1. Збір текстових даних та їх обробка.
2. Розбиття текстових даних на токени.
3. Формування пар токенів, $((x_1), x_2)$, $((x_1, x_2), x_3)$, $((x_1, x_2, .., x_{n-1}), x_n)$ де перший елемент пари є токени на вході, а другий елемент – цільове значення.
4. Вираховування помилка моделі для кожної пари та за допомогою методу зворотного поширення помилки разом з оптимізатором, оновлення параметрів моделі.

2.3 Файнтюнінг мовної моделі

Файнтюнінг або задачо орієнтоване навчання – це другий етап навчання, в якому модель навчається виконувати якусь задачу, наприклад відповідати на питання про компанію або можливо виконувати функції в програмному середовищі.

Є різні методи файн-тюнінгу, використання одного залежить від типу задачі, розміру моделі, а також розміру та якості даних.

Full Fine-Tuning.

Повний файн-тюнінг моделі, тобто модель навчають на підмножені даних задачі, але при цьому навчають всі параметри моделі, що може призвести до більшої кількості використаних ресурсів. Цей метод може змінити модель, зробити її більш нестабільною, а також вона може “забути” попередньо вивчену інформацію, це називається – катастрофічне забування(catastrophic forgetting).

Prompt Tuning.

Розрізняють Soft prompts та Hard prompts.

Hard prompt, це метод файн-тюнінгу, в якому задачу яку потрібно зробити описують текстом та передають разом з текстом який потрібно обробити.

Soft prompt[8], це схожий метод, але замість тексту який описую задачу, використовують декілька векторів на вході у моделі, які навчаються як потрібно описувати задачу, які напряму передаються в основні шари уваги разом з додатковим текстом.

Якщо $X=[t_1,t_2,...,t_n]$ вхід моделі, який складається з кодованих токенів у векторах, то soft prompt метод додає додаткові вектори $X=[t_1,t_2,...,t_n,h_1,h_2,...,h_k]$, де h_i - вектор soft prompt.

Для задачі, ці вектори навчаються за допомогою back propagation.

Цей підхід дозволяє великим моделям отримати кращі результати, а ще й кількість ресурсів потрібних для навчання буде малою, але при цьому не має можливості “зрозуміти” або описати текстом вектори які використовуються, тобто це black box (темний ящик).

Цей підхід працює найкраще для великих моделей.

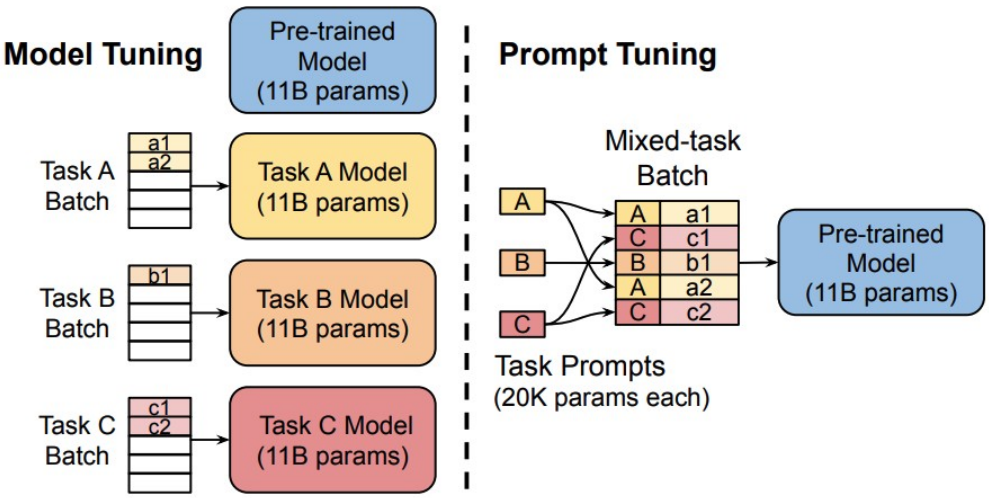

Рисунок 2.1 – Soft tuning

Prefix tuning.

Цей метод схожий на soft prompt, але замість додавання векторів на початку, в цьому методі вектори додаються на кожному шарі моделі. В моделі з'являється більше можливості "зрозуміти" інформацію з даних.

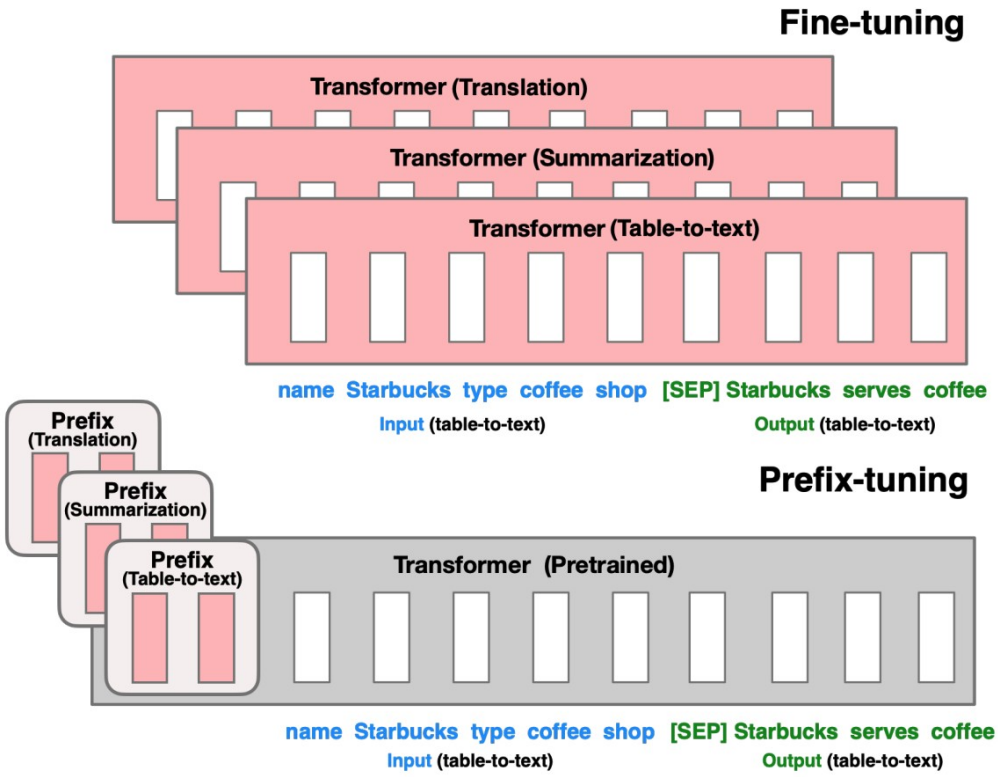

Рисунок 2.2 – Зrefix tuning

LoRA (Low-Rank Adaptation).

Метод, який з’явився завдяки розуміння того, що у навчених мовних моделях, оновлення, що стосуються конкретного завдання, лежать у низькому внутрішньому розмірі простору параметрів моделі(low intrinsic dimension).

Цей метод використовує декомпозицію матриці для отримання меншої кількості параметрів для навчання.

Якщо ваги моделі представити однією матрицею W_0, то lora буде оновляти ваги матриці ΔW, ранг якої менше ніж W_0.

Матриця ΔW можна представити як добуток двох матриць $A_{r,\text{size}}$, $B_{\text{size},r}$

$$\Delta W = B_{\text{size},r} * A_{r,\text{size}},$$

де size – ранг матриці ваг моделі;

r – ранг матриці ΔW та $r \ll \text{size}$.

Тоді модель з оновленими параметрами матиме вигляд: $W = W_0 + \frac{r}{a}\Delta W$.

При цьому, ваги W_0 моделі не оновлюються, тільки ΔW.

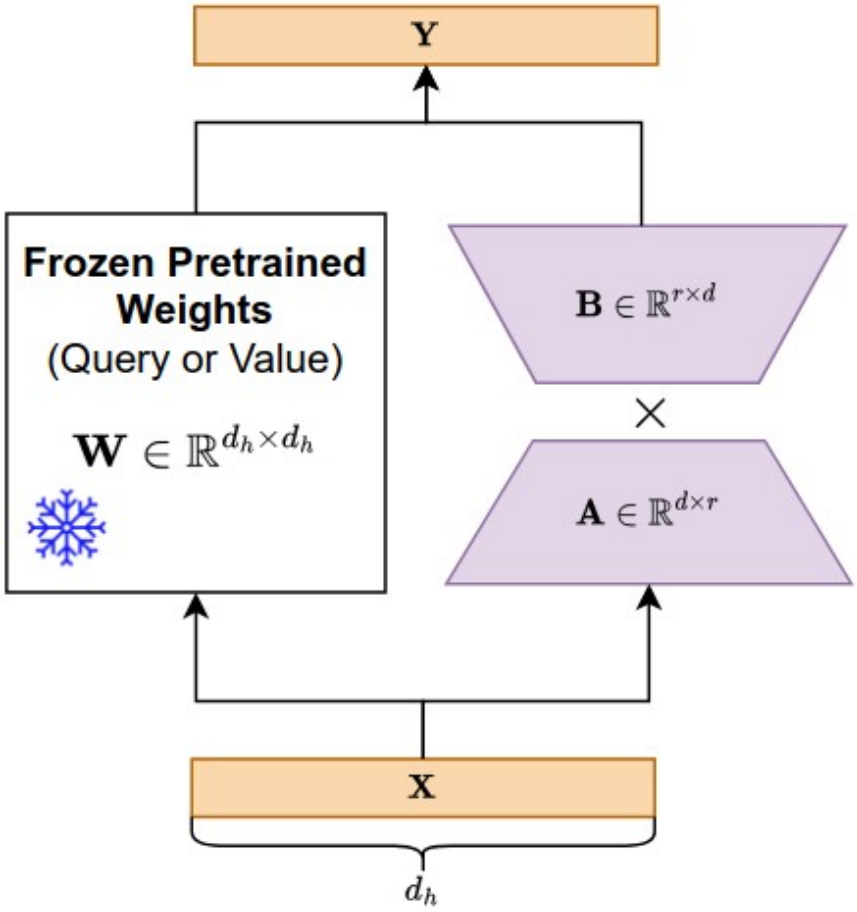

Рисунок 2.3 – Lora

Кількість параметрів для оновлення буде r * size * 2 << size * size.

Наприклад для моделі Llama 65b та рангом r = 4, кількість параметрів для оновлення буде $\text{size} = \sqrt{(65000000000)}$, $\text{size} * r * 2 \approx 254951 * 4 * 2 \approx 2039608$, це 0.0031% параметрів від оригінальної кількості.

Тобто файнтюнінг моделі за допомогою LoRA потребує набагато менше ресурсів, але й здатність моделі при цьому не зменшується.

QLoRA (Quantized LoRA) [3].

Метод Lora який використовує додаткові 3 підметоди, які роблять qlora набагато ефективнішим ніж звичайний lora.

1. Краща квантизація параметрів моделі до 4біт (NF4).
2. Подвійна квантизація квантизаційних констант.
3. Збереження стану оптимізатора у пам’яті при великій кількості токенів.

KronA [4].

Метод схожий до Lora який також використовує матрицю з меншим рангом ніж матриця основних ваг моделі. Але замість декомпозиції, цей метод використовує добуток кронекера.

Оновлюються параметри матриць A, B.

$$W = W_0 + A \otimes B$$

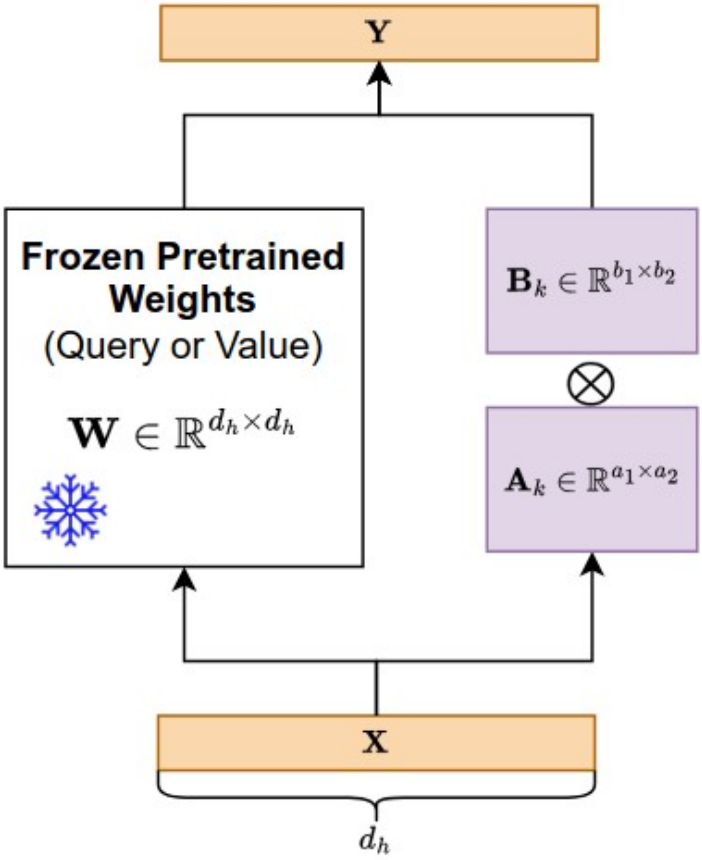

Рисунок 2.4 – метод KronA

РОЗДІЛ 3 АНАЛІЗ МЕТОДІВ ФАЙНТЮНІНГУ

3.1 Постановка задачі

Мета роботи полягала в порівнянні якості відповідей ВММ для різних методів файнтюнінгу в залежності від обраних гіперпараметрів.

Для задачі, була обрана мовна модель SmolLM2-360m-Instruct з 360 мільонами параметрів. Ця модель розповсюджується з ліцензією Apache License 2.0, тобто для її використання не потрібні додаткові угоди, її вільно можна використовувати для власних та комерційних проектів.

Для перевірки методів файнтюнінгу, була обрана задача - допомога в написанні модів та ігор з використанням ігрового рушію Luanti [7].

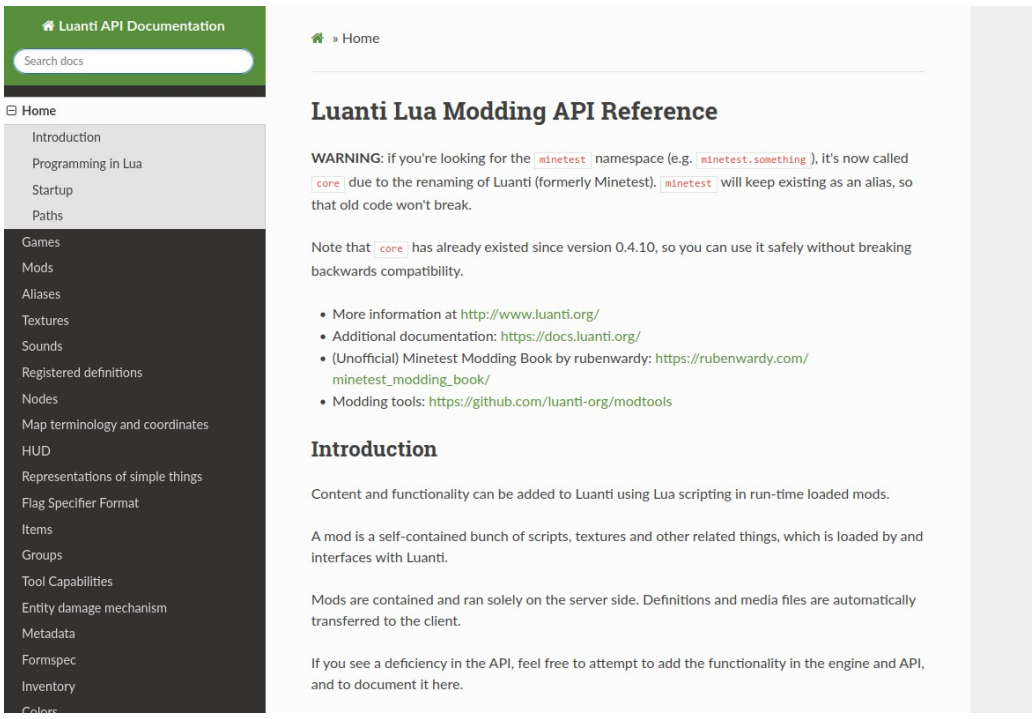

Рисунок 3.1 – довідник API Luanti

Luanti рушій використовує скриптову мову lua, а також має детальний довідник.

Файнтюнинг моделі буде з такими методами: QLoRA, Prompt

tuning(Hard prompts)

3.2 Підготовка середовища

Вся робота пробовидалась у середовищі Python 3.13.

Для роботи з мовними моделями були встановлені такі пакети:

- **peft** – пакет для роботи з методами файнтюнінгу моделей;
- **trl** – пакет для навчання моделей;
- **datasets** – пакет для роботи з даними;
- **transformers** – пакет для роботи з мовними моделями;
- **bitsandbytes** – пакет для квантизації моделей;
- **torch –** пакет для роботи з методами машинного навчання, потрібен для всіх попередніх пакетів.

Перед втстановленням пакетів, було створено віртуальне середовище для python використовуючи модуль venv.

python3 -m venv .env.

Виконавши цю команди в терміналі, в каталозі проекту, буде створене середовище у папці .env.

Після цього, можна активувати це середовище.

source ./.env/bin/activate

Якщо роботу було завершено у середовищі, командою deactivate можна деактивувати середовище.

3.3 Обробка та підготовка даних

Для навчання, дані були зібрані з декількох ресурсів про luanti, luanti api, довідник про luanti та інші [5].

Після отримання текстових файлів, вони були оброблені та підготовлені для навчання використовуючи python скрипт який відкриває текстові файли, зчитує строки та за необхідністю очищає їх від непотрібних елементів. (Додаток А process_data_luanti.py).

Для обробки даних, було написано декілька допоміжних функцій, включаючи основний скрипт обробки.

Один з таких методів виглядає так:

```
def clean_data(st):
  text = re.sub(r'\[.*?\]\((.*?)\)', r'\1', st)
  text = re.sub(r'<!--[a-z A-Z 0-9]*-->', "", text)
  text = re.sub(r'\={3,}', "", text)
  text = re.sub(r'\:{1,}', "", text)
  text = re.sub(r'\\{2,}', "", text)
  text = re.sub(r'\|{1,}', "", text)
  text = re.sub(r'\-{3,}', "", text)
  text = re.sub(r'\#{1,}', "", text)
  text = re.sub(r'\*{1,}', "", text)
  text = re.sub(r'\n{2,}', "\n", text)
  text = re.sub(r'\s{2,}', " ", text)

  return text.strip()
```

Тобто використовуючи regex, цей метод шукає в тексті непотрібні елементи, символи форматування та інше, замінюючи їх на пусті символи або пробіли.

Наприклад елемент [Link here] (https://example.com) буде оброблений та перетворений у такий текст https://example.com, тобто просто посилання без форматування.

3.4 Файнтюнінг моделі

Для кожного файнтюнінгу були обрані декілька параметрів.

QLorA

r=2,4

alpha = 2,4

Метод prompt tuning(Hard prompt) не потребує параметрів

При цьому, для кожного методу загальні параметри швидкості навчання та кількість епох були однакові.

Швидкість навчання – 5e-4.

Кількість епох – 3.

Цільові шари для яких будуть створені матриці параметрів – всі лінійні шари, тобто всі матриці W_o, W_k, W_v, W_q .

Перед навчанням, дані були завантажені, оброблені у вид який приймає метод та розподілені на підмножини даних - тестові та навчальні.

Під час навчання LoRA, функція навчання повертає статистику кожні k кроків. В статистиці є:

- **loss** – значення функції втрат на даному кроці;
- **grad_norm** – норма градієнту останнього оновлення;
- **learning_rate** – швидкість навчання на даному кроці;
- **num_tokens** – кількість токенів на яких модель вже навчилась;
- **mean_token_accuracy** – відсоток корректних генерацій токенів;
- **epoch** – епоха на даному кроці.

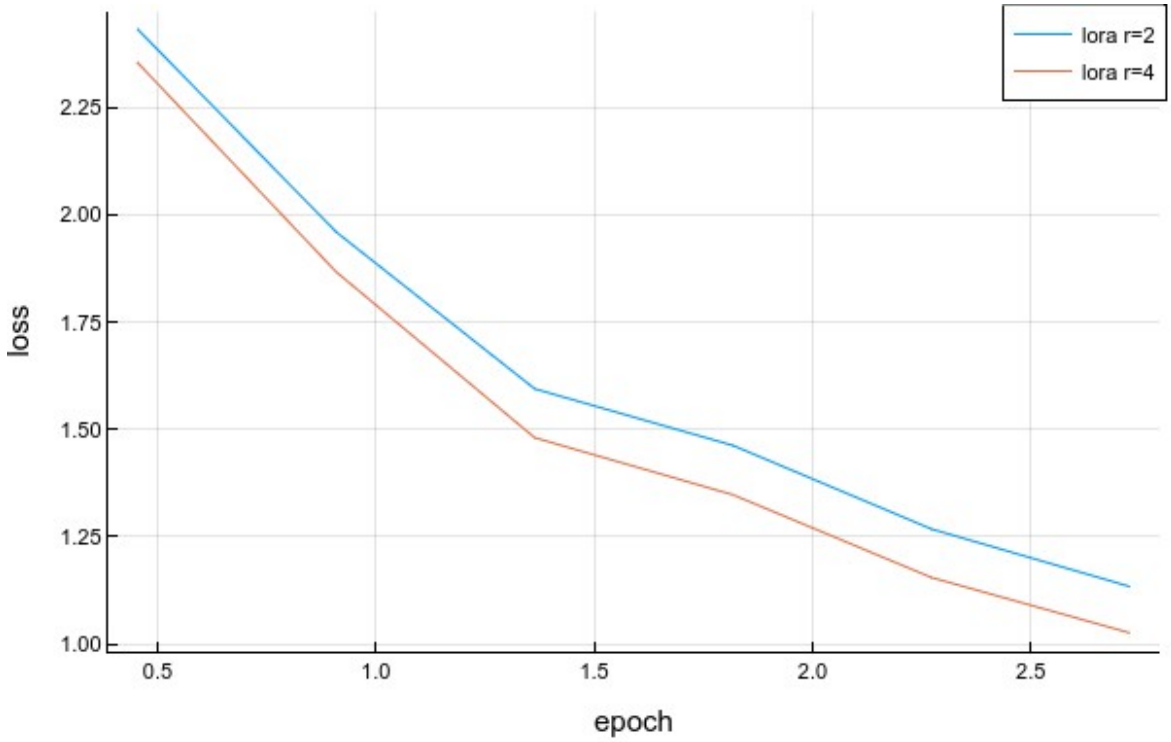

Рисунок 3.2 – значення функції втрати при навчанні методом LoRA

LoRA з рангом 4 має менше початкове значення функції втрат, скоріше за все через те, що маючи більше параметрів – модель краще навчається.

3.5 Тестування та висновки

Для тестування та аналізу методів, була використана метрика pass@k .

Ця метрика визначається формулою: $E[1-\frac{C_k^{(n-c)}}{C_k^n}]$,

де: n – кількість генерацій на один запит;

c – кількість вірних відповідей на один запит моделлю;

k – кількість обраних відповідей.

Ця метрика допомагає переконатись що з k відповідей на питання, хоча б один буде вірний з імовірністю pass@k. Так як мовні моделі не детерміністичні, то отримання більше коректних відповідей на один і той самий запит, робить модель більш корисною для роботи.

Вірні відповіді були вибрані вручну, тобто кожну відповідь було перевірено на корректну інформацію за довідником ігрового рушію Luanti. Довжина та детальність відповіді при тестуванні не враховувались.

Була створена множина запитів, всього 23, які поділяються на 3 рівні складності. Всі запити відносились тільки до ігрового рушію Luanti.

Приклад трьох рівнів складності:

1. "What is luanti?", "What language does luanti use for making games?"
2. "What files are inside game folder?", "What is world-specific game?"
3. "What methods are deprecated in core namespace?", "What arguments does core.get_connected_players function accept?"

Для тестування був вибран метод генерації – семплінг, для якого були обрані такі значення:

temperature = 0.4, 0.6

top_k = 20

top_p = 0.95

Для метрики pass@k були обрані такі параметри:

n = 3

k = 2

Тобто при генерації двох запитів, ми отримаємо ймовірність що хоча б один вірний

Також для автоматизації перевірки запитів та відповідей моделі була створена програма з простим інтерфейсом з можливістю вибору кількості правильних відповідей та підрахунку результатів(Додаток А chat_testing.py)

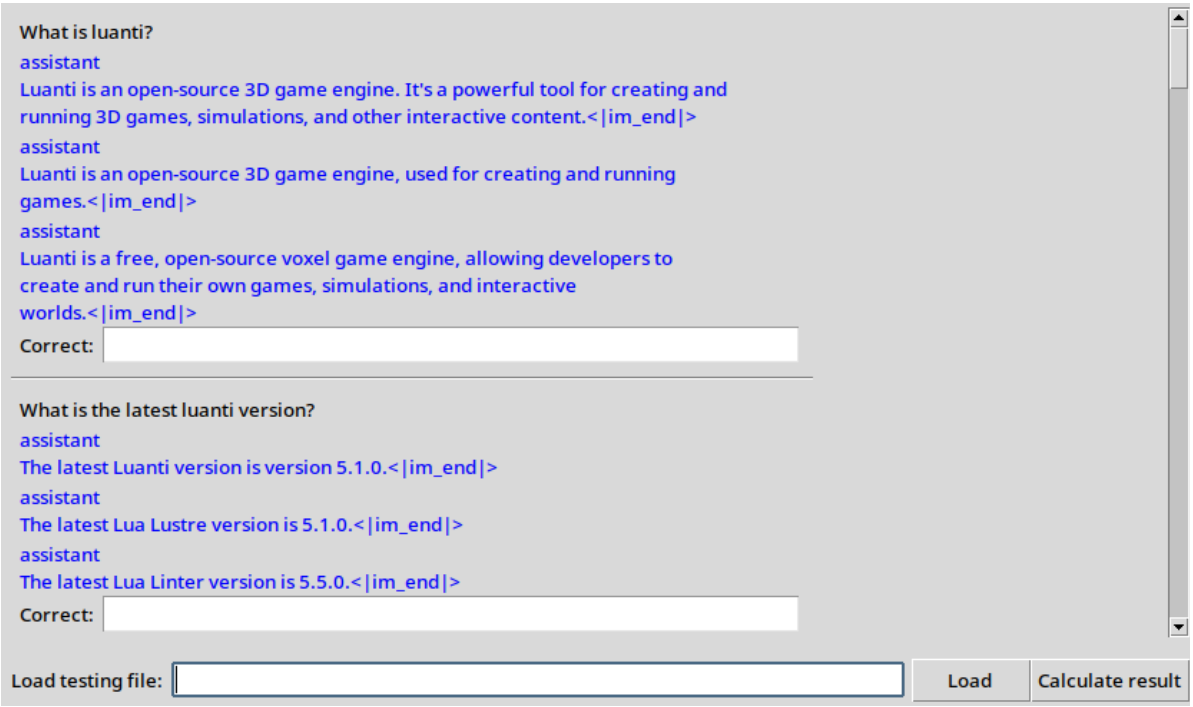

Рисунок 3.3 – Інтерфейс програми

Програма була написана за допомогою python та пакету tkinter.

Після тестування та отримання результатів, були зроблені графіки.

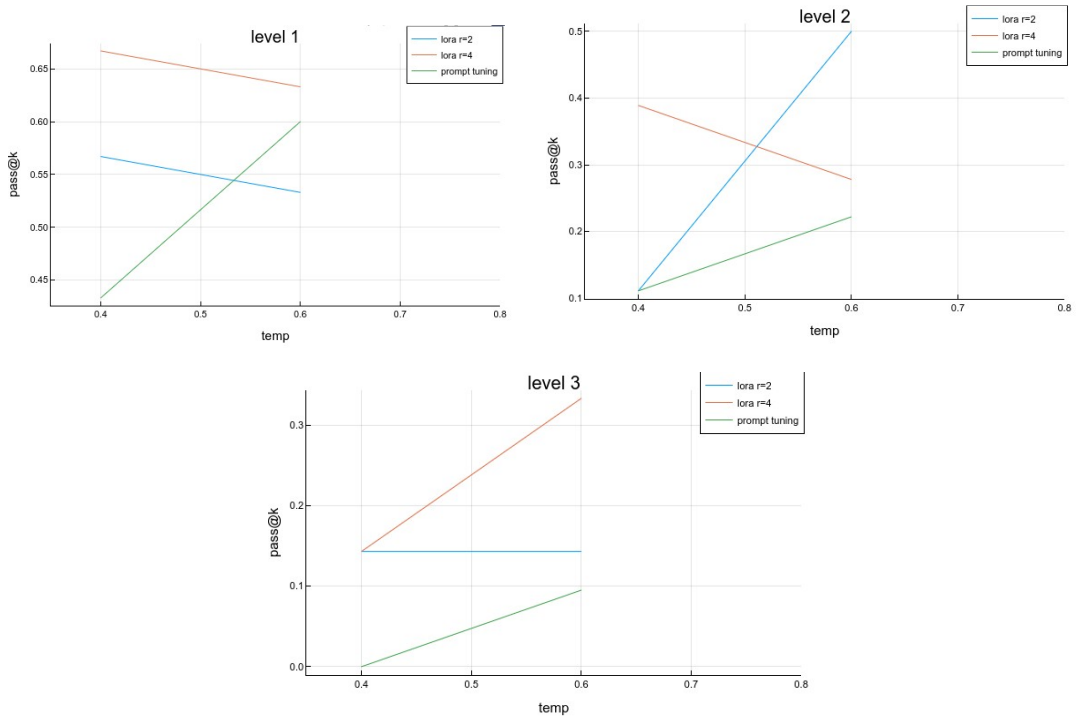

Рис. 3.4 – результати тестування

Тестування показало що файнтюнінг моделі за допомогою LoRa з рангом 4 дає найкращі результати, але, окрім файнтюнінгу, на точність відповідей впливає також метод генерації та значення параметрів.

Для prompt tuning, збільшення значення параметру temperature призвело до збільшення точності відповідей, в той час як для lora з рангом 4 для перших двох рівнів складності це призвело до зменшення точності, для останнього рівня – навпаки, до збільшення точності.

При цьому, метод lora потребує в 150 разів більше інформації (300 тисяч токенів тексту) для навчання, тоді як для prompt tuning потрібно усього приблизно 2000 токенів.

ВИСНОВКИ

Файнтюнінг мовних моделей дозволяє створити інтелектуальний інструмент, здатний значно підвищити ефективність виконання завдань. При цьому процес адаптації моделі не потребує значних обчислювальних ресурсів — як на етапі навчання, так і під час подальшого використання.

На основі отриманих результатів можна зробити висновок, що для досягнення максимальної точності доцільно використовувати метод LoRA. Водночас, у випадках обмеженої кількості даних або ресурсів, Prompt Tuning(Hard prompt) є цілком життєздатною альтернативою.

Незалежно від обраного підходу, для досягнення найкращих результатів необхідне додаткове тестування та тонке налаштування параметрів генерації. Це дозволяє врахувати особливості конкретного завдання й забезпечити найвищу якість відповідей мовної моделі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Attention Is All You Need / A. Vaswani [et al.] // *arXiv.org* [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/1706.03762 (дата звернення: 01.06.2025).
2. Generation strategies. *Hugging Face – The AI community building the future* [Електронний ресурс]. — Режим доступу: https://huggingface.co/docs/transformers/generation_strategies (дата звернення: 01.06.2025).
3. LoRA: Low-Rank Adaptation of Large Language Models /E. J. Hu [et al.] // *arXiv.org* [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/2106.09685 (дата звернення: 01.06.2025).
4. KronA: Parameter Efficient Tuning with Kronecker Adapter / A. Edalati [et al.] // *arXiv.org* [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/2212.10650 (дата звернення: 01.06.2025).
5. Luanti API Documentation. *Luanti API Documentation* [Електронний ресурс]. — Режим доступу: https://api.luanti.org/ (дата звернення: 01.06.2025).
6. PEFT: Parameter-Efficient Fine-Tuning Methods for LLMs. *Hugging Face – The AI community building the future* [Електронний ресурс]. — Режим доступу: https://huggingface.co/blog/samuellimabraz/peft-methods (дата звернення: 01.06.2025).
7. Luanti | Open source voxel game engine. *Luanti | Open source voxel game engine* [Електронний ресурс]. — Режим доступу: https://www.luanti.org/ (дата звернення: 01.06.2025).
8. Lester B. The Power of Scale for Parameter-Efficient Prompt Tuning / B. Lester, R. Al-Rfou, N. Constant // *arXiv.org* [Електронний ресурс]. — Режим доступу: https://arxiv.org/abs/2104.08691 (дата звернення: 01.06.2025).
9. Учасники проектів Вікімедіа. Велика мовна модель – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Велика_мовна_модель (дата звернення: 01.06.2025).

ДОДАТОК А
Код програми

process_data_luanti.py

```
import json
import datasets
import random
import io
import re
import os
import utils.clean as utils

chunks = []

def process_chunks(file, chunks):
  with open(file, "r") as f:
    j = 0
    st = []
    is_code = False
    for i in f.readlines():
      if len(i) != 0 and i != "\n":

        if re.search(r"`{2,}lua", i):
          is_code = True
        elif is_code and re.search(r"`{2,}", i):
          is_code = False

        if j >= 500:
          text = " ".join(st)
```

```
        chunks.append({"text": text})
        j = 0
        st = []
      else:
        if not is_code:
          st.append(utils.clean_data(i))
        j = j + 1
    if len(st) > 0:
      chunks.append({"text": " ".join(st)})

text_corpus_files = "./llms/datasets/raw/luanti/text_corpus"
include_file_types = ["md", "txt", "conf"]

def process_files_in_directory(directory):
  chunk_files = os.listdir(directory)
  for i in chunk_files:
    file = directory + "/" + i
    print(file)
    if os.path.isfile(file):
      ext = i.rfind(".")
      if ext != -1:
        if i[(ext+1):] in include_file_types:
          process_chunks(file, chunks)
    elif os.path.isdir(file):
      process_files_in_directory(file)

process_files_in_directory(text_corpus_files)

dataset_chunks = datasets.Dataset.from_list(chunks)
```

```
json.dump(chunks, open("./llms/datasets/chunks.json", "w"))
dataset_chunks.save_to_disk("./llms/datasets/data-luanti-chunks.hf")
```

training.py

```
from transformers import AutoTokenizer, AutoModelForCausalLM, Trainer, TrainingArguments, DataCollatorWithPadding, BitsAndBytesConfig
import transformers
from peft import get_peft_model, LoraConfig, TaskType, IA3Config, IA3Model, LoKrConfig
import datasets
from datasets import load_dataset, concatenate_datasets
import json
from trl import apply_chat_template, SFTConfig, SFTTrainer
import inspect


checkpoint = "HuggingFaceTB/SmolLM2-360M-Instruct"

device = "cuda"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
quantization_config = BitsAndBytesConfig(load_in_4bit=True, bnb_4bit_compute_dtype="float16", bnb_4bit_use_double_quant=True, bnb_4bit_quant_type="nf4", )
model = AutoModelForCausalLM.from_pretrained(checkpoint, quantization_config=quantization_config)

lora_config = LoraConfig(
```

```
    r=2,
    target_modules="all-linear",
    task_type=TaskType.CAUSAL_LM,
    lora_alpha=2,
    lora_dropout=0.3,
)

lora_model = get_peft_model(model, lora_config)
lora_model.print_trainable_parameters()


training_args = SFTConfig(
    packing=False,
    output_dir="PATH_TO_SAVE_FINETUNED_MODEL", #TODO
    per_device_train_batch_size=1,
    gradient_accumulation_steps=1,
    num_train_epochs=3,
    learning_rate=5.0e-4,
    pad_token="<|im_end|>"
)


### Luanti dataset
dataset = datasets.Dataset.load_from_disk("PATH_TO_DATASET").train_test_split(train_size=0.8) #TODO
###

trainer = SFTTrainer(
```

```
    model=lora_model,
    args=training_args,
    train_dataset=dataset,
)

trainer.train()
```

testing.py

```
from transformers import BlipProcessor, BlipForConditionalGeneration, AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig
import datasets
import json
import time
from utils import clean

checkpoint = "HuggingFaceTB/SmolLM2-360M-Instruct"
device = "cuda"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model = AutoModelForCausalLM.from_pretrained(checkpoint, device_map="cuda")

test_prompts = {
    "1": [
    "What is luanti?",
    "What is the latest luanti version?",
    "when was latest luanti released?",
    "Is minetest same as luanti?",
```

```
    "What is minetest?",
    "Is luanti open-source?",
    "Is luanti voxel engine?",
    "What language does luanti use for making games?",
    "Is luanti a minecraft clone?",
    "Was Luanti created by Mojang?",
    ],

    "2": [
    "What files are inside game folder?",
    "What files are inside mod folder?",
    "In which file can I define mod name?",
    "What are voxels in Luanti?",
    "What is mod.conf file inside mod directory?",
    "What is world-specific game?",
    ],

    "3": [
    "What methods are deprecated in core namespace?",
    "What is the size of a world in luanti?",
    "What can I do with core.register_tool function?",
    "What is core.register_on_placenode function?",
    "What is the definition for particle emitter in luanti?",
    "Show me basic code to register tool",
    "What arguments does core.get_connected_players function accept?"
    ]
}

file_name = "PATH_TO_SAVE_MODEL_REPLIES"
```

```
data = []

try:
    with open(file_name, "r") as f:
        data = json.load(f)
except Exception as ex:
    print("Doesnt exist!")

system_prompt = ""

prompt_tuning_file = "PATH_TO_HARD_PROMPT_FILE_DATA"

with open(prompt_tuning_file, "r") as f:
    lines = f.readlines()
    for i in lines:
        system_prompt = system_prompt + clean.clean_data(i)

print(len(system_prompt.split(" ")))

def generate_llm_text(text):

    ctx = []
      ctx.append({"role": "system", "content": f"You are helpful AI assistant which helps with Luanti development!"})# {system_prompt}"})
    ctx.append({"role": "user", "content": text})

                input_text=tokenizer.apply_chat_template(ctx, tokenize=True, return_dict=True, return_tensors="pt").to(device)
```

```
    outputs = model.generate(
        **input_text,
        max_new_tokens=50,
        temperature=0.6,
        top_p=0.95,
        top_k=20,
        do_sample=True,
        pad_token_id=tokenizer.pad_token_id
    )
    output = tokenizer.decode(outputs[0])
    output_text = output

    outputs = output_text.rsplit("<|im_start|>")

    return outputs[-1]

k = 3
for level in test_prompts:
    for j in test_prompts[level]:
        l = {"level": level, "prompt": j, "replies": []}
        for i in range(k):
            l['replies'].append(generate_llm_text(j))
        data.append(l)

json.dump(data, open(file_name, "w"), indent=4)
```

chat_testing.py

```
import tkinter as tk
from tkinter import ttk
import json
import time
from utils import clean
import math

class ChatAI_UI:
    def __init__(self, master):
        self.master = master
        master.title("Chat testing")

        self.messages_data = []

        # --- Chat Display Area ---
        self.chat_frame = ttk.Frame(master)
        self.chat_frame.pack(pady=10, padx=10, fill="both", expand=True)

        self.chat_canvas = tk.Canvas(self.chat_frame)
        self.chat_canvas.pack(side="left", fill="both", expand=True)

            self.scrollbar = ttk.Scrollbar(self.chat_frame, orient="vertical",
command=self.chat_canvas.yview)
        self.scrollbar.pack(side="right", fill="y")

        self.scrollable_frame = ttk.Frame(self.chat_canvas)
        self.scrollable_frame.bind(
            "<Configure>",
```

```
            lambda e: self.chat_canvas.configure(
                scrollregion=self.chat_canvas.bbox("all")
            )
        )

        self.chat_canvas.create_window((0, 0), window=self.scrollable_frame, anchor="nw")
        self.chat_canvas.configure(yscrollcommand=self.scrollbar.set)

        # --- Input Area ---
        self.input_frame = ttk.Frame(master)
        self.input_frame.pack(pady=5, padx=10, fill="x")

        self.prompt_label = ttk.Label(self.input_frame, text="Load testing file:")
        self.prompt_label.pack(side="left", padx=(0, 5))

        self.prompt_entry = ttk.Entry(self.input_frame, width=60)
        self.prompt_entry.pack(side="left", expand=True, fill="x", padx=(0, 5))
        self.prompt_entry.bind("<Return>", self.load_testing_data) # Send on Enter key

        self.load_button = ttk.Button(self.input_frame, text="Load", command=self.load_testing_data)
        self.load_button.pack(side="left")

        self.export_button = ttk.Button(self.input_frame, text="Calculate result", command=self.export_message)
        self.export_button.pack(side="left")
```

```
        def add_message_to_display(self, prompt, messages, message_index):
            message_pair_frame = ttk.Frame(self.scrollable_frame, padding=5)
            message_pair_frame.pack(fill="x", expand=True)

                prompt_label = ttk.Label(message_pair_frame, text=prompt,
wraplength=500, justify="left")
            prompt_label.pack(anchor="w")


            for i in range(len(messages)):
                    reply_label = ttk.Label(message_pair_frame, text=messages[i],
wraplength=500, justify="left", foreground="blue")
                reply_label.pack(anchor="w")

            options_frame = ttk.Frame(message_pair_frame)
            options_frame.pack(anchor="w", expand=True)

            option1_label = ttk.Label(options_frame, text="Correct:")
            option1_label.pack(side="left", padx=(0, 5))
            option1 = ttk.Entry(options_frame, width=60)
            option1.pack(side="left", padx=(0, 5))
                option1.bind("<KeyPress>", lambda event, idx=message_index:
self.update_message("correct", option1, idx))

              ttk.Separator(self.scrollable_frame, orient="horizontal").pack(fill="x",
pady=5)

            # Auto-scroll to the bottom
            self.chat_canvas.update_idletasks() # Update layout
```

```
            self.chat_canvas.yview_moveto(1.0) # Scroll to the bottom

        def export_message(self, event=None):
            data = {}

            for i in self.messages_data:
                level = i['level']
                if not (level in data):
                    data[level] = []


                correct = int(i['correct'].get())
                                    data[level].append(1 - math.comb((i['n']-correct),
2)/math.comb((i['n']), 2))

            for k in data.keys():
                print(str(k))
                print(str(sum(data[k])/len(data[k])))

        def load_testing_data(self, event=None): # event is passed when bound to
<Return>
            prompt = self.prompt_entry.get()
            if not prompt:
                return

            data = {}
            with open("./llms/" + prompt, "r") as f:
                data = json.load(f)
```

```
            i = 0
            for k in data:
                self.messages_data.append({'level': k['level'], 'correct': None, 'n':
len(k['replies'])})
                self.add_message_to_display(k['prompt'], k['replies'], i)
                i = i + 1

            # Clear input field
            self.prompt_entry.delete(0, tk.END)

        def update_message(self, t, message, message_index):
            self.messages_data[message_index][t] = message
```