МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія (кафедра) комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

БЕНЧМАРКІНГ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

Виконав: студент групи 2П-21
Спеціальності
121 Інженерія програмного забезпечення
Дмитро ШИПОВАЛОВ
Керівник:
Станіслав МАРЧЕНКО

Черкаси 2025

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

Кафедра комп'ютерної інженерії та інформаційних технологій

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри КІ та ІТ

_____ Владислав ХОТУНОВ
(підпис)
«_____»_____2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Шиповалову Дмитру Вадимовичу

1. Тема кваліфікаційної роботи Бенчмаркінг великих мовних моделей
Керівник роботи Марченко Станіслав Віталійович, спеціаліст І категорії

затверджені наказом закладу вищої освіти від «07» жовтня 2024 року № 68у.
2. Строк подання студентом кваліфікаційної роботи 02.06.2025
3. Вихідні дані до кваліфікаційної роботи роботи мова програмування Python, інструменти для роботи з LLM requests, pandas, openai, фреймворк для тестування програмного рішення pytest, сервіс OpenRouter для API-інтерфейсу.
4. Зміст випускної роботи (перелік питань, які потрібно розробити) огляд та вибір LLM для бенчмаркінгу, методики оцінки продуктивності (швидкість генерації, точність), написання промптів для тестування, порівняння за стандартними метриками, розробка прототипів бенчмаркінгу (створення тестів для оцінки продуктивності, точності), тестування програмного рішення (вибір методів та інструментів тестування), побудова плану для тестування, аналіз результатів (сильні та слабкі сторони моделей, рекомендації для покращення).
5. Дата видачі завдання 16.09.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання з підписами керівника і студента
1	Вступ	14.10.2024	
2	Розділ 1. Огляд великих мовних моделей та їх методів оцінки	9.12.2024	
3	Розділ 2. Розробка та реалізація системи бенчмаркінгу	10.03.2025	
4	Розділ 3. Аналіз результатів бенчмаркінгу	28.04.2025	
5	Висновки	12.05.2025	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	26.05.2025	
7	Перевірка кваліфікаційної роботи на наявність ознак плагіату	02.06.2025	
8	Подання кваліфікаційної роботи на затвердження завідувачу кафедри	10.06.2025	

Студент _____ Дмитро ШИПОВАЛОВ
(підпис)

Керівник роботи _____ Станіслав МАРЧЕНКО
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота присвячена комплексному аналізу сучасних великих мовних моделей штучного інтелекту та розробленню автоматизованої системи бенчмаркінгу для оцінювання їх якості, продуктивності та функціональних можливостей. У роботі детально розглянуто архітектурні особливості провідних великих мовних моделей (LLM), методи їх навчання на масштабних текстових масивах, а також основні сфери застосування, від обробки природної мови до автоматизації програмування і математичних обчислень. Особлива увага приділена вибору LLM для бенчмаркінгу, описано процес інтеграції моделей через API, а також методи підключення та обробки даних для проведення якісного і комплексного тестування.

Розроблено власну систему бенчмаркінгу, яка забезпечує автоматизоване та стандартизоване тестування мовних моделей за різними категоріями завдань, включно із загальними знаннями, логічним мисленням, програмуванням, математичними розрахунками та іншими спеціалізованими напрямками. Система інтегрує сучасні метрики оцінювання якості, такі як точність, повнота, швидкодія відповіді, а також вимірює здатність моделей до генерації контекстно коректної та логічно послідовної інформації.

Проведене комплексне тестування дозволило не лише порівняти продуктивність і точність обраних моделей, а й виявити їх сильні та слабкі сторони, особливості поведінки у специфічних типах завдань та ефективність використання у реальних прикладних сценаріях. Результати підтверджують високу конкурентоспроможність досліджуваних моделей та демонструють практичну користь розробленої системи бенчмаркінгу як інструменту для наукових досліджень і індустріальних застосувань.

Отримані результати мають важливе значення для подальшого розвитку технологій ШІ, які сприяють підвищенню якості та ефективності LLM.

Ключові слова: ВЕЛИКІ МОВНІ МОДЕЛІ, БЕНЧМАРКІНГ, LLM, ПРОДУКТИВНІСТЬ, ТОЧНІСТЬ, API, ШТУЧНИЙ ІНТЕЛЕКТ.

ABSTRACT

This thesis is devoted to a comprehensive analysis of modern large language models of artificial intelligence and the development of an automated benchmarking system for evaluating their quality, performance, and functionality. The thesis examines in detail the architectural features of leading large language models (LLMs), methods for training them on large-scale text datasets, and their main areas of application – from natural language processing to programming automation and mathematical calculations. Particular attention is paid to the selection of LLMs for benchmarking, the process of integrating models via API is described, as well as methods of connecting and processing data for high-quality and comprehensive testing.

A proprietary benchmarking system was developed to provide automated and standardized testing of language models across various task categories, including general knowledge, logical reasoning, programming, mathematical calculations, and other specialized areas. The system integrates modern quality assessment metrics such as accuracy, completeness, response speed, and also measures the ability of models to generate contextually correct and logically consistent information.

Comprehensive testing allowed us not only to compare the performance and accuracy of the selected models, but also to identify their strengths and weaknesses, behavioral characteristics in specific types of tasks, and effectiveness in real-world application scenarios. The results confirm the high competitiveness of the models studied and demonstrate the practical usefulness of the developed benchmarking system as a tool for scientific research and industrial applications.

The results obtained are important for the further development of AI technologies that contribute to improving the quality and efficiency of LLM.

Keywords: LARGE LANGUAGE MODELS, BENCHMARKING, LLM, PERFORMANCE, ACCURACY, API, ARTIFICIAL INTELLIGENCE.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

LLM – Large Language Model

RNN – Recurrent Neural Network

LSTM – Long Short-Term Memory

API – Application Programming Interface

ARC – Abstraction and Reasoning Corpus

GPT – Generative Pre-trained Transformer

ROUGE – Recall-Oriented Understudy for Gisting Evaluation

BLEU – Bilingual Evaluation Understudy

MMLU – Massive Multitask Language Understanding

GPQA – General Purpose Question Answering

MBPP – Mostly Basic Python Problems

SWE-bench – Software Engineering Benchmark

AIME – Artificial Intelligence Math Evaluation

CLUEWSC – Chinese Language Understanding Evaluation - Winograd Schema Challenge

DROP – Discrete Reasoning Over Paragraphs

GSM8K – Grade School Math 8K

ЗМІСТ

ВСТУП

Швидкий розвиток та поширення великих мовних моделей викликав потребу в порівнянні їх якісних показників для визначення оптимальних інструментів для виконання конкретних задач: кодингу, створення цифрових творів мистецтва, наукових досліджень тощо. Оскільки переважно моделі працюють за принципом «чорного ящика», тобто не дають логічних пояснень отриманих результатів, дуже важливим стає вироблення чітких та ефективних критеріїв і метрик для порівняння. Проблема достовірності та змістовності відповідей залишається актуальною через широке використання ШІ-чатів в освіті, науці та технологічних галузях.

Щоб досягти кращої ефективності в даних сферах, дуже важливим є процес формулювання запитів до великих мовних моделей (large language models, LLM), або промпт-інжиніринг. Цей процес відкриває нові можливості для роботи зі штучним інтелектом (ШІ) у цілому, роблячи технології більш доступними та ефективними в використанні. Тому для швидкого розвитку LLM промпт-інженери створюють підказки для штучного інтелекту, які є невід'ємною частиною у його роботі. Саме промти вказують на те, як будуть реагувати різні мовні моделі на запити користувачів. Тому добре написаний промпт вдосконалює взаємодію між людиною та ШІ. Однак на етапах тестування промту, інженери стикаються з деякими проблемами: непередбачування результатів, складнощі з інтерпретацією внутрішніх алгоритмів моделі, відсутність стандартів і обмеження універсальних запитів тощо.

Звідси, бенчмаркінг великих мовних моделей є важлим завданням в подальшому навчанні ШІ та потребує подальшого дослідження, оскільки це дає змогу дослідити сильні та слабкі сторони моделей, оцінити точність та продуктивність в різних умовах використання. Це допомагатиме виробити правильні підходи до вдосконалення архітектур і різних методів навчання.

Об'єктом дослідження є сучасні великі мовні моделі. Дослідження зосереджено на їх архітектурних рішеннях, характеристиках продуктивності та

можливості застосування в різних галузях. Основна увага приділяється порівнянню за допомогою бенчмарків для визначення їх сильних і слабких сторін.

Предметом дослідження є методи бенчмаркінгу моделей. Дослідження охоплює підходи до порівняння моделей та виявлення факторів, які впливають на продуктивність в різних умовах застосування. Особлива увага приділяється швидкості генерації, точності, релевантності та використанню обчислювальних ресурсів.

Метою дослідження є систематизація актуальних метрик та необхідних процедур для здійснення бенчмаркінгу великих мовних моделей, аналіз сучасних програмних рішень з точки зору повноти функціональних можливостей, коректності та репрезентативності отриманих числових показників, а також розроблення власного програмного рішення (прототипу бенчмарка) для оцінки продуктивності й точності великих мовних моделей.

Практичні завдання в контексті вказаної тематики та мети дослідження такі:

1) розглянути ключові показники та критерії для оцінювання продуктивності великих мовних моделей;

2) сформувати перелік великих мовних моделей та методів оцінювання для порівняння їх роботи;

3) спроєктувати та підготувати набори промптів та тестових сценаріїв для кількісного порівняння продуктивності обраних LLM.

4) розробити прототип бенчмарка для автоматизації процесу тестування, провести аналіз отриманих результатів тестування.

РОЗДІЛ 1
ОГЛЯД ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ТА ЇХ МЕТОДІВ ОЦІНКИ

1.1 Методика вибору LLM та характеристика обраних моделей

Вибір великої мовної моделі (LLM) для проведення бенчмаркінгу є критично важливим, оскільки від нього залежить валідність, інформативність та інтерпретованість результатів. У сучасних умовах дослідники та інженери зазвичай обирають між потужними закритими моделями, такими як GPT-4 або GPT-4o від OpenAI, та відкритими альтернативами, такими як DeepSeek R1.

Насамперед вибір моделі залежить від типу завдань, які будуть тестуватися. Якщо бенчмарк орієнтовано на загальний рівень здібностей, наприклад, задачі MMLU (Massive Multitask Language Understanding) або ARC (Abstraction and Reasoning Corpus), доцільно використовувати універсальні моделі широкого профілю, як GPT-4o або DeepSeek R1. Якщо ж задачі передбачають роботу в режимі діалогу, обробку багатомодальних даних або генерацію коду, важливо враховувати спеціалізацію моделі [1]. Серед актуальних великих мовних моделей для дослідження можна обрати такі:

– GPT-4o – це мультимодальна модель з підтримкою тексту, аудіо та зображень, що демонструє вищу швидкодію та доступність порівняно з попередніми версіями.

– GPT-4 – добре збалансована за точністю, надійністю та підтримкою API, однак менш оптимізована для мультимодальних задач.

– DeepSeek R1 – відкрита модель з конкурентною продуктивністю на багатьох бенчмарках, оптимізована для локального використання, добре підходить для оцінювання на open-source інфраструктурі [2].

– Claude 3 (Anthropic) – серія потужних LLM із фокусом на безпечність, пояснюваність та якість генерації. Claude 3 Opus, наприклад, показує високі результати на MMLU та GSM8K [3].

– LLaMA 3 (Meta) – потужна відкрита модель із високою

продуктивністю у багатьох бенчмарках. Поширюється під умовною ліцензією для дослідницьких цілей. Підтримується у Hugging Face, Ollama, OpenLLM [3].

– Gemini (Google DeepMind) – флагманська мультимодальна модель від Google. Gemini Pro та Gemini 1.5 демонструють високу продуктивність на креативних і складних завданнях. Доступні через Google Cloud Vertex AI [4].

– Mistral 7B / Mixtral 8x7B – легковагові моделі з відкритим кодом, оптимізовані для швидкого логічного висновування (inference) при високій точності. Широко використовуються в локальних або крайових сценаріях.

Закриті моделі, такі як GPT-4 та GPT-4o, забезпечують високу якість результатів, але потребують доступу до API (OpenAI), що може створювати обмеження для офлайн-експериментів або кастомізації. Натомість DeepSeek R1 доступна у вигляді моделей з відкритим кодом, що робить її зручною для локального тестування, адаптації та масштабування.

GPT-4 і GPT-4o не потребують локального запуску, але вимагають стабільного API-доступу, відповідних тарифів та дотримання політик OpenAI. У той же час, DeepSeek R1 потребує для запуску наявність власної інфраструктури, однак це забезпечує повний контроль над середовищем роботи.

Аналогічні відмінності спостерігаються і серед інших моделей. Наприклад, Claude 3 (Anthropic) – це закрита модель зі значними досягненнями в розумінні інструкцій, однак також доступна лише через API. Моделі Gemini 1.5+ мають сильну інтеграцію з екосистемою Google та підтримку мультимодальності, проте недоступні для локального використання.

Натомість моделі LLaMA 3 (Meta) та Mistral – це відкриті LLM, які демонструють високу якість на багатьох задачах і можуть бути самостійно запущені на локальних або хмарних потужностях. Вони також доречні для кастомізації, донавчання або інтеграції в приватні системи.

Усі моделі підтримують стандартні формати текстових запитів та сумісні з більшістю сучасних бенчмарків – MMLU, DROP, HellaSwag, GSM8K тощо. Для GPT-4o, GPT-4, Claude 3 та Gemini доступні API для інтеграції з Python. DeepSeek R1, LLaMA 3, Mistral можна запускати локально через Hugging Face, OpenLLM

або інші open-source фреймворки.

ChatGPT – це одна з найвідоміших великих мовних моделей (LLM), розроблена компанією OpenAI на основі архітектури GPT-4. Основою GPT-4 є трансформерна архітектура, вперше запропонована Vaswani та колегами у 2017 році в роботі «Attention is all you need» [5]. Трансформер використовує механізм багатоголової уваги (multi-head attention), що дозволяє моделі ефективно розуміти контекст тексту, відстежуючи зв'язки між словами на різних відстанях [6]. Це суттєво покращує якість тексту в порівнянні з попередніми підходами, такими як рекурентні нейронні мережі RNN чи LSTM.

Архітектура трансформера стала проривною інновацією в обробці послідовних даних, значно перевершуючи традиційні рекурентні нейронні мережі (RNN) та їх різновиди [7]. Головною особливістю трансформерів є механізм самоуваги (self-attention), який дає змогу моделі ефективно оцінювати важливість кожного елемента у вхідній послідовності, незалежно від його позиції. Це означає, що трансформери не потребують послідовної обробки інформації, що істотно підвищує швидкість і якість аналізу даних.

Як показано на рис. 1.1, основні компоненти архітектури трансформера взаємодіють між собою для перетворення вхідної послідовності у вихідний результат. Спершу, етап вхідних ембеддингів (input embeddings) конвертує послідовність токенів (наприклад, слів речення) у числові вектори, які зберігають семантичну та синтаксичну інформацію. Кожен токен трансформується у вектор у багатовимірному просторі, що дає змогу моделі навчатися розпізнавати відношення між словами.

Для ефективного використання великих мовних моделей у прикладних задачах, необхідно забезпечити зручну інтеграцію між LLM та зовнішніми системами. З цією метою більшість сучасних моделей надаються через програмні інтерфейси (API), що дає змогу викликати модель із зовнішнього середовища, передавати їй запити (промпти) та отримувати результати у відповідь. Наприклад, OpenAI надає доступ до моделей GPT через OpenRouter, де користувач може надсилати HTTP-запити із зазначенням параметрів, таких

як prompt, temperature, max_tokens, stop_sequence тощо.

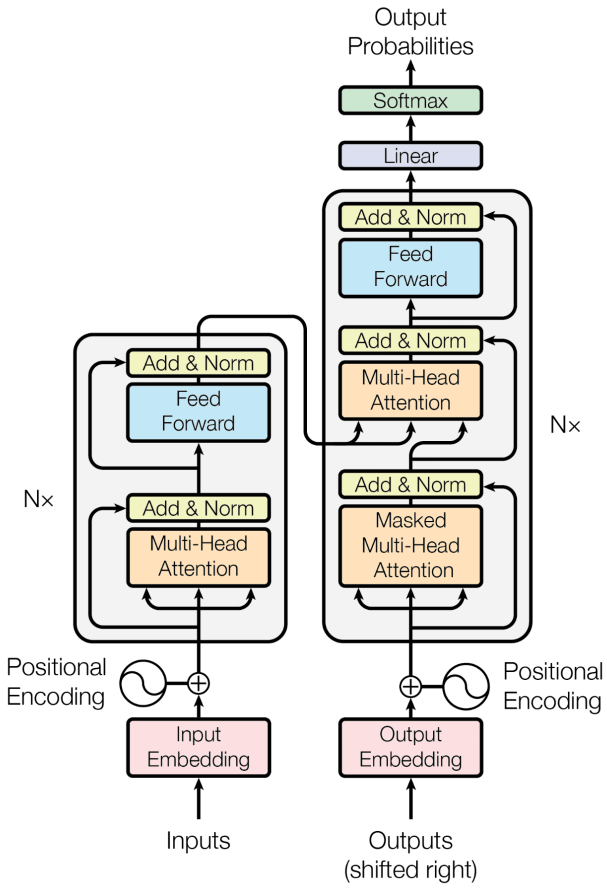


Рисунок 1.1 – Діаграма основних складників архітектури моделі трансформера [8]

Google Gemini – це передова мультимодальна велика мовна модель, розроблена компанією Google, яка поєднує потужність трансформерної архітектури з можливістю одночасної роботи з різними типами даних: текстом, зображеннями, аудіо і навіть відео [9]. Ця модель є наступним кроком у розвитку лінійки великих мовних моделей Google і створена для задоволення зростаючих вимог до інтелектуальних систем, здатних розуміти та генерувати контент у кількох форматах одночасно.

Архітектурно Google Gemini базується на глибоких трансформерах, які дозволяють обробляти великі об’єми інформації з високою ефективністю, підтримуючи контекстуальне розуміння на рівні, що перевищує можливості

попередніх моделей [10]. Завдяки мультимодальності, Gemini може працювати з різнорідними даними, що відкриває нові горизонти для застосування у таких сферах, як автоматичний аналіз зображень разом із текстом, розпізнавання та генерування аудіо, а також обробка відео в реальному часі.

Впровадження Google Gemini в систему бенчмаркінгу великих мовних моделей дозволяє значно розширити спектр оцінюваних завдань і підвищити точність тестування. Завдяки підтримці мультимодальних вхідних даних, можна тестувати модель не лише на класичних текстових датасетах, але й на комбінаціях різних типів інформації, що робить оцінювання більш комплексним і наближеним до реальних застосувань. Для цього необхідно підготувати спеціалізовані набори даних, які включатимуть розмічені тексти, зображення, аудіо або відео залежно від типу задачі, а також визначити метрики оцінки якості генерації чи розуміння у кожному з режимів [11].

Підключення Google Gemini для бенчмаркінгу можливе через офіційний API Google Cloud, який надає доступ до функціональності моделі в вигляді RESTful-сервісу. Цей API дає змогу надсилати запити з текстом або мультимодальними даними і отримувати відповіді у структурованому форматі JSON, що зручно для автоматизованої обробки результатів у системах тестування. Для інтеграції необхідно зареєструватися в Google Cloud, створити проєкт, активувати Gemini API та отримати ключ доступу (API key), який потім використовується в запитах [12].

Окрім офіційного API, Google Gemini можна підключити через універсальні маршрутизатори API, такі як OpenRouter. Це платформа, що централізує доступ до кількох великих мовних моделей від різних провайдерів, включно з Google Gemini, OpenAI GPT, Anthropic Claude тощо. Використання OpenRouter дозволяє спростити інтеграцію, уніфікувати виклики API, керувати лімітами та відстежувати статистику звернень в одному місці. Це особливо корисно при порівняльному бенчмаркінгу, де одна система тестує одночасно кілька моделей, отримуючи результати через уніфікований інтерфейс.

Підготовка даних для бенчмаркінгу Google Gemini передбачає кілька

ключових етапів. По-перше, необхідно зібрати якісні та різноманітні датасети, які відображають реальні умови роботи моделі. Для мультимодального тестування це можуть бути поєднання текстів з відповідними зображеннями чи аудіозаписами. По-друге, важливо належним чином анотувати дані, тобто забезпечити точне розмічення для оцінки правильності відповідей моделі. По-третє, дані мають бути підготовлені у форматах, сумісних з API (наприклад, JSON з відповідними полями для різних типів вхідних даних) [12]. Важливо також забезпечити різноманітність тем і стилів, щоб перевірити здатність моделі адаптуватися до різних контекстів.

Додатково для автоматизації процесу бенчмаркінгу з Google Gemini можна застосувати спеціалізовані бібліотеки клієнтів, які полегшують формування запитів та обробку відповідей. Такі бібліотеки зазвичай підтримують роботу з API ключами, аутентифікацію, управління чергами запитів та масштабування навантаження [13]. Інтеграція із системами логування і моніторингу дозволяє отримувати детальні звіти про продуктивність і якість моделі в реальному часі.

Таким чином, Google Gemini представляє собою потужний інструмент для сучасного бенчмаркінгу великих мовних моделей, що поєднує гнучкість мультимодального аналізу з високою точністю генерації і розуміння даних різних форматів. Його підключення через офіційний API або універсальні платформи на OpenRouter забезпечує простоту інтеграції й масштабування, а правильна підготовка даних гарантує коректність і релевантність оцінювання.

1.2 Методи оцінювання та підготовка даних для бенчмаркінгу

Застосування великих мовних моделей у різних галузях вимагає ретельної оцінки їх продуктивності та виявлення потенційних ризиків. Рамкова структура оцінювання LLM допомагає розробникам і дослідникам визначити ефективність моделей, порівняти їх між собою та виявити напрямки для покращення [14]. Першим кроком у процесі оцінки є встановлення цілей та завдань: визначення

сфери застосування моделі та формулювання критеріїв оцінювання. Вони охоплюють метрики для вимірювання точності та продуктивності у базових задачах обробки природної мови, таких як переклад, підсумовування тексту та відповіді на запитання; оцінку розуміння мови та генерації тексту для заданих вхідних речень; визначення загальних і спеціалізованих знань у конкретній доменній області; виявлення упереджень або ризиків, пов’язаних з етичними порушеннями чи питаннями безпеки; порівняння продуктивності з людськими показниками або іншими великими мовними моделями [14].

Цілі оцінювання мають бути чіткими, конкретними та вимірюваними, із прямим зв’язком із призначенням моделі. Наприклад, модель, створена для медичної сфери, може оцінюватися за рівнем володіння медичними знаннями, дотриманням стандартних протоколів і ключовими показниками ефективності [14].

Після визначення цілей наступним кроком є вибір метрик для вимірювання їх досягнення. Саме за їх допомогою можна об’єктивно визначити, наскільки модель відповідає поставленим завданням. Зазвичай ці показники поділяються на три основні категорії: метрики точності, метрики вільного володіння та метрики стійкості [14].

Метрики точності походять із традиційних задач класифікації. Крім показника точності (відношення коректних передбачень до їх загальної кількості), в машинному навчанні однією з найпопулярніших метрик для вимірювання точності моделі є матриця невідповідностей (confusion matrix), представлена на рис. 1.2. Вона є наочним поданням ефективності класифікаційного алгоритму у вигляді таблиці, де порівнюються фактичні (істинні) мітки класів із передбаченими моделлю. Ця матриця дозволяє оцінити роботу класифікатора та слугує основою для обчислення таких показників продуктивності, як точність (precision), повнота (recall) та F1-міра [14].

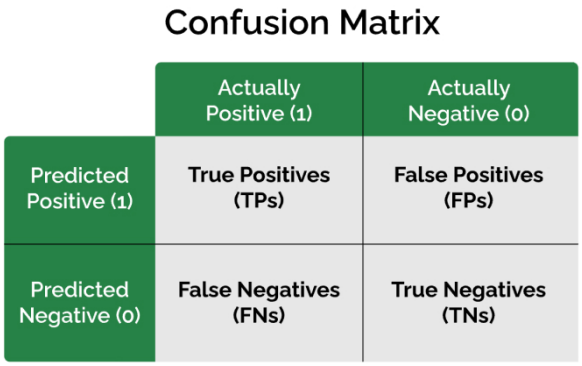


Рисунок 1.2 – Матриця невідповідностей

Повнота (Recall) – це частка правильно передбачених позитивних результатів серед усіх фактичних позитивних прикладів. Повнота є особливо важливою метрикою в випадках, коли пропущення позитивного випадку може мати серйозні наслідки, наприклад, при виявленні хвороби або шахрайських дій. Модель із високим показником повноти прагне зменшити кількість хибнонегативних результатів, що особливо критично в медичній діагностиці чи системах безпеки.

F1-міра (F1 Score) – це середнє гармонійне між точністю та повнотою, що забезпечує збалансовану оцінку обох метрик. Цей показник особливо корисний у задачах із незбалансованими класами, де важливо одночасно враховувати як кількість правильно виявлених позитивних результатів, так і уникнення хибних спрацювань.

Метрика вільного володіння мовою (fluency) дає можливість оцінити природність і достовірність згенерованої мови. Вона допомагає визначити, наскільки текст звучить автентично: чи відповідає граматичним нормам, має логічну структуру та стилістичну узгодженість [14].

Ще однією з базових метрик є перплексія (perplexity). Вона вимірює, наскільки добре модель передбачає наступне слово в послідовності. Нижчі значення перплексії свідчать про вищу точність передбачень, тобто текст, згенерований моделлю, сприймається як більш природний та зв’язний. Зразок використання метрики, реалізований засобами мови Python, наведено в лістингу 1.1.

Лістинг 1.1 – Скрипт роботи метрики perplexity

```
import lmppl
scorer = lmppl.LM('gpt2')
text = [
'sentiment classification: I dropped my laptop on my knee, and someone stole my coffee. I am happy.',
'sentiment classification: I dropped my laptop on my knee, and someone stole my coffee. I am sad.'
]
ppl = scorer.get_perplexity(text)
print(list(zip(text, ppl)))
```

Ще однією важливою метрикою для оцінки якості машинного перекладу є метрика BLEU (Bilingual Evaluation Understudy). Показник Reference (еталонний переклад) – це список списків слів, що відображає правильний переклад. Метрика BLEU підтримує кілька референсних перекладів, однак у цьому випадку розглядається лише один. Показник Candidate (кандидатський переклад) – це список слів, який представляє переклад, створений моделлю, або речення, яке потрібно оцінити [14]. Уривок коду з програмним використанням метрики BLEU показано в лістингу 1.2.

Лістинг 1.2 – Скрипт роботи метрики BLEU

```
from nltk.translate.bleu_score import sentence_bleu
reference = [['this', 'movie', 'was', 'awesome']]
candidate = ['this', 'movie', 'was', 'awesome', 'too']
score = sentence_bleu(reference, candidate)
print(score)
>>> 0.668740304976422
```

Значення метрики BLEU коливається від 0 до 1, де 1 означає ідеальну відповідність між кандидатським і референсним перекладом. У наведеному прикладі результат 0.6687 свідчить про доволі високу схожість речень. Проте він не дорівнює 1, оскільки в кандидатському перекладі є зайве слово “too” в кінці. BLEU штрафує як за пропущені, так і за зайві слова в перекладі.

Оцінювання читабельності та зв’язності тексту передбачає залучення людей для оцінки текстів, згенерованих великою мовною моделлю. Цей метод вважається найефективнішим для перевірки якості мови, оскільки люди можуть помічати особливості, які машинні алгоритми пропускають. Зазвичай оцінюють

такі ключові аспекти, як читабельність (наскільки текст зрозумілий для читача, тобто наскільки легко його читати) та зв'язність (наскільки плавно і логічно матеріал побудований та пов'язаний у фінальному тексті).

Набір показників ROUGE (Recall-Oriented Understudy for Gisting Evaluation) використовується для автоматичного оцінювання якості рефератів та машинного перекладу. Основний принцип обчислення полягає у визначенні кількості збігів одиниць тексту (n-грам, послідовності слів або пар слів) між автоматичним і референсним текстом. Існує декілька варіантів ROUGE. Наприклад, ROUGE-N вимірює схожість n-грам між системним та референсними даними. З іншого боку, ROUGE-L оцінює найдовшу спільну підпослідовність (Longest Common Subsequence, LCS) між згенерованим і референсним текстом. Ця метрика не потребує безперервних збігів, а враховує збіги, що йдуть у послідовному порядку. Ідея полягає в тому, що довша спільна підпослідовність свідчить про більшу схожість текстів. Функціонування метрики ROUGE-L ілюструється на рис. 1.3.

Рисунок 1.3 – Функціонування метрики ROUGE-L

Версія метрики ROUGE-S обчислює перекриття пропущених біграм (skip-bigram) між системним рефератом та референсними рефератами. ROUGE-S дає змогу додати певну ступінь гнучкості до порівняння n-грам. Як і в випадку з ROUGE-N, тут обчислюється повнота, але з урахуванням можливих слів, які можуть з'являтися між елементами, що продемонстровано на рис. 1.4.

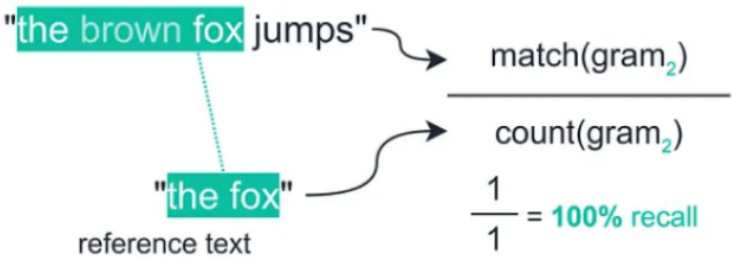


Рисунок 1.4 – Функціонування метрики ROUGE- S [14]

Аналогічний підхід можна застосувати і до метрики точності (precision), як показано на рис. 1.5.

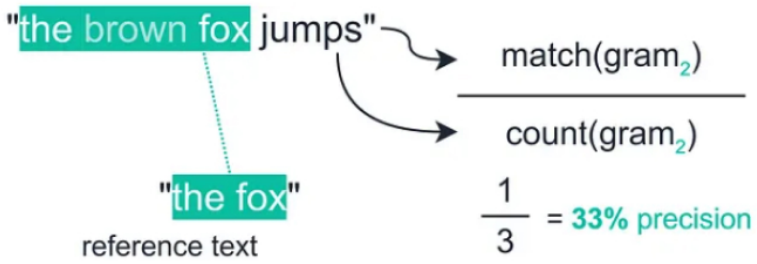


Рисунок 1.5 – Функціонування метрики точності за допомогою ROUGE-S [14]

Метрики стійкості (Robustness Metrics) оцінюють продуктивність моделі в різних умовах, при роботі з різними перефразуваннями одного й того ж вхідного запиту. Вона перевіряє, наскільки результат моделі залежить від формулювання вхідного тексту, як показано на рис. 1.6.

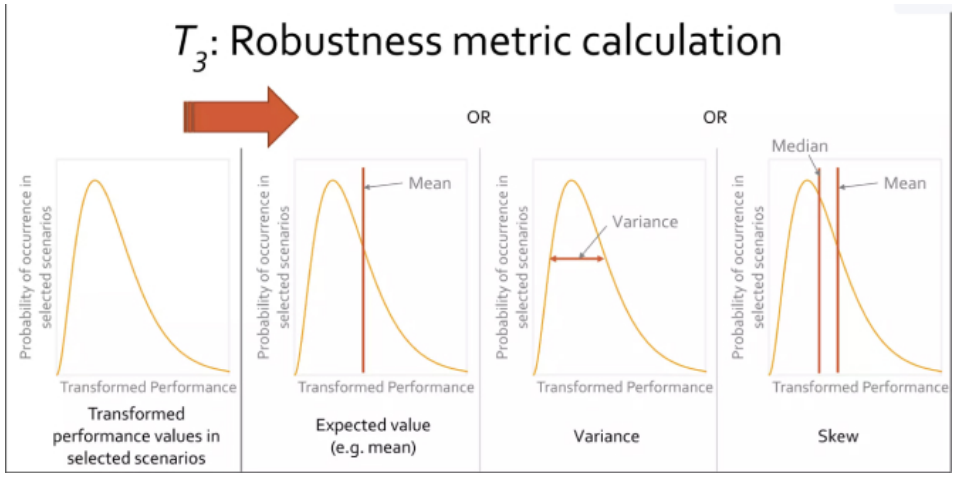


Рисунок 1.6 – Обчислення метрики стійкості

Оцінка узгодженості (Consistency Score) показує відношення результатів, які залишаються незмінними при різних перефразуваннях одного й того ж вхідного запиту, відображаючи здатність моделі зберігати зміст при варіаціях формулювання.

Стійкість до змін у вхідних даних (Stability to Input Perturbations). Ця метрика демонструє, наскільки вихідні результати моделі змінюються під впливом незначних і несуттєвих варіацій у вхідних даних. Вона дозволяє оцінити здатність моделі стабільно працювати навіть у присутності шуму або інших малозначущих змін.

Показник стабільності (Stability Score) вимірює, наскільки стабільним є вихід моделі при невеликих змінах у вхідних даних. Вищі значення цієї метрики свідчать про меншу чутливість моделі до таких змін. Це можна подати у вигляді формули 1.1.

$$R(x, y) = \min_{\|\delta\| \leq \epsilon} \{\|\delta\| : f(x + \delta) \neq y\} \quad (1.1)$$

де $R(x, y)$ – стійкість моделі f при вхідних даних x з істинною міткою y; δ – шум або збурення, яке додається до вхідних даних x; ϵ – максимально допустимий розмір (норма) збурення; $f(x + \delta)$ – вихід моделі після застосування збурення до вхідних даних. Це рівняння описує спосіб вимірювання мінімального збурення в межах заданої норми ϵ, яке може призвести до неправильного класифікування вхідних даних x моделлю. Якщо модель залишається стійкою до збурень в межах допустимого ліміту ϵ, її вважають стійкою до атак (adversarially robust).

Загалом ретельно побудована система оцінювання не лише показує ефективність моделі, але й описує її поведінку, визначає правильні підходи до розроблення та сприяє відповідальному й корисному застосуванню великих мовних моделей у суспільстві [14].

Як і в будь-якій задачі машинного навчання, успіх значною мірою

залежить від якості підготованих даних. Саме завдяки даним модель формує уявлення про реальний світ і здобуває здатність виконувати складні завдання після навчання. Також ключовим є те, як саме обробляються ці дані: якість очищення та ефективність побудованого пайплайна істотно впливають на кінцевий результат. Підготовка масштабного датасету може бути дорогою, якщо не оптимізувати всі етапи.

Загальний процес охоплює кілька важливих фаз, від збору та структурування до перевірки та перетворення даних. Кожна з них потребує відповідних інструментів і рішень для досягнення максимальної ефективності.

Основні вимоги до тестових даних у LLM-бенчмарках

1. Ізоляція від тренувального корпусу. Один із головних принципів – гарантія того, що жодна частина тестового набору не входить до навчального масиву моделі. Це виключає «запам’ятовування» й дозволяє вимірювати реальну здатність моделі до генералізації.

2. Баланс тематик і рівнів складності. Відомі бенчмарки, зокрема MMLU, спеціально будуються за принципом рівномірного покриття дисциплін – від математики до права, та рівнів – від середньої школи до вищої освіти .

3. Стандартизація формату. Типовим підходом є використання формату JSONL, де кожен запис включає інструкцію, опціональні варіанти відповідей, правильну відповідь, метаінформацію (тему, рівень складності, джерело тощо).

4. Перевіреність та валідація. Бенчмарки високої якості передбачають експертну перевірку відповідей. У DROP, наприклад, еталонні відповіді формувалися на основі вичитаних уривків з тексту, що дозволяє уникати неоднозначностей.

5. Уніфікованість форматів і інтерфейсів. Більшість сучасних LLM-бенчмарків використовують структури, що легко інтегруються в автоматизоване тестування: це, зокрема, набори даних HuggingFace, Apache Arrow, або CSV із чітко визначеними колонками [15].

Підготовка даних до бенчмаркування моделей проходить кілька

послідовних стадій:

1. Добір джерел: використовуються або наявні датасети (MMLU, HumanEval, DROP, MBPP, SWE-bench), або створюються власні корпуси. При самостійному формуванні важливо дотримуватись принципів ізоляції, охоплення та тематичного балансу.

2. Очищення та фільтрація: на цьому етапі видаляються дублі, приклади з некоректними формулюваннями або неоднозначними відповідями. Також виконується перевірка на потенційне дублювання з тренувальними корпусами, що є критичним для забезпечення чистоти експерименту.

3. Анотування: до кожного прикладу додається метаінформація – категорія знань, мова, рівень складності, тип завдання (множинний вибір, відкрите питання, генерація коду тощо). Такий підхід дозволяє аналізувати продуктивність моделі в окремих доменах.

4. Стандартизація формату: дані перетворюються у машиночитні формати. Найбільш уживаними є JSONL, CSV, а також готові об’єкти бібліотеки datasets (HuggingFace), які мають вбудовані механізми перевірки типів і схем [15].

5. Фінальна валідація: перевіряється якість еталонних відповідей, збалансованість тематики, відсутність витоків з тренувального корпусу. У випадку з HumanEval додатково застосовується перевірка результатів шляхом автотестів для кожного згенерованого фрагменту коду.

Підготовка великих тестових корпусів часто супроводжується використанням автоматизованих інструментів:

1. HuggingFace Datasets – бібліотека з інтегрованими бенчмарками, що підтримує завантаження, трансформацію й фільтрацію наборів даних.

2. Apache Arrow – використовується як високошвидкісний формат представлення табличних даних у деяких фреймворках.

3. OpenLLM tools, BIG-bench, EleutherEval – фреймворки для оцінювання моделей із можливістю створення кастомних тестів.

Таким чином, якість бенчмаркування великих мовних моделей

визначається не лише вибором метрик, а передусім, правильністю підготовки тестових даних. Наявність чистих, збалансованих і добре анотованих прикладів забезпечує об’єктивність оцінки.

1.3 Існуючі бенчмарки LLM

Завдяки поєднанню відкритих та власних бенчмарків організації отримують уявлення про ефективність LLM у таких сферах, як багатозадачне навчання, питання-відповіді та програмування. Нижче розглядаються найпоширеніші бенчмарки для оцінки сучасних штучних інтелектів і їхня роль у забезпеченні чесної та уніфікованої оцінки моделей.

MMLU (Massive Multitask Language Understanding) створений для оцінки здатності великої мовної моделі узагальнювати знання з різноманітних галузей – від історії та права до математики й програмування. Це один із найвідоміших і найцитованіших бенчмарків, який охоплює 57 предметних областей різного рівня складності. Основною метою MMLU є перевірка того, наскільки добре модель здатна узагальнювати інформацію та орієнтуватися у міждисциплінарних знаннях. Типовими завданнями можна вважати тести з множинним вибором відповідей із галузей науки, гуманітарних і професійних дисциплін. Критеріями оцінки виступають точність вибору правильної відповіді у порівнянні з базовими людськими показниками.

У межах порівняння мовних моделей MMLU дає глибоке уявлення про здатність моделей зберігати, узагальнювати й застосовувати знання, тому він є одним із ключових інструментів як для відкритих, так і для внутрішніх оцінок LLM [16]. Приклад роботи бенчмарка наведено в лістингу 1.3.

Лістинг 1.3 – Приклад роботи MMLU

```
Category: "Physics"
Question: "Which instrument is used to measure atmospheric pressure?"
Options: [A) Thermometer, B) Barometer, C) Hygrometer, D) Scale]
Correct Answer: B) Barometer
```

Інструмент GPQA (General Purpose Question Answering) є одним із ключових інструментів для оцінки здатності LLM до логічного міркування. На відміну від класичних наборів даних для запитань і відповідей, GPQA зосереджений на складних багатокрокових завданнях, що вимагають аналізу, синтезу й обґрунтування відповіді. Структурно GPQA орієнтований на оцінку міркувань: модель має вміти аналізувати контекст, знаходити релевантні факти та робити логічні висновки. Система балів враховує не лише точність, а й логічну послідовність, акцентуючи на глибині розуміння, а не на запам'ятовуванні. Також можливе порівняння з людиною, GPQA слугує еталоном «експертного мислення», дозволяючи виміряти, наскільки модель наближена до людського рівня логіки.

GPQA часто використовується в порівняльному аналізі мовних моделей, допомагаючи виявити різницю в рівні логічного мислення та вузькі місця у здатності до дедуктивного висновку [16]. Приклад роботи бенчмарка наведено в лістингу 1.4.

Лістинг 1.4 – Приклад роботи GPQA

```
Category: "Physics"
Question: "What is the speed of light in vacuum?"
Options: [A) 3×10^5 km/s, B) 3×10^8 m/s, C) 3×10^6 m/s, D) 3×10^4 km/s]
Correct Answer: B) 3×10^8 m/s
```

У контексті зростаючої ролі ШІ в розробці програмного забезпечення, бенчмарк HumanEval став одним із основних інструментів для оцінки здібностей LLM до програмування. Цей тест, запропонований OpenAI, дозволяє перевірити, наскільки ефективно модель генерує коректний і працездатний код на основі текстових інструкцій. Типовими завданнями для цього інструменту такі: моделі надається опис проблеми, на основі якого вона має згенерувати функцію мовою Python, що проходить заздалегідь визначені модульні тести. Результат вважається успішним, якщо згенерований код працює коректно та виконує всі тестові кейси. Результати HumanEval активно використовуються для рейтингу моделей ШІ за рівнем здатності до програмування.

HumanEval є цінним інструментом для компаній, які прагнуть об’єктивно виміряти програмістські навички LLM, особливо в задачах середнього рівня складності. У міру інтеграції ШІ в робочі процеси розробки цей бенчмарк слугує надійним індикатором здатності моделі до генерації реального коду [16]. Приклад роботи бенчмарка наведено в лістингу 1.5.

Лістинг 1.5 – Приклад роботи HumanEval

```
Function: "def add(a, b):"
Test: "assert add(2,3) == 5"
```

MBPP (Mostly Basic Python Problems) – це ще один поширений бенчмарк для оцінки генерації коду, що містить понад 900 програмувальних завдань. Подібно до HumanEval, цей тест перевіряє функціональну коректність коду шляхом запуску набору тестів. Формат завдань MBPP – це короткі задачі на Python з чіткими умовами, орієнтовані на базовий рівень програмування. Моделі тестуються у few-shot (з кількома прикладами) та fine-tuned (попередньо натренованих) сценаріях. Основними метриками для оцінювання виступають:

– Pass@1 (або Task Success Rate) – частка завдань, які вирішує хоча б один варіант відповіді;

– Sample Success Rate – частка окремих згенерованих прикладів, що проходять усі тести.

MBPP дає змогу оцінити базовий рівень здатності моделей LLM до кодування і доречний для навчання та бенчмаркінгу в початкових або прикладних програмних завданнях [16]. Приклад роботи бенчмарка наведено в лістингу 1.6.

Лістинг 1.6 – Приклад роботи MBPP

Category: "Physics"

Question: "Which instrument is used to measure atmospheric pressure?"

Options: [A) Thermometer, B) Barometer, C) Hygrometer, D) Scale]

Correct Answer: B) Barometer

SWE-bench – це спеціалізований бенчмарк для оцінки здатності LLM до генерації коду, подібно до HumanEval, але з фокусом на реальні задачі програмної інженерії. Його головна мета – перевірити, чи може модель виправити помилку або реалізувати нову функцію у вже наявному коді. До основних перевірочних задач можна віднести внесення змін у реальні репозиторії на основі опису проблеми або запиту на зміну функціональності. Моделі отримують опис задачі (issue) і частину коду та мають внести релевантні зміни. Метрикою оцінювання стає частка успішно виконаних завдань, тобто таких, де згенеровані зміни проходять перевірку і вирішують поставлену проблему.

SWE-bench є важливим інструментом для оцінки інженерної коректності та практичної користі LLM у підтримці програмного забезпечення, що робить його актуальним для застосування в реальних розробницьких середовищах. Приклад роботи бенчмарка наведено в лістингу 1.7.

Лістинг 1.7 – Приклад роботи SWE-bench

```
Category: " Software Engineering"
Issue: "The function 'divide' crashes when the denominator is zero. Modify
it to raise an exception instead of crashing."
Code Snippet:
def divide(a, b):
    return a / b
Corrected Code:
def divide(a, b):
    if b == 0:
        raise ValueError("Denominator cannot be zero")
    return a / b
```

Іншим важливим підходом для оцінки великих мовних моделей є DROP (Discrete Reasoning Over Paragraphs) – тест, що перевіряє здатність моделей виконувати дискретне міркування над текстовими абзацами. Цей бенчмарк містить близько 96 000 запитань, сформованих на основі текстів Вікіпедії та краудсорсингових даних, зібраних через Amazon Mechanical Turk. Типовими завданнями є запитання, що вимагають від моделей здійснювати математичні операції (додавання, віднімання, порівняння) на основі інформації, розпорошеної по текстовому уривку. Моделі повинні ідентифікувати кілька

чисел у тексті, поєднати їх відповідно до умови задачі і отримати правильну відповідь. Передові моделі, такі як GPT-4 та PaLM, демонструють точність близько 80-85%, тоді як показник людини на цьому наборі даних становить близько 96%. DROP є ключовим тестом для оцінки логічного та арифметичного мислення LLM, що особливо важливо для застосувань, де потрібен аналіз і обробка структурованої інформації з текстів [17]. Приклад роботи бенчмарка наведено в лістингу 1.8.

Лістинг 1.8 – Приклад роботи DROP

```
Category: "Reading Comprehension"
Passage: "Emma had 20 candies. She gave 8 to her friend and then bought 5 more candies."
Question: "How many candies does Emma have now?"
Correct Answer: 17
Explanation: (20 - 8) + 5 = 17
```

1.4 Постановка завдання на розробку

Для розробки в межах кваліфікаційної роботи пропонується створення системи бенчмаркінгу для оцінювання якості роботи великих мовних моделей GPT-4o, GPT-4 та Gemini 2.5 Turbo. За результатами огляду сучасних інструментів і сервісів було прийнято рішення використовувати наступні технології та підходи:

1) Python як одну з найпопулярніших мов програмування загального призначення, яка відома своєю простотою та читабельністю коду;

2) requests – популярна бібліотека для роботи з HTTP-запитами у Python, яка дозволяє легко надсилати GET, POST та інші типи запитів до вебсервісів, що дуже зручно для взаємодії з API таких сервісів, як OpenRouter.

3) Pandas – бібліотека для обробки та аналізу даних, яка надає зручні структури даних, такі як DataFrame;

4) openai – офіційна клієнтська бібліотека для взаємодії з API OpenAI, що забезпечує зручний доступ до моделей GPT-4, GPT-4o та ін.;

Для автоматизації інфраструктури передбачене застосування OpenRouter.

Це сучасний сервіс, який надає API-інтерфейс для роботи з різними великими мовними моделями (LLM) через уніфікований інтерфейс. OpenRouter дозволяє легко інтегрувати та переключатися між різними мовними моделями (наприклад, GPT-4, GPT-4o, Gemini 2.5 Pro тощо), що спрощує розробку застосунків з підтримкою штучного інтелекту. Сервіс забезпечує масштабованість, безпеку та гнучкість у виборі моделей, а також дозволяє керувати запитами, обліковими записами та моніторингом використання.

РОЗДІЛ 2
РОЗРОБКА ТА РЕАЛІЗАЦІЯ СИСТЕМИ БЕНЧМАРКІНГУ

2.1 Вимоги до програмного рішення

Під час розробки програмного рішення для бенчмаркінгу великих мовних моделей буде враховано низку функціональних та нефункціональних вимог, які забезпечать ефективне тестування моделей GPT-4, GPT-4o та Gemini 2.5 Turbo. Програмне рішення має забезпечити інтеграцію з API трьох моделей, а саме GPT-4, GPT-4o та Gemini 2.5 Turbo, що дозволить здійснювати запити й отримувати відповіді для подальшого аналізу. Основна увага буде спрямована на оцінку продуктивності та точності відповіді моделей за допомогою різних видів бенчмарків з майбутнім аналізом отриманих результатів. В системі буде реалізовано можливість запуску різних бенчмарків, які охоплюють широкий спектр завдань: загальні знання, програмування, математику та мовні специфіки. Зокрема, до таких тестів належать MMLU, DROP, GPQA-Diamond, HumanEval, LiveCodeBench, MATH-500, AIME, CLUEWSC та C-Eval тощо.

Для кожного бенчмарка будуть підготовлені відповідні тестові набори, що міститимуть вхідні дані і еталонні відповіді, які слугуватимуть основою для оцінки якості моделей. Запити до моделей будуть формуватися в вигляді підказок (prompts), що адаптуватимуться під особливості конкретного бенчмарку та моделі, із контролем обмежень по часу обробки та обсягу токенів. Відповіді, отримані від моделей, система аналізуватиме за різними критеріями: від точності збігу з правильними відповідями, через коректність згенерованого коду для програмних завдань, до правильності розрахунків у математичних тестах та якості відповідей у мовних завданнях.

Результатом роботи системи буде формування звіту, який міститиме показники точності, середній час відповіді, кількість оброблених прикладів та ілюстрації відповідей. Крім того, програмне рішення передбачатиме обробку помилок, включаючи збої API, тайм-аути, відсутність відповіді та інші виняткові

ситуації, з відповідним логуванням і механізмами повторних спроб. Для користувача буде реалізовано механізм автоматичної перевірки і налаштування залежностей, що забезпечить запуск системи без зайвих труднощів.

Щодо нефункціональних вимог, то планується створити систему таким чином, щоб вона забезпечувала надійність і стабільність роботи, була здатна працювати безперервно навіть за умов часткових збоїв зв'язку або отримання некоректних відповідей. Вся робота системи супроводжуватиметься логуванням, що дозволить відслідковувати процес тестування та оперативно виявляти можливі проблеми. Збереження результатів відбуватиметься в форматі CSV та JSON, придатному для подальшої обробки і аналізу. Важливо, що більшість обробки тестових даних і логіки виконуватиметься локально, а зовнішні залежності будуть мінімальними і обмежуватимуться виключно запитами до API моделей.

2.2 Сценарії використання та вибір метрик для оцінювання

Під час розробки системи буде передбачено кілька ключових сценаріїв використання, які максимально відображатимуть реальні потреби користувачів і завдання, що стоять перед розробниками та дослідниками ШІ. Основним сценарієм стане порівняльний аналіз моделей GPT-4, GPT-4o та Gemini 2.5 Turbo, у якому система дозволить запускати стандартизовані бенчмарки для оцінки їхньої продуктивності, точності та здатності роботи у різних типах інформації. Це буде корисно для вибору найефективнішої моделі для конкретних завдань: від роботи із знаннями та математичними задачами до складних програмних інженерних кейсів і мовних специфік, таких як китайська мова. Також передбачене забезпечення можливості глибокого аналізу якості відповідей моделей у різних контекстах, що дасть змогу дослідникам оцінювати не лише кількість правильних відповідей, а й їхню релевантність, повноту й логічність. Система буде використовуватись для регулярного моніторингу продуктивності моделей у процесі їхнього навчання, оновлення чи адаптації, що

дозволить своєчасно виявляти зміни в якості результатів і коригувати процеси оптимізації моделей.

Для оцінки якості моделей будуть використані стандартні метрики, які відповідатимуть особливостям кожного тесту та типу завдань. Серед базових метрик ключове місце займатиме точність (accuracy) – показник, що відображатиме відсоток коректних відповідей моделей на тестові запитання, що є одним із найважливіших критеріїв для оцінки якості моделей. Для оцінки швидкодії моделей буде використано середній час відповіді, що дозволить аналізувати не лише якість, а й ефективність роботи моделей при різних обсягах навантаження. Окрім того, у тестах, пов’язаних із вилученням та розпізнаванням інформації, будуть застосовуватися метрики F1-міри, які допомагають оцінити баланс між точністю (precision) та повнотою (recall). Для задач програмування, що передбачають генерацію коду, будуть враховуватися показники правильності згенерованого коду, а також складність і послідовність кроків у процесі генерації. Такий комплексний підхід до вибору метрик забезпечить об’єктивну оцінку, яка дозволить не тільки порівнювати їх між собою, та зробити висновки щодо можливостей і обмежень кожної з них у різних сферах застосування.

2.3 Методика створення промптів для бенчмаркінгу

Промпти будуть формуватися з урахуванням факторів, які впливають на якість і точність відповідей. Насамперед, при розробці промптів враховуватиметься тип завдання, який моделі необхідно виконати. Це означатиме, що промпти для тестів загальних знань, як MMLU, матимуть інший формат та структуру порівняно з промптами для програмування або для математичних задач. Такий підхід дозволить краще відобразити специфіку кожного тесту і створити умови, максимально наближені до реальних сценаріїв використання моделей.

У текстах промптів буде забезпечено контекст, який допоможе моделі правильно інтерпретувати завдання і уникнути неоднозначностей. Наприклад, у

тестах на загальні знання будуть чітко сформульовані питання із конкретними варіантами відповіді, що спростить аналіз результатів і зменшить ймовірність помилок через неправильне розуміння питання. Промпти типу: «Яка столиця Франції? Варіанти: а) Лондон, б) Париж, в) Берлін» або «Хто написав «Кобзар»? Варіанти: а) Тарас Шевченко, б) Леся Українка в) Іван Франко» будуть формуватися для оцінювання базових знань.

Для програмувальних завдань промпти матимуть вигляд детальних інструкцій із описом функціональності, яку необхідно реалізувати. Наприклад, для HumanEval це може бути: «Напиши функцію sum_list, яка приймає список чисел і повертає їхню суму. Функція має підтримувати як порожні списки, так і списки з від'ємними числами». Такий рівень деталізації допоможе моделі зорієнтуватися у вимогах і створити більш релевантний код. Окрім того, у промптах для програмування будуть додані приклади вхідних і вихідних даних, що покращить якість відповіді. Наприклад: «Вхід: [1, 2, 3], вихід: 6».

Математичні промпти, які застосовуватимуться у тестах MATH-500, міститимуть текстові задачі із проханням надати розв'язок або покрокове пояснення. Наприклад, промпт може звучати так: «Обчисліть суму перших десяти членів арифметичної прогресії, де перший член дорівнює 3, а різниця – 2». Для задач, які вимагають логічного обґрунтування, промпти стимулюватимуть модель до використання підходу формулювання проміжних кроків та пояснень. Такий підхід дозволить не лише отримати правильну відповідь, а й оцінити здатність моделі до логічного мислення.

Також для підвищення точності генерації відповідей в промпти включатимуться приклади коректних відповідей або рішень. Це допоможе моделі зрозуміти стиль, формат і рівень деталізації, який очікується у відповіді. Наприклад, у промпті для задачі програмування можуть бути наведені приклади функцій із коментарями, що пояснюють кожен крок коду, або у тестах на логіку – розгорнуті відповіді з покроковим викладом міркувань.

Окремо буде забезпечена стандартизація промптів для кожного типу завдань та моделей. Це дозволить уникнути перекосів при тестуванні та

забезпечить порівняння результатів. Водночас, підказки будуть адаптовані під специфіку кожної з моделей, з урахуванням їхніх сильних і слабких сторін. Наприклад, для моделей із кращою здатністю обробляти довгі тексти промпти можуть бути більш розгорнутими, тоді як для моделей із швидшою відповіддю – більш лаконічними.

Отже, методика створення промптів базуватиметься на детальному аналізі типів завдань, характеру відповідей і особливостей кожної моделі, що забезпечить отримання максимально релевантних і точних результатів під час бенчмаркінгу. Це створить основу для об’єктивного порівняння продуктивності та ефективності моделей у різних сферах застосування.

2.4 Архітектура та проєктування програмного рішення

Програмне рішення для бенчмаркінгу великих мовних моделей має модульну архітектуру, яка забезпечує гнучкість, зручність у підтримці та можливість подальшого розвитку системи. Основними компонентами системи є модуль інтеграції з API моделей GPT та Gemini, модуль управління бенчмарком та модуль обробки і аналізу результатів.

Модуль інтеграції відповідає за встановлення зв’язку з API моделей, формуванням запитів, надсилання промптів та отримання відповідей. Для взаємодії з моделями використовується платформа OpenRouter, яка виступає в ролі єдиного універсального шлюзу до різних LLM. Це дозволяє спростити реалізацію інтеграції, оскільки OpenRouter надає простий інтерфейс доступу до декількох моделей через єдиний API. Для підвищення продуктивності та зменшення часу очікування реалізована асинхронна обробка запитів, що дозволяє паралельно взаємодіяти з різними сервісами.

Модуль управління бенчмарками координує запуск тестів, таких як MMLU, DROP, HumanEval, MATH-500 тощо. Він відповідає за послідовне або паралельне виконання тестів, обробку промптів і збір результатів з подальшим логуванням помилок та винятків.

Модуль обробки і аналізу результатів виконує порівняння отриманих відповідей із еталонними, обчислює кількісні метрики, точність, продуктивність, F1-міру та генерує аналітичні звіти у зручному для користувача вигляді. Це дозволяє робити оцінку продуктивності кожної моделі за різними напрямками.

Промпти для бенчмаркінгу знаходяться безпосередньо в коді програмного рішення. Вони реалізовані у вигляді шаблонів та фіксованих текстових запитів, які використовуються мовними моделями під час тестування. Це дозволяє легко змінювати й налаштовувати промпти. При необхідності підказки можуть бути розширені або адаптовані під нові завдання без значних змін в архітектурі системи. На рис. 2.1 продемонстровано діаграму послідовності для спроєктованого рішення.

Для розробки вибрано мову Python, оскільки вона має великий набір бібліотек для роботи з API, асинхронної обробки, аналізу даних та побудови інтерфейсів користувача. Як альтернативні технологічні варіанти розглядалось використання серверної платформи Node.js з JavaScript, що може надати додаткові можливості для масштабування і інтеграції. Також можливе застосування контейнеризації за допомогою Docker для полегшення розгортання та ізоляції компонентів, а в подальшому – використання систем оркестрації, таких як Kubernetes, для автоматизації керування сервісами.

Звідси, архітектура спроєктована з урахуванням сучасних технологій і принципів розробки, що забезпечують високу продуктивність, гнучкість і можливість подальшого розвитку системи. Вибір технологій і компонентів орієнтований на максимальну ефективність та простоту інтеграції з різними мовними моделями, що робить рішення адаптивним до майбутніх змін і розширень.

2.5 Огляд програмної реалізації

Програмна реалізація системи побудована на основі модульної архітектури, що забезпечує масштабованість, гнучкість при додаванні нових

тестів та інтеграцію з різними мовними моделями. Для взаємодії з моделями використовується API-платформа OpenRouter, яка надає доступ до API моделей.

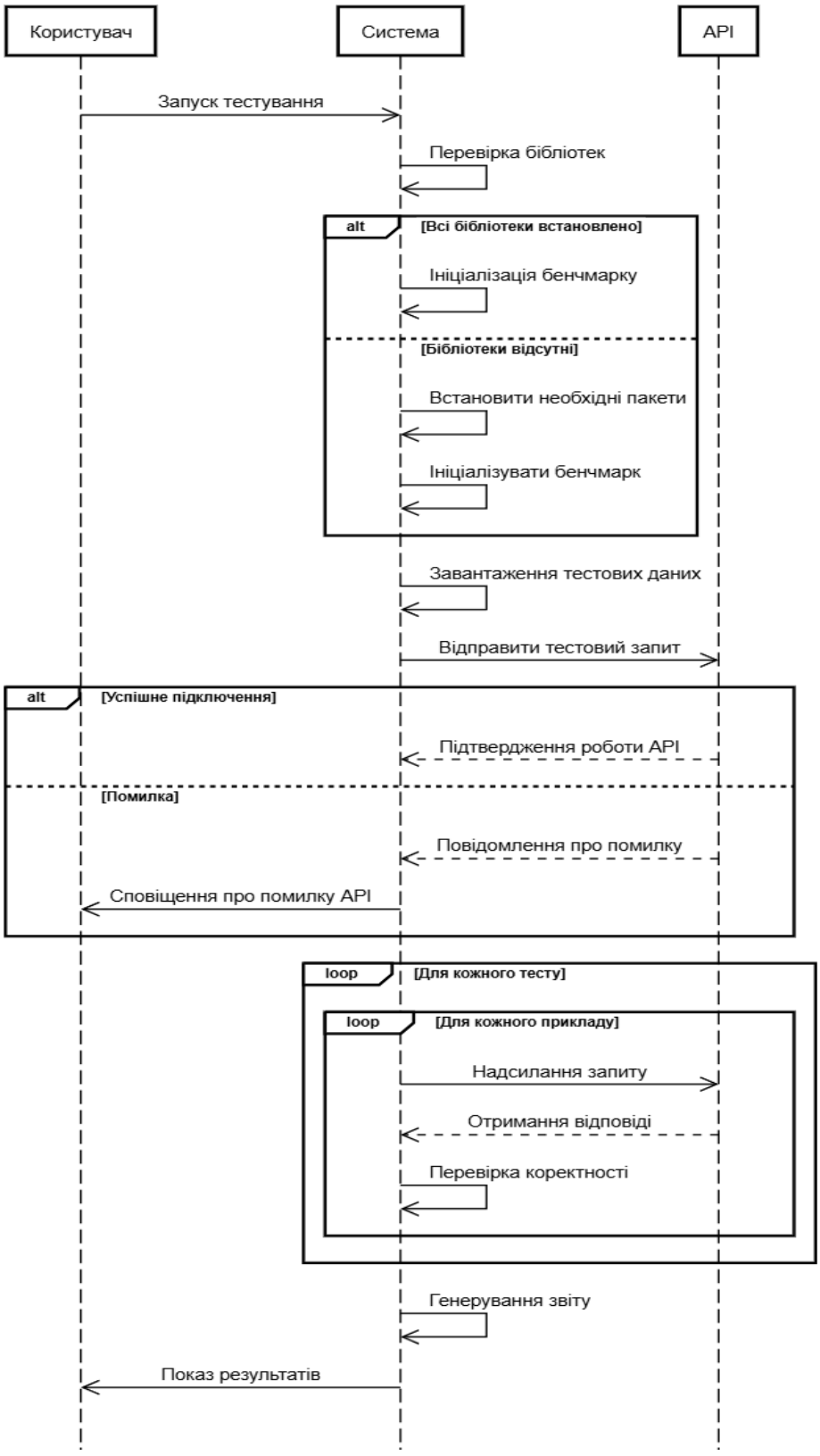


Рисунок 2.1 – Діаграма послідовності виконання програмного рішення

Основною мовою розробки є Python завдяки обширній бібліотекі для роботи з API, асинхронного програмування, обробки даних та аналітики. Для реалізації бенчмаркінгу були використані основні бібліотеки та інструменти Python, зібрані в таблиці 2.1. Ці бібліотеки забезпечують гнучкість, простоту розробки, зручну інтеграцію з API, а також дозволяють ефективно проводити аналіз отриманих результатів тестування моделей.

Таблиця 2.1 – Використані бібліотеки та їх призначення

Бібліотека	Призначення
openai	Взаємодія з OpenRouter, надсилання запитів до API.
pandas	Обробка та аналіз табличних даних, побудова узагальнених звітів за результатами бенчмарків.
json	Робота з форматом JSON для збереження детальних результатів тестувань.
subprocess та sys	Автоматична перевірка та встановлення необхідних залежностей перед запуском системи.
pkg_resources	Отримання інформації про встановлені пакети у середовищі Python для динамічної перевірки залежностей.
typing	Анотація типів для покращення читабельності коду та підтримки валідації типів під час розробки.

Зроблена автоматична перевірка встановлених бібліотек перед запуском програми, наведено в лістингу 2.1.

Лістинг 2.1 – Автоматична перевірка бібліотек

```
def check_dependencies():
    """Перевірка та встановлення залежностей"""
    required = {'requests', 'pandas', 'openai'}
    try:
        installed = {pkg.key for pkg in pkg_resources.working_set}
        missing = required - installed
        if missing:
            print(f"Встановлення відсутніх пакетів: {', '.join(missing)}")
            subprocess.check_call([sys.executable, '-m', 'pip', 'install',
*missing])
    except:
        try:
```

```
            import requests
            import pandas
            import openai
        except ImportError as e:
            print(f"Відсутня залежність: {str(e)}")
            print("Будь ласка, встановіть необхідні пакети вручну:")
            print("pip install requests pandas openai")
            sys.exit(1)
```

Ініціалізація системи бенчмаркінгу виконується через основний об’єкт класу, який відповідає за встановлення параметрів тестування, конфігурацію підключення до мовної моделі через OpenRouter API та підготовку набору метрик для оцінки, показано в лістингу 2.2.

Лістинг 2.2 – Класс GeminiBenchmark

```
class GeminiBenchmark:
    def __init__(self):
        self.client = openai.OpenAI(
            base_url="https://openrouter.ai/api/v1",
            api_key="API-Ключ"
        )
        self.model = "google/gemini-2.5-pro-preview"
        self.metrics = {
            "Робота з даними": {
                "Загальні знання": ["MMLU", "MMLU-Redux", "MMLU-Pro",
"DROP", "IF-Eval"],
                "Спеціалізовані тести": ["GPQA-Diamond", "FRAMES",
"LongBench"]
            },
            "Програмування": {
                "Генерація коду": ["HumanEval-Mul", "LiveCodeBench",
"LiveCodeBench-COT"],
                "Інженерія": ["Codeforces", "SWE-Verified", "Aider-Edit",
"Aider-Polyglot"]
            },
            "Математика": {
                "Базова": ["MATH-500"],
                "Просунута": ["AIME", "CNMO"]
            },
            "Китайська": ["CLUEWSC", "C-Eval", "C-SimpleQA"]
        }
 self.test_data = self._load_test_data()
 self.results = {}
```

У ході ініціалізації відбувається:

– встановлення параметрів доступу до API OpenRouter з використанням відповідного ключа авторизації, яке реалізовано через об’єкт

self.client – клієнт бібліотеки openai, налаштований для роботи з OpenRouter API;

– вибір конкретної мовної моделі для тестування, що задається у атрибуті self.model;

– визначення структури метрик та набору тестів, які зберігаються у словнику self.metrics для систематичного запуску бенчмарків;

– підготовка тестових даних та ініціалізація сховища для збереження результатів, що виконується у відповідних внутрішніх методах класу.

Для взаємодії з моделями використовується OpenRouter API через офіційну клієнтську бібліотеку openai, налаштовану на роботу з відповідним базовим URL. Об’єкт self.client створюється з параметрами:

- base_url="https://openrouter.ai/api/v1" – базова адреса API OpenRouter;
- api_key – персональний ключ авторизації, що забезпечує безпечний доступ до сервісу.

Запити до моделі формуються як чат-комунікація за допомогою методу chat.completions.create(), для якого передається назва моделі (self.model); список повідомлень у форматі { "role": "user", "content": prompt }, де prompt – текст запиту; максимальна кількість токенів у відповіді (max_tokens); параметр температури (temperature), що контролює креативність відповіді. Після відправлення запиту метод повертає відповідь моделі, яку далі обробляє система для оцінювання якості, наведено в лістингу 2.3.

Оцінка кожного тесту здійснюється методом _evaluate_metric(). Для кожного тестового прикладу система виконує такі кроки:

- генерує текстовий запит (промпт) на основі вхідних даних;
- надсилає запит до великої мовної моделі через API;
- вимірює час, необхідний для отримання відповіді;
- перевіряє коректність відповіді відповідно до критеріїв конкретної метрики;
- накопичує кількість правильних відповідей для подальшого розрахунку точності.

Лістинг 2.3 – Ключовий метод для надсилання запитів до LLM

```
def _query_gemini(self, prompt: str, max_tokens: int = 500) -> str:
        """Запит до Gemini через OpenRouter"""
        try:
            completion = self.client.chat.completions.create(
                model=self.model,
                messages=[
                    {"role": "user", "content": prompt}
                ],
                max_tokens=max_tokens,
                temperature=0.7
            )
            return completion.choices[0].message.content
        except Exception as e:
            print(f"Помилка API: {str(e)}")
            return None
```

Роботу метода _evaluate_metric(), наведено в лістингу 2.4

Лістинг 2.4 – Робота методу _evaluate_metric()

```
    def _evaluate_metric(self, metric: str, num_samples: int):
        if metric not in self.test_data or not self.test_data[metric]:
            print(f"⚠ Немає тестових даних для {metric}")
            return
        samples = self.test_data[metric][:num_samples]
        correct = 0
        detailed_results = []
        print(f"\nТестування {metric} ({len(samples)} прикладів)...")
        for i, test_case in enumerate(samples):
            try:
                question = test_case[0]
                print(f"\nПриклад {i+1}: {question[:60]}...")
                start_time = time.time()
                response = self._query_gemini(question, max_tokens=500)
                elapsed = time.time() - start_time
                if response is None:
                    raise Exception("Помилка запиту до API")
                is_correct = self._check_answer(metric, test_case,
response)
                correct += 1 if is_correct else 0
                detailed_results.append({
                    "питання": question,
                    "відповідь": response,
                    "правильно": is_correct,
                    "час": elapsed
                })
                print(f"  {'if is_correct else '} Час: {elapsed:.2f}с")
                print(f"  Відповідь: {response[:100]}..." if response else
"  Немає відповіді")
            except Exception as e:
                print(f"  Помилка: {str(e)}")
                detailed_results.append({
                    "питання": question,
                    "відповідь": f"Помилка: {str(e)}",
                    "правильно": False,
                    "час": 0
                })
        accuracy = correct / len(samples) if samples else 0
```

```
            avg_time = sum(r['час'] for r in detailed_results) / len(samples)
if samples else 0
            self.results[metric] = {
                "точність": accuracy,
                "середній_час": avg_time,
                "тестовані_приклади": len(samples),
                "детальні_результати": detailed_results
            }
```

У цьому фрагменті:

– self.test_data – словник із тестовими прикладами для кожної метрики;

– _query_gemini(question) – функція надсилання запиту до моделі і отримання відповіді;

– _check_answer(metric, test_case, response) – метод, що визначає правильність відповіді;

– self.results – словник для збереження результатів тестування з ключами за назвами метрик.

Для базової перевірки коректності відповіді мовної моделі в системі використовується простий демонстраційний механізм. Він передбачає перевірку наявності тексту у відповіді, тобто якщо відповідь порожня або складається лише з пропусків, вона вважається некоректною. В іншому випадку відповідь вважається прийнятною без додаткового аналізу.

Цей підхід є спрощеним і застосовується для початкової валідації роботи моделі, без детального зіставлення з еталонною відповіддю. Він корисний на етапі розробки, коли потрібно відфільтрувати явно порожні або некоректні результати.

Запуск процесу тестування здійснюється у блоці if __name__ == "__main__":, що забезпечує запуск лише при безпосередньому виконанні скрипта. Роботу наведено в лістингу 2.5.

На початку виконується функція check_dependencies(), яка перевіряє наявність усіх необхідних бібліотек і залежностей для коректної роботи системи. Далі створюється об'єкт класу GeminiBenchmark, який відповідає за проведення бенчмарку. Метод run_benchmark(num_samples=5) запускає тестування, обробляючи по 5 прикладів для кожної заданої метрики.

Лістинг 2.5 – Запуск тестування

```
if __name__ == "__main__":
    check_dependencies()
    try:
        print("Ініціалізація тестування Gemini 2.5 Pro Preview...")
        benchmark = GeminiBenchmark()
        benchmark.run_benchmark(num_samples=5)
        print("\nРезультати тестування:")
        report = benchmark.generate_report()
        print(report.to_markdown(tablefmt="grid", stralign="left"))
        benchmark.save_results()
    except Exception as e:
        print(f"Критична помилка: {str(e)}")
```

На рис. 2.2 продемонстровано запуск тестування моделі Gemini. Також на рис. 2.3 показано повідомлення про помилку підключення до API.

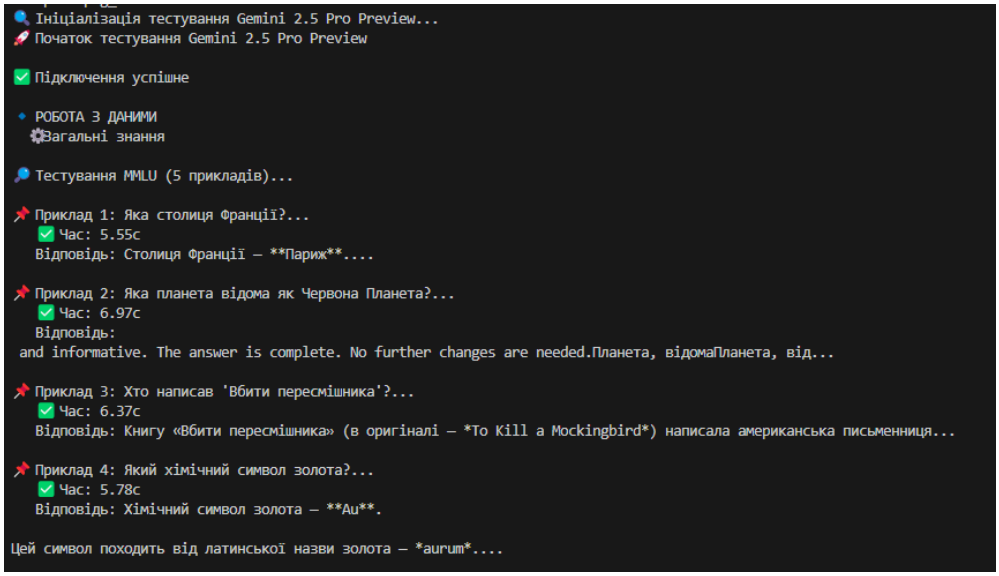


Рисунок 2.2 – Запуск тестування Gemini 2.5 Pro у VS Code з виводом у консолі

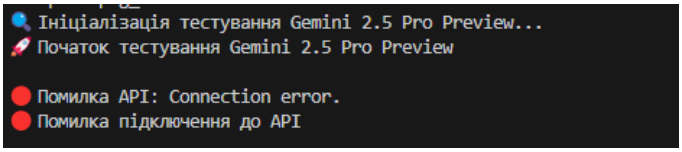
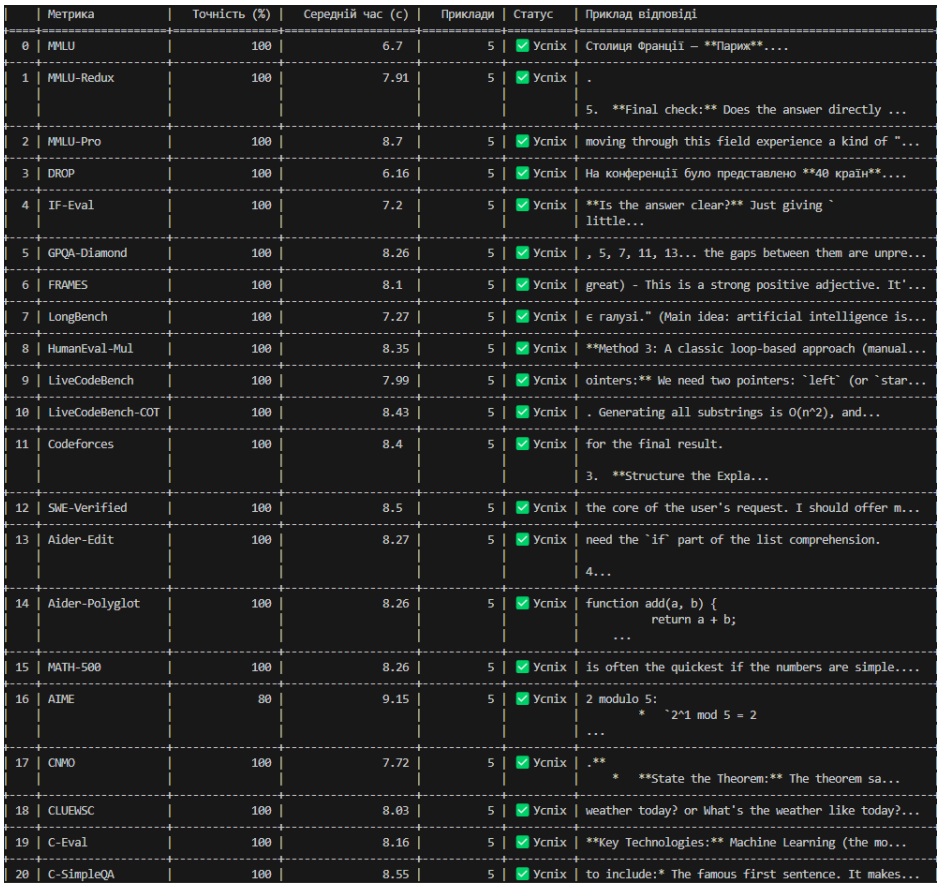


Рисунок 2.3 – Повідомлення про помилку підключення API

Після завершення тестування викликається метод generate_report(), який формує структурований звіт з результатами. Звіт виводиться у вигляді таблиці з форматуванням grid за допомогою бібліотеки tabulate, що забезпечує зручне та

читабельне відображення. На рис 2.4 показано сформований звіт з результатами тестування.



	Метрика	Точність (%)	Середній час (с)	Приклади	Статус	Приклад відповіді
0	MMLU	100	6.7	5	✅ Успіх	Столиця Франції — **Париж**....
1	MMLU-Redux	100	7.91	5	✅ Успіх	. 5. **Final check:** Does the answer directly ...
2	MMLU-Pro	100	8.7	5	✅ Успіх	moving through this field experience a kind of "...
3	DROP	100	6.16	5	✅ Успіх	На конференції було представлено **40 країн**....
4	IF-Eval	100	7.2	5	✅ Успіх	**Is the answer clear?** Just giving ` little...
5	GPQA-Diamond	100	8.26	5	✅ Успіх	, 5, 7, 11, 13... the gaps between them are unpre...
6	FRAMES	100	8.1	5	✅ Успіх	great) - This is a strong positive adjective. It'...
7	LongBench	100	7.27	5	✅ Успіх	є галузі." (Main idea: artificial intelligence is...
8	HumanEval-Mul	100	8.35	5	✅ Успіх	**Method 3: A classic loop-based approach (manual...
9	LiveCodeBench	100	7.99	5	✅ Успіх	ointers:** We need two pointers: `left` (or `star...
10	LiveCodeBench-COT	100	8.43	5	✅ Успіх	. Generating all substrings is O(n^2), and...
11	Codeforces	100	8.4	5	✅ Успіх	for the final result. 3. **Structure the Expla...
12	SWE-Verified	100	8.5	5	✅ Успіх	the core of the user's request. I should offer m...
13	Aider-Edit	100	8.27	5	✅ Успіх	need the `if` part of the list comprehension. 4...
14	Aider-Polyglot	100	8.26	5	✅ Успіх	function add(a, b) { return a + b; ...
15	MATH-500	100	8.26	5	✅ Успіх	is often the quickest if the numbers are simple....
16	AIME	80	9.15	5	✅ Успіх	2 modulo 5: * `2^1 mod 5 = 2 ...
17	CNMO	100	7.72	5	✅ Успіх	.** * **State the Theorem:** The theorem sa...
18	CLUEWSC	100	8.03	5	✅ Успіх	weather today? or What's the weather like today?...
19	C-Eval	100	8.16	5	✅ Успіх	**Key Technologies:** Machine Learning (the mo...
20	C-SimpleQA	100	8.55	5	✅ Успіх	to include:* The famous first sentence. It makes...

Рисунок 2.4 – Сформований звіт з результатами тестування

Результати тестування зберігаються у файл за допомогою методу save_results(), що дозволяє подальший аналіз та документування. Вся процедура обгорнута у блок try-except, який дозволяє коректно обробити та повідомити про критичні помилки, що можуть виникнути під час роботи. На рис 2.5 продемонстровано повідомлення про згенерований звіт у файлах формату JSON та CSV.

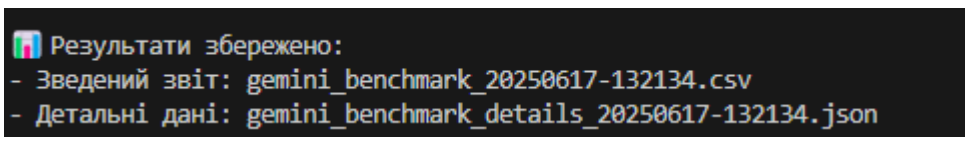


Рисунок 2.5 – Повідомлення про згенерований звіт у файлах

Такий підхід забезпечує контрольований, надійний та зручний запуск процесу оцінювання великих мовних моделей із автоматизованим збором результатів. Генерація фінального звіту здійснюється методом generate_report(), який агрегує отримані результати тестування по всіх метриках у структурованому форматі. Для представлення звіту використовується бібліотека pandas, що дозволяє формувати таблиці з метриками точності, часом відповіді та іншими параметрами. Код генерації звіту наведено в лістингу 2.7.

Звіт можна виводити на екран у табличному форматі для швидкого перегляду або зберігати у зовнішні файли для подальшого аналізу. Для збереження передбачена функція save_results(), наведено в лістингу 2.8, яка формує два файли: CSV – зведений звіт для зручного відкриття в табличних редакторах, і JSON – детальні дані для глибшого дослідження.

Лістинг 2.7 – Генерація звіту тестування

```
    def generate_report(self) -> pd.DataFrame:
        """Генерація звіту"""
        report_data = []
        for metric, result in self.results.items():
            report_data.append({
                "Метрика": metric,
                "Точність (%)": f"{result['точність']*100:.1f}",
                "Середній час (с)": f"{result['середній_час']:.2f}",
                "Приклади": result['тестовані_приклади'],
                "Статус": "Успіх" if result['точність'] >= 0.7 else "⚠ Частково" if result['точність'] >= 0.4 else "Невдача",
    "Приклад відповіді": result['детальні_результати'][0]['відповідь'][:50] + "..." if result['детальні_результати'] else ""
            })
        return pd.DataFrame(report_data)
```

Лістинг 2.8 – Генерація звіту тестування

```
    def save_results(self):
        """Збереження результатів у файли"""
        timestamp = time.strftime("%Y%m%d-%H%M%S")
        report = self.generate_report()
        csv_file = f"gemini_benchmark_{timestamp}.csv"
        report.to_csv(csv_file, index=False)
        json_file = f"gemini_benchmark_details_{timestamp}.json"
        with open(json_file, 'w', encoding='utf-8') as f:
            json.dump(self.results, f, ensure_ascii=False, indent=2)
        print(f"\nРезультати збережено:")
        print(f"- Зведений звіт: {csv_file}")
        print(f"- Детальні дані: {json_file}")
```

Такий підхід сприяє більш точної оцінки моделей та дозволяє швидко порівняти продуктивність різних LLM, а також створює умови для автоматизованого формування документів з результатами досліджень. Загалом, у другому розділі кваліфікаційної роботи було виконано проєктування та програмну реалізацію прототипу власної системи бенчмаркінгу великих мовних моделей на основі популярних спеціалізованих метрик. Сформований програмний проєкт готовий до тестування.

РОЗДІЛ 3
АНАЛІЗ РЕЗУЛЬТАТІВ БЕНЧМАРКІНГУ

3.1 Вибір методів тестування програмного рішення

Для гарантії стабільної роботи системи бенчмаркінгу на різних етапах і рівнях розробки застосовувалися автоматизовані методи тестування. Тестування охоплювало перевірку окремих модулів, взаємодії між компонентами системи, а також функціонування API в різних умовах. Використаний підхід базується на типовій класифікації, що включає: (1) модульне, (2) інтеграційне, (3) функціональне та (4) тестування API. Відповідність між цими видами тестування та застосованими інструментами наведена в таблиці 3.1.

Таблиця 3.1 – Використані інструменти для різних типів тестування

Рівень тестування	Інструменти
Модульне тестування	pytest
Інтеграційне тестування	unittest
Функціональне тестування та тестування API	requests, pytest

Модульне тестування реалізовано за допомогою фреймворку pytest, який є одним із найпопулярніших у Python-спільноті завдяки простоті використання, гнучкості та широкому набору плагінів. Pytest дозволяє легко писати як прості, так і складні тести, підтримує параметризацію тестів і паралельний запуск, що значно економить час у великих проєктах. Завдяки ізоляції модулів можна перевіряти окремі функції та класи без залучення зовнішніх залежностей, що робить тести швидкими та надійними. [18]

Інтеграційне тестування реалізоване за допомогою фреймворку unittest з використанням Mock-об’єктів та тестових подвійників (test doubles). Це дозволяє емуляцію взаємодії між компонентами системи без залучення реальної бази даних або зовнішніх сервісів. Такий підхід забезпечує можливість перевіряти коректність взаємодії між модулями в умовах, наближених до робочих, водночас знижуючи складність налаштувань та скорочуючи час виконання тестів. [19]

Функціональне тестування та тестування API здійснюється за допомогою бібліотеки requests, яка використовується для написання автоматизованих сценаріїв тестування API у коді, що дає змогу перевіряти правильність роботи окремих ендпоінтів у різних умовах, включно з авторизацією та обробкою помилок [20].

3.2 Формування тестового плану

Тестовий план охоплює обсяги, методи, інструменти та загальну логіку перевірки, які забезпечують коректну та стабільну роботу системи. Враховуючи, що застосунок має багаторівневу архітектуру на базі мови Python з інтеграцією через OpenRouter, тестування було зосереджено на перевірці коректності взаємодії із зовнішніми API та внутрішньої логіки обробки даних.

Передусім, тестуванню підлягали модулі, які відповідали за ініціалізацію бенчмарку (функція check_dependencies() та метод run_benchmark класу GeminiBenchmark), завантаження тестових даних (метод _load_test_data), формування та надсилання запитів до API (конструктор класу GeminiBenchmark, де ініціалізується self.client), обробку отриманих відповідей (метод generate_report() та вивід результатів), а також формування та збереження звітів (метод save_results, який зберігає дані у форматах CSV та JSON).

Тестування виконувалося поетапно, що відображено у таблиці 3.2.

Таблиця 3.2 – План тестування за різними етапами

Етап тестування	Опис робіт	Очікуваний результат
Модульне тестування	Перевірка окремих компонентів або функцій системи	Кожен модуль працює коректно і видає правильні дані
Інтеграційне тестування	Тестування взаємодії між модулями, перевірка їх співпраці	Модулі коректно взаємодіють і передають дані без помилок
Функціональне тестування	Перевірка коректності роботи API: запити, відповіді, обробка помилок	API працює відповідно до специфікації, повертає коректні відповіді

Основними приймальними критеріями були такі: всі ключові функції мають виконуватися без помилок; під час навантаження система має

забезпечувати швидку обробку: 85% запитів повинні оброблятися за час, що не перевищує 5 секунд; коректна обробка негативних сценаріїв без збоїв та аварійного завершення. За допомогою pytest проведено модульне тестування основних компонентів системи: ініціалізації, завантаження даних, взаємодії з API, обробки результатів та збереження звітів. Особливу увагу приділено перевірці роботи з коректними та помилковими вхідними даними. Тестові сценарії наведено в таблиці 3.3.

Таблиця 3.3 – Тестові сценарії для модульних тесту

№	Назва тесту (pytest)	Мета тестування	Вхідні дані	Очікуваний результат	Короткий опис процедури тестування
1	test_initialization	Перевірити коректне створення об’єкта та ініціалізацію API клієнта	Виклик конструктора класу без параметрів	Об’єкт створено, налаштований клієнт OpenAI з ключем	Створити об’єкт класу, перевірити, що атрибути не порожні
2	test_query_gemini	Перевірити, що метод _query_gemini успішно робить запит до API	Виклик _query_gemini з простим текстом ("Скажи 'тест'")	Отримано не пусту відповідь без помилок	Викликати метод, переконатися, що є текстова відповідь
3	test_run_benchmark_empty	Перевірити поведінку при запуску тестів, якщо test_data порожні	Виклик run_benchmark з порожнім test_data	Вивід попереджень про відсутність тестових даних	Ініціалізувати об’єкт з пустим test_data, викликати запуск
4	test_evaluate_metric	Перевірити коректність підрахунку точності і часу	Метод _evaluate_metric з тестовими даними	Коректний підрахунок, збереження результатів	Викликати _evaluate_metric з тестами, перевірити results
5	test_generate_report	Перевірити формат і зміст зведеного звіту	Виклик generate_report після виконання тестів	DataFrame із ключовими метриками, коректними статусами	Виконати бенчмарк, викликати генерацію звіту, перевірити формат

Продовження таблиці 3.3

6	test_save_results	Перевірити, що файли CSV та JSON створені коректно	Виклик save_results після запуску бенчмарку	Файли створені, вивід шляхів до файлів	Запустити тест, викликати збереження, перевірити наявність файлів
7	test_api_error_handling	Перевірити поведінку при помилці запиту до API	Імітація виклику _query_gemini, який повертає None	Логування помилки, продовження роботи без аварії	Змоделювати помилку в _query_gemini, переконатися у логуванні
8	test_check_answer	Перевірити, що відповіді без тексту вважаються неправильними	Виклик _check_answer з пустою або None відповіддю	Повернення False	Викликати метод з порожнім рядком, перевірити результат

У лістингу 3.1 показано приклад написання тесту зі застосуванням фреймворку pytest.

Лістинг 3.1 – Приклад написання тесту

```
def test_initialization():
    benchmark = GeminiBenchmark()
    assert benchmark is not None,
    assert benchmark.client is not None,
    assert isinstance(benchmark.metrics, dict),
    assert isinstance(benchmark.test_data, dict),
```

Інтеграційне тестування в цьому проєкті було спрямоване на перевірку коректної взаємодії між основними компонентами системи, зокрема, правильності формування та надсилання запитів до моделі Gemini через OpenRouter API, коректного оброблення відповідей, а також збереження та обробки результатів тестування. Для забезпечення максимально реалістичного середовища тестування, при цьому без залучення зовнішніх ресурсів, у тестах застосовувався підхід з використанням Mock-об’єктів та тестових дублерів (test doubles), що дозволяв емуляцію поведінки API та взаємодії компонентів. Такий підхід знижував залежність від зовнішніх факторів, підвищував надійність тестування та дозволяв виявляти помилки у взаємодії модулів на ранніх стадіях розробки.

Під час функціонального тестування OpenRouter, який надавав доступ до LLM через API, застосовувалися бібліотека requests та тестовий фреймворк pytest. Було створено набір тестів, які імітували взаємодію з моделями через API, перевіряли правильність формування запитів, отримання та обробку відповідей, а також поведінку системи у випадках некоректних або неповних даних. Тестування охоплювало як основні робочі сценарії, так і ситуації з помилковою автентифікацією API. Відповідні тестові випадки описано в таблиці 3.3.

Таблиця 3.3 – Тестові випадки для функціонального тестування OpenRouter

№	Назва тесту (pytest)	Мета тестування	Вхідні дані	Очікуваний результат	Короткий опис процедури тестування
1	test_valid_request	Перевірити коректну роботу запиту до OpenRouter	Коректний запит із валідними даними	Отримання очікуваної відповіді від моделі	Викликати API із правильними параметрами, перевірити відповідь
2	test_authentication_error	Перевірити реакцію системи на некоректний токен авторизації	Запит із неправильним або відсутнім токеном авторизації	Отримання помилки автентифікації	Виконати запит без або з неправильним токеном, перевірити помилку
3	test_invalid_input_data	Перевірити поведінку системи при неправильних вхідних даних	Запит із пошкодженими або неповними параметрами	Коректне повідомлення про помилку або відхилення запиту	Надіслати запит з некоректними даними, перевірити реакцію системи
4	test_exception_handling	Перевірити стійкість системи при несподіваних помилках	Імітація виняткових ситуацій при виклику API	Логування помилки, без аварійного завершення	Змоделювати виключення в API, перевірити коректність обробки помилки

У лістингу 3.2 наведено приклад написання тесту зі застовування фрейморку pytest та бібліотеки requests.

Лістинг 3.2 – Приклад написання тесту API

```
def test_valid_request(headers):
    payload = {
        "model": "google/gemini-2.5-pro-preview",
        "prompt": "Привіт, OpenRouter!",
        "max_tokens": 50
    }
    response = requests.post(f"{BASE_URL}/completions", json=payload,
headers=headers)
    assert response.status_code == 200
    data = response.json()
    assert "choices" in data
    assert isinstance(data["choices"], list)
    assert len(data["choices"]) > 0
    assert "text" in data["choices"][0]
    assert len(data["choices"][0]["text"]) > 0
```

За допомогою інструментів pytest і requests було проведено тестування API, що дозволило перевірити коректність роботи, обробку помилок та стабільність системи під різними умовами навантаження. Це сприяло виявленню потенційних проблем і підвищенню якості сервісу.

3.3 Порівняльний аналіз результатів тестування LLM

З метою виявлення сильних і слабких сторін сучасних великих мовних моделей було проведено порівняльне тестування трьох провідних систем – Gemini, GPT-4 та GPT-4o – за критеріями точності відповідей та продуктивності (часу генерації відповіді). Оцінювання здійснювалося на основі кількох типових бенчмарків: MMLU, DROP, HumanEval і т. п.

У контексті точності відповідей у всіх протестованих наборах даних модель Gemini 2.5 Turbo продемонструвала виняткову стабільність, досягнувши 100% точності в кожному з тестів. Це свідчить про високу здатність моделі генерувати відповіді, що є водночас фактологічно коректними та логічно послідовними.

Натомість модель GPT-4 виявила суттєву варіативність у результатах. Хоча вона впевнено впоралася з базовими завданнями DROP та LongBench, її ефективність у складніших тестах, зокрема MMLU-Pro та GPQA, була або недостатньою, або повністю відсутньою. Це вказує на обмеження моделі в

контексті генерації складних абстрактних міркувань.

GPT-4o, оновлена модифікація GPT-4, продемонструвала вищу точність у простих і середньоскладних завданнях, проте частково поступилася Gemini у запитаннях, що потребують глибокого аналізу MMLU-Pro, GPQA-Diamond. Відповідно, результати GPT-4o можна розглядати як компроміс між швидкістю та точністю.

Щодо часу генерації відповіді, найкращі результати продемонструвала модель GPT-4o, стабільно забезпечуючи час відповіді в межах 1–2 секунд у більшості тестів. Це робить її придатною для використання в інтерактивних системах у реальному часі.

GPT-4 виявилася найповільнішою з моделей: середній час відповіді у складних тестах подекуди перевищував 20 секунд. У завданнях, де критично важлива оперативність, такі затримки можуть бути небажаними.

Попри поступ у швидкості обробки порівняно з GPT-4o, модель Gemini демонструє значно кращі результати, ніж GPT-4. Середній час відповіді в межах 6–7 секунд є прийнятним для більшості офлайн-сценаріїв аналітичного характеру.

Таблиця 3.4 – Порівняльні результати тестування LLM за точністю та продуктивністю

Тест	Показник	Gemini	GPT-4	GPT-4o
MMLU	Точність	100%	80%	100%
	Час (сек.)	6.30	2.61	1.46
MMLU-Redux	Точність	100%	40%	80%
	Час (сек.)	6.71	2.23	2.39
MMLU-Pro	Точність	100%	0%	40%
	Час (сек.)	7.01	13.80	4.71
DROP	Точність	100%	100%	100%
	Час (сек.)	5.34	4.29	1.86
IF-Eval	Точність	100%	60%	80%
	Час (сек.)	6.82	5.98	1.01
GPQA-Diamond	Точність	100%	0%	0%
	Час (сек.)	7.36	23.84	7.27
FRAMES	Точність	100%	60%	40%
	Час (сек.)	6.87	2.39	1.83
LongBench	Точність	100%	80%	60%
	Час (сек.)	6.03	2.01	4.27

Отже, в третьому розділі було здійснено повний цикл тестування розробленої системи бенчмаркінгу великих мовних моделей. На етапі вибору методів тестування визначено найбільш доцільні метрики для оцінки точності, продуктивності та стабільності моделей, а також сформовано критерії для аналізу результатів.

У процесі формування тестового плану були розроблені сценарії використання та підібрані відповідні бенчмарки, що охоплюють різні типи завдань – від загальних знань до генерації коду і математичних обчислень. Це забезпечило всебічну перевірку можливостей моделей у різних умовах.

На завершальному етапі було виконано порівняльний аналіз результатів. Дослідження показало, що моделі GPT-4o та Gemini 2.5 демонструють найвищу точність і продуктивність у більшості тестів, тоді як GPT-4 поступається їм за швидкодією, хоча забезпечує стабільно високу якість відповідей. Проведений аналіз дозволив виявити сильні та слабкі сторони кожної з моделей, а також підтвердити ефективність розробленої системи бенчмаркінгу для подальшого використання в практичних і наукових проєктах.

ВИСНОВКИ

У результаті виконання всіх етапів кваліфікаційної роботи було проведено огляд сучасних великих мовних моделей. Проаналізовано їхні архітектурні особливості, методи навчання та сфери застосування. Визначено основні підходи до оцінювання якості мовних моделей та сформовано перелік актуальних метрик і методів тестування.

Також здійснено вибір великих мовних моделей для тестування, серед яких GPT-4, GPT-4o та Gemini 2.5, що є актуальними й широко використовуваними рішеннями у сфері штучного інтелекту. Їхні характеристики та можливості було проаналізовано для подальшого включення до системи бенчмаркінгу.

Було розроблено прототип системи бенчмаркінгу, який інтегрується з API та підтримує виконання тестів різних категорій, від завдань на загальні знання до програмування та математичних обчислень. Проведено повний цикл тестування обраних моделей GPT-4, GPT-4o та Gemini 2.5 для об’єктивного порівняння їх продуктивності, точності та швидкодії.

У ході дослідження встановлено, що моделі GPT-4o та Gemini 2.5 продемонстрували найкращі результати у більшості тестів, забезпечуючи високу точність і швидкість обробки запитів. Хоча продуктивність GPT-4 була дещо нижчою, модель вирізнялася стабільною якістю відповідей.

Розроблена система бенчмаркінгу підтвердила свою ефективність для комплексного порівняння великих мовних моделей і може бути використана для подальших досліджень та адаптації в освітніх і наукових проєктах, де необхідна об’єктивна оцінка можливостей сучасних мовних моделей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Msakinn M. How to Choose the Right LLM Tools, Benchmarks, and Pro Tips. URL: https://medium.com/@meryemmsakinn/how-to-choose-the-right-llm-tools-benchmarks-and-pro-tips-47d8223770f2 (дата звернення: 01.06.2025).
2. DeepSeek R1 Model Architecture. URL: https://pub.towardsai.net/deepseek-r1-model-architecture-853fefac7050 (дата звернення: 01.06.2025).
3. Moguloju S. ChatGPT vs Gemini AI Pro vs LLaMA vs Copilot vs DeepSeek R1. URL: https://medium.com/@saimoguloju2/chatgpt-vs-gemini-ai-pro-vs-llama-vs-copilot-vs-deepseek-r1-9ce268b3492d (дата звернення: 01.06.2025).
4. Microsoft Copilot vs ChatGPT vs Claude vs Gemini vs DeepSeek: Full Guide Report Comparison. URL: https://www.datastudios.org/post/microsoft-copilot-vs-chatgpt-vs-claude-vs-gemini-vs-deepseek-full-guide-report-comparison-of-cor (дата звернення: 01.06.2025).
5. Pandey S. Understanding Attention is All You Need // Medium. URL: https://medium.com/@santoshpandey987/understanding-attention-is-all-you-need-750713a1631b (дата звернення: 01.06.2025).
6. IBM. What is a Transformer model? // IBM Think. URL: https://www.ibm.com/think/topics/transformer-model (дата звернення: 01.06.2025).
7. DataCamp. How Transformers Work. URL: https://www.datacamp.com/tutorial/how-transformers-work (дата звернення: 01.06.2025).
8. Sciencely98. Unraveling the Power of Attention: A Dive into Transformer Architecture. URL: https://medium.com/@sciencely98/unraveling-the-power-of-attention-a-dive-into-transformer-architecture-eb6594ed77a3 (дата звернення: 01.06.2025).
9. What is Google Gemini? URL: https://www.ibm.com/think/topics/google-gemini (дата звернення: 01.06.2025).

10. Google Gemini 25 Pro Explained: Everything You Need to Know. URL: https://www.techtarget.com/whatis/feature/Google-Gemini-25-Pro-explained-Everything-you-need-to-know (дата звернення: 01.06.2025).
11. Discover How Gemini Works. URL: https://cloud.google.com/gemini/docs/discover/works (дата звернення: 01.06.2025).
12. Joy N. Building an LLM-Powered Application with Google's Gemini API: A Step-by-Step Guide. URL: https://medium.com/@nikhitha.joy.official/building-an-llm-powered-application-with-googles-gemini-api-a-step-by-step-guide-915a0e9d1088 (дата звернення: 01.06.2025).
13. Gemini Pro API Tutorial. URL: https://www.datacamp.com/tutorial/gemini-pro-api-tutorial (дата звернення: 01.06.2025).
14. LLM Evaluation Framework: Steps and Components. URL: https://www.deepchecks.com/llm-evaluation-framework-steps-components/ (дата звернення: 01.06.2025).
15. What are LLM Benchmarks? URL: https://www.ibm.com/think/topics/llm-benchmarks (дата звернення: 01.06.2025).
16. ORQ AI. LLM Benchmarks // ORQ AI Blog. URL: https://orq.ai/blog/llm-benchmarks (дата звернення: 01.06.2025).
17. Тести для LLMs. URL: https://www.unite.ai/uk/тести-для-llms/ (дата звернення: 01.06.2025).
18. Pytest Guide. URL: https://betterstack.com/community/guides/testing/pytest-guide/ (дата звернення: 01.06.2025).
19. Scaling Python: Unittest and Mock. URL: https://betterstack.com/community/guides/scaling-python/python-unittest-mock/ (дата звернення: 01.06.2025).
20. NashTech Global. API Testing with Pytest and Python Requests: A Beginner's Guide // NashTech Global Blog. URL: https://blog.nashtechglobal.com/api-testing-with-pytest-and-python-requests-a-beginners-guide/?utm_source=chatgpt.com (дата звернення: 01.06.2025).

ДОДАТКИ

Додаток А – Посилання на репозиторій

Репозиторій з програмною реалізацією, виконаною в межах кваліфікаційної роботи, розташовано за адресою https://github.com/DEVILOCHEEK/DIPLOM-BENCH