

Національний лісотехнічний університет України

(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук

(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)

(рівень вищої освіти)

на тему: Використання методів NLP для розширеного пошуку та аналізу текстових даних у спортивних інформаційних системах

Виконала: студентка 6 курсу групи КН-61
спеціальності

122 "Комп'ютерні науки"

(шифр і назва напрямку підготовки, спеціальності)

Бабій Х. М.

(прізвище та ініціали)

Керівник

Процик Ю. С.

(прізвище та ініціали)

Рецензент

Флуд А. О.

(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій

Кафедра комп'ютерних наук

Рівень вищої освіти другий (магістерський)

Спеціальність 122 "Комп'ютерні науки"

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри КН

 Борецька І. Б.
"10" грудня 20 25 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бабій Христині Михайлівній

(прізвище, ім'я, по батькові)

1. Тема роботи Використання методів NLP для розширеного пошуку та аналізу текстових даних у спортивних інформаційних системах

керівник роботи Процик Юрій Степанович, к.ф.-м.н.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "29" квітня 2025 року № С-288

2. Термін подання студентом роботи 10 грудня 2025 р.

3. Вихідні дані до роботи Аналіз шляхів вирішення задачі, організаційна структура системи

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Вступ. Розділ 1. Стан проблемної області.

Розділ 2. Інформаційне забезпечення.

Розділ 3. Математичне забезпечення.

Розділ 4. Програмне забезпечення.

Розділ 5. Розроблення стартап-проєкту.

Висновки. Список використаних джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
слайди для доповіді (підготовка матеріалу для доповіді загальним обсягом 10-12 слайдів)


6. Дата видачі завдання 1 травня 2025 р.

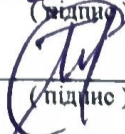
КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Ознайомлення з підходами використання методів NLP у пошукових системах та спортивних базах даних. Дослідження предметної області	01.05.2025 р. 25.05.2025 р.	<i>Виконано</i>
2.	Розроблення керуючої інструкції для LLM, що класифікує запити користувачів	26.05.2025 р. 08.07.2025 р.	<i>Виконано</i>
3.	Реалізація механізму генерації SQL-запитів на основі проаналізованих запитів користувача	09.07.2025 р. 19.08.2025 р.	<i>Виконано</i>
4.	Забезпечення інтеграції з хмарними платформами LLM	20.08.2025 р. 17.09.2025 р.	<i>Виконано</i>
5.	Розроблення користувацького інтерфейсу для прямої взаємодії з мовною моделлю	18.09.2025 р. 20.10.2025 р.	<i>Виконано</i>
6.	Тестування роботи створеної інтелектуальної системи	21.10.2025 р. 15.11.2025 р.	<i>Виконано</i>
7.	Оформлення пояснювальної записки та задача на рецензування	16.11.2025 р. 10.12.2025 р.	<i>Виконано</i>

Студент

Керівник роботи


(підпис)


(підпис)

Бабій Х. М.
(прізвище та ініціали)

Процик Ю. С.
(прізвище та ініціали)

АНОТАЦІЯ

Магістерська робота містить 63 сторінки пояснювальної записки, 18 рисунків, 14 таблиць, 1 додаток, 19 джерел.

Дана робота присвячена розробці інтелектуальної системи з фігурного катання, що використовує технології обробки природної мови (NLP) та великі мовні моделі (LLMs) для автоматичної генерації SQL-запитів на основі питань українською мовою. Проєкт створено з використанням інтерфейсу програмування додатків Windows Forms та мови програмування C# з інтеграцією хмарних API штучного інтелекту та візуалізацією даних через бібліотеку OxyPlot. Проведено порівняльний аналіз шести мовних моделей за метриками точності та ефективності виконання запитів.

Ключові слова:

Методи NLP, задача Text-to-SQL, LLM, C#, Windows Forms, фігурне катання.

ABSTRACT

The thesis contains 63 pages of explanatory note, 18 figures, 14 tables, 1 appendix, 19 sources.

This work is dedicated to the development of an intelligent figure skating system that utilizes Natural Language Processing (NLP) technologies and Large Language Models (LLMs) for automatic SQL query generation based on questions in the Ukrainian language. The project was created using the Windows Forms Application Programming Interface and the C# programming language, with integration of cloud-based artificial intelligence APIs and data visualization through the OxyPlot library. A comparative analysis of six language models was conducted based on accuracy and query execution efficiency metrics.

Keywords:

NLP methods, Text-to-SQL task, LLM, C#, Windows Forms, figure skating.

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно розширити функціональні можливості інформаційно-пошукової системи “Figure Skating” шляхом інтеграції сучасних методів обробки природної мови (NLP) та великих мовних моделей (LLM) для підвищення ефективності пошуку та аналізу текстових даних, а саме:

- ознайомитися з існуючими підходами використання методів NLP у пошукових системах та спортивних базах даних;
- розробити керуючу інструкцію для LLM, що класифікує запити користувачів для їх маршрутизації до внутрішньої бази даних або зовнішніх інтернет-джерел;
- реалізувати механізм генерації SQL-запитів на основі проаналізованих запитів користувача;
- забезпечити інтеграцію з хмарними платформами LLM;
- розробити користувацький інтерфейс для прямої взаємодії з мовною моделлю;
- протестувати роботу створеної інтелектуальної системи.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ	8
ВСТУП	9
РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ	11
1.1 Основні напрями застосування NLP у інформаційних системах	11
1.2 Методи реалізації задачі Text-to-SQL	15
1.3 Роль великих мовних моделей у задачі Text-to-SQL	16
Висновки до розділу	18
РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ.....	19
2.1 Методи проєктування промптів у задачі Text-to-SQL	19
2.2 Проєктування промптів програми.....	20
2.3 Структура бази даних	22
Висновки до розділу	25
РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ	26
3.1 Основи математичного моделювання великих мовних моделей.....	26
3.2 Аналіз ефективності мовних моделей	27
Висновки до розділу	37
РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.....	39
4.1 Мова програмування C#.....	39
4.2 Windows Forms - розробка графічних інтерфейсів.....	41
4.3 Бібліотеки для роботи з API штучного інтелекту.....	43
4.4 OxyPlot – візуалізація даних у Windows Forms	44
4.5 Апаратне та технічне забезпечення	45

4.6 Особливості реалізації інтелектуальної системи "Figure Skating" на основі NLP та LLMs.....	47
Висновки до розділу	56
РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП–ПРОЄКТУ	58
5.1 Опис ідеї проєкту	58
5.2 Аналіз технологічних можливостей реалізації ідей проєкту	59
5.3 Аналіз ринкових можливостей запуску стартап-проєкту.....	60
Висновки до розділу	61
ВИСНОВКИ.....	63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	64
ДОДАТКИ.....	66
ДОДАТОК А.....	66

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

- AI – Artificial Intelligence (штучний інтелект);
- API – Application Programming Interface (інтерфейс програмування застосунків);
- HTTP – Hypertext Transfer Protocol (протокол передачі гіпертексту);
- ISU – International Skating Union (Міжнародний союз ковзанярів);
- JSON – JavaScript Object Notation;
- LINQ – Language Integrated Query (запити, інтегровані в мову);
- LLM – Large Language Model (велика мовна модель);
- ML – Machine Learning (машинне навчання);
- NLP – Natural Language Processing (обробка природної мови);
- SDK – Software Development Kit (набір інструментів розробки);
- SQL – Structured Query Language (мова структурованих запитів);
- SSD – Solid-State Drive (твердотілий накопичувач);
- БД – База даних;
- ООП – Об'єктно-орієнтоване програмування;
- ОС – Операційна система;
- СУБД – Система управління базами даних;
- ШІ – Штучний інтелект;
- ІС – Інформаційна система.

ВСТУП

Фігурне катання є одним із найпопулярніших видів спорту, що щорічно генерує величезні обсяги структурованих даних про спортсменів, змагання та результати. Національні федерації, тренери, аналітики та медіа потребують швидкого доступу до цієї інформації для прийняття рішень, підготовки аналітичних матеріалів та стратегічного планування. Однак традиційні системи пошуку даних вимагають знання мови SQL, що обмежує коло користувачів і знижує оперативність отримання інформації. У зв'язку з цим виникає потреба у створенні інтелектуальної системи, яка забезпечить доступ до спортивних даних широкому колу користувачів без технічних знань та з підтримкою української мови.

Актуальність такого підходу зумовлена можливістю застосування методів NLP (обробки природної мови), що забезпечують розширені можливості аналізу, класифікації та інтерпретації текстових даних.

Об'єкт дослідження – процес обробки та аналізу текстових даних в спортивних інформаційно-пошукових системах.

Предмет дослідження – методи обробки природної мови (NLP) та застосування великих мовних моделей (LLMs) для реалізації розширеного пошуку та аналізу текстових даних в інформаційно-пошуковій системі "Figure Skating".

Метою роботи є розробка інтелектуальної системи, що за допомогою методів NLP інтерпретує користувацькі запити, визначає їх релевантність, автоматично формує SQL-запити або виконує веб-пошук та надає структуровану й зрозумілу відповідь.

Для досягнення поставленої мети необхідно вирішити наступні *завдання*:

- ознайомитися з існуючими підходами використання методів NLP у пошукових системах та спортивних базах даних;
- розробити керуючу інструкцію для LLM, що класифікує запити користувачів для їх маршрутизації до внутрішньої бази даних або зовнішніх інтернет-джерел;
- реалізувати механізм генерації SQL-запитів на основі проаналізованих запитів

користувача;

- забезпечити інтеграцію з хмарними платформами LLM;
- розробити користувацький інтерфейс для прямої взаємодії з мовною моделлю;
- протестувати роботу створеної інтелектуальної системи.

Наукова новизна дослідження полягає в розробці гібридного підходу до інформаційного пошуку, який динамічно адаптується до наявності даних у спеціалізованій базі даних та загальнодоступних веб-ресурсах, що забезпечує більш інтелектуальну взаємодію між користувачем і спортивною інформаційною системою. Особливістю проєкту є орієнтація на українську мову, яка рідко представлена в комерційних NLP-системах.

Практичне значення роботи полягає в можливості створення універсального інструменту пошуку та аналізу спортивних даних, який забезпечує доступ до інформації як з локальних баз даних, так і з глобальних мережевих джерел, з урахуванням природної мови користувача. Розроблена система може бути використана в спортивних центрах, освітніх установах та медіа, що висвітлюють спортивні події.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

Пошук інформації є невід’ємною частиною життя сучасної людини. Інформаційно-пошукові системи перетворюють необроблену інформацію на потрібні дані за допомогою запитів користувача. Вони застосовуються у різних сферах діяльності, і сфера спорту не є винятком. Однак для ефективного використання величезних масивів текстових даних, що генеруються у спортивних інформаційних системах, необхідні більш досконалі методи, ніж просто пошук за ключовими словами. Обробка природної мови (NLP) стає критично важливою технологією, що дозволяє вийти за межі поверхневого пошуку та забезпечити глибокий аналіз текстових даних.

Перед розробкою власного програмного застосунку, що використовує методи NLP для розширеного пошуку та аналізу, слід добре проаналізувати стан предметної області і провести огляд даних, щоб знайти необхідні підходи та шляхи створення таких систем.

1.1 Основні напрями застосування NLP у інформаційних системах

NLP (Natural Language Processing) – це обробка природної мови, галузь штучного інтелекту (ШІ), яка фокусується на взаємодії між людьми та комп’ютерами на основі природної мови [1]. За допомогою алгоритмів, розроблених NLP, спілкування людей з комп’ютером стало більш ефективним.

Завдяки розвитку NLP сучасні інформаційні системи (ІС) стають розумнішими, гнучкішими та здатними до самонавчання, що істотно покращує якість управління даними та комунікацію з користувачами.

Виділяють такі напрями застосування NLP у інформаційних системах:

1. **Аналіз даних** – NLP здатна обробляти великі обсяги неструктурованих текстів, а саме: повідомлення у соцмережах, електронні листи, коментарі користувачів тощо. Це дає змогу виявляти тренди та закономірності, які допомагають ухвалювати обґрунтовані управлінські рішення. Завдяки цьому компанії можуть виявляти тренди, закономірності та настрої користувачів.

Прикладом можуть слугувати такі платформи як Brandwatch (див. рис. 1.1), Sprout Social, що допомагають компаніям взаємодіяти з аудиторією, аналізуючи пости у Facebook, X та Instagram, для виявлення громадської думки.

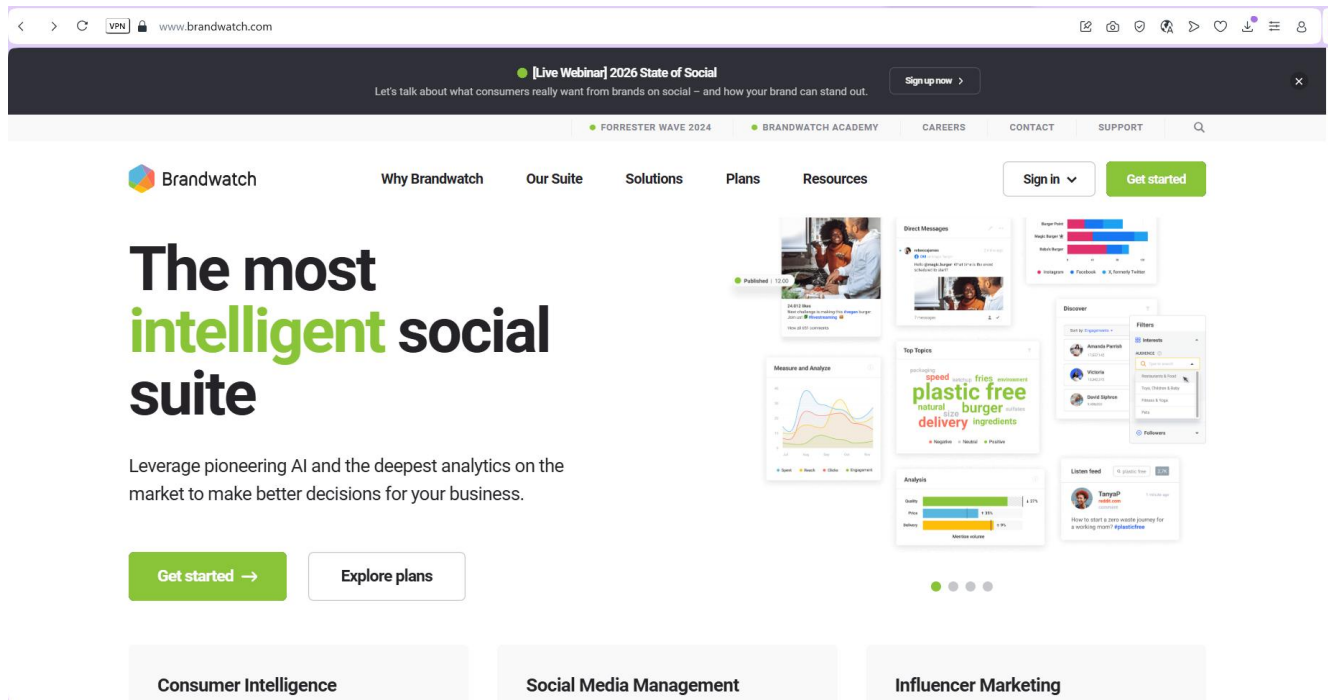


Рисунок 1.1 – Офіційний сайт brandwatch.com

2. **Оптимізація пошуку** – NLP покращує пошукові механізми, дозволяючи системам розуміти запити у формі звичайної мови, а не лише за ключовими словами. Це підвищує точність і швидкість пошуку в ІС.

Наприклад, Google Search – найбільша пошукова система, яка використовує NLP-модель BERT для розуміння контексту запитів. Microsoft Bing і ElasticSearch – системи, які реалізують семантичний пошук, що враховує значення слів, а не лише їхню форму. Semantic Scholar (див. рис. 1.2) – академічна пошукова система, яка визначає змістовну схожість наукових статей.

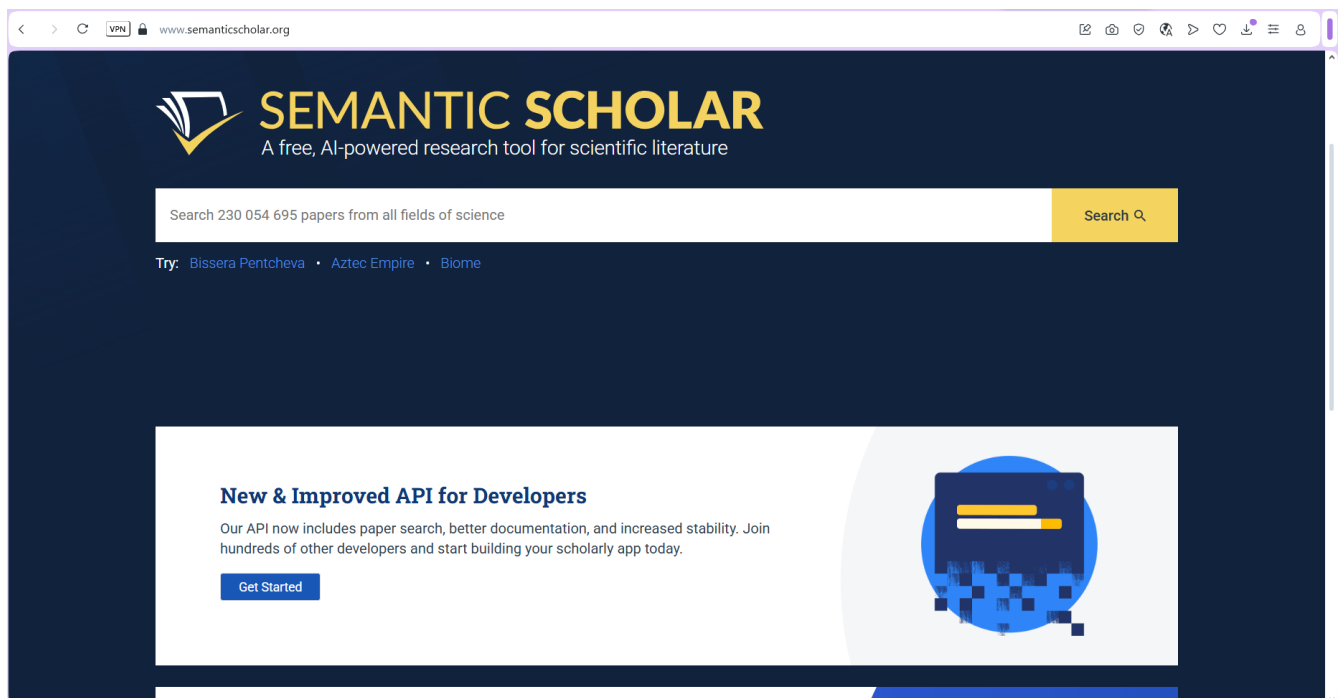


Рисунок 1.2 – Офіційний сайт [semanticscholar.org](https://www.semanticscholar.org)

3. Автоматизація рутинних завдань – обробка природної мови допомагає автоматично класифікувати документи, сортувати електронні листи та фільтрувати спам, що скорочує час і знижує ймовірність людських помилок.

Наприклад, Gmail – використовує NLP для автоматичного розпізнавання спаму та створення коротких відповідей, а Microsoft Outlook – сортує вхідну пошту за важливістю завдяки аналізу змісту.

4. Покращення обслуговування клієнтів – NLP часто використовується в чат-ботах і голосових помічниках, які забезпечують швидку взаємодію з користувачами без участі оператора.

Приклади застосування: Replika (див. рис. 1.3) – чатбот, який розуміє емоційний контекст та спілкується у дружньому форматі. Watson Assistant (IBM) – використовується у банках і контакт-центрах для автоматичних консультацій.



Рисунок 1.3 – Застосунок Replika

5. Також виділяють *спеціалізовані завдання* NLP:

Аналіз тональності – виявлення емоційного забарвлення повідомлень користувачів для оцінки рівня задоволеності клієнтів. Наприклад, MonkeyLearn, Lexalytics, Amazon Comprehend.

Узагальнення тексту - автоматичне скорочення довгих документів або звітів до ключових тез. Це завдання виконують такі інструменти як SMMRY, Scholarcy тощо.

Машинний переклад – переклад текстів різними мовами для розширення міжнародної співпраці. Прикладом можуть слугувати такі сервіси як Google Translate, DeepL, Microsoft Translator.

Розпізнавання мовлення – перетворення усного мовлення на текст. Найпопулярніші Google Speech-to-Text, Apple Siri та Amazon Alexa.

1.2 Методи реалізації задачі Text-to-SQL

У сучасних інформаційних системах користувачі щодня працюють із великими обсягами даних, що зберігаються у реляційних базах. Доступ до цих даних здійснюється мовою структурованих запитів – SQL (Structured Query Language). Проте написання SQL-запитів потребує технічних знань, які має далеко не кожен користувач. Це створює проблему бар'єру між природною мовою людини та мовою баз даних.

Рішенням цієї проблеми стала задача Text-to-SQL, яка дозволяє перетворювати запит, сформульований звичайними словами, у коректний SQL-запит, готовий до виконання над базою даних. Text-to-SQL виступає ключовим напрямом у сфері обробки природної мови (Natural Language Processing, NLP) та штучного інтелекту (ШІ), роблячи аналітику даних доступною для широкого кола користувачів.

Основна мета Text-to-SQL – навчити комп'ютерну систему розуміти запит, виражений природною мовою, і побудувати його еквівалент у вигляді SQL-команди. Задача включає кілька етапів:

- Аналіз запиту – розпізнавання сутностей (таблиць, стовпців, числових значень, умов).
- Побудова проміжного представлення – створення структури, що описує логічні відношення між елементами.
- Генерація SQL-коду, який відповідає структурі запиту й узгоджений зі схемою бази даних.

Розвиток задачі Text-to-SQL відбувався у кілька етапів [2] – від простих правил систем до складних нейронних моделей та великих мовних моделей (LLMs). Кожне покоління підходів робило крок до більш гнучкого розуміння запитів природною мовою.

Перші системи в 1980-2000 роках ґрунтувалися на правилах і шаблонах, які явно описували відповідність між словами користувача та елементами бази даних. Такі системи, як PRECISE, працювали лише у вузьких доменах і вимагали ручного

налаштування. Їхньою перевагою була точність, але недоліком – відсутність гнучкості та масштабованості.

Із розвитком глибокого навчання з'явилися Seq2Seq-моделі, які навчалися автоматично перетворювати текст у SQL. Відомими прикладами стали Seq2SQL та SQLNet. Пізніше такі роботи, як IRNet і PICARD, удосконалили процес, враховуючи структуру бази даних і контролюючи синтаксис під час генерації. Цей етап зробив системи значно точнішими та універсальнішими.

Сучасні системи на базі моделей, таких як GPT-4, Claude, Gemini, здатні розуміти контекст і схему бази даних без спеціального навчання. LLMs не лише генерують правильні SQL-запити, а й можуть вести діалог із користувачем, уточнюючи неясні питання. Це перетворює Text-to-SQL на інтерактивний інструмент аналітики, доступний навіть нефахівцям.

Прикладом застосування технології Text-to-SQL є система Microsoft Power BI Q&A, яка дозволяє користувачам ставити запитання природною мовою для автоматичного створення SQL. Спочатку вона базувалася на шаблонних правилах, а нині інтегрує нейронні та LLM-підходи через Azure OpenAI (GPT-4). Це забезпечує точніше розуміння намірів користувача й динамічне формування запитів без спеціальних знань у програмуванні.

1.3 Роль великих мовних моделей у задачі Text-to-SQL

Великі мовні моделі (Large Language Models, LLMs) – це категорія моделей глибокого навчання, натренованих на величезних обсягах даних, що дає їм змогу розуміти та генерувати природну мову, а також інші типи контенту для виконання широкого спектра завдань [3]. LLMs побудовані на основі нейронної архітектури типу «трансформер», яка відзначається високою ефективністю в обробці послідовностей слів і виявленні закономірностей у тексті.

LLMs забезпечують ефективну генерацію SQL-запитів завдяки здатності до глибокого розуміння природної мови.

Виділяють *чотири методи генерації* Text-to-SQL посиленні LLMs:

Prompt Engineering – стратегія яка використовує деталізовані вхідні промпти і не вимагає додаткового навчання, містить три види технік: Zero-shot, Few-shot та Chain of Thought [4].

Fine-Tuning – донавчання вже попередньо навчених моделей для конкретного завдання. Буває Full-Parameters Fine-Tuning (донавчання всіх параметрів моделі) та Parameter-Efficient Fine-Tuning (донавчання окремих шарів або модулів) [5].

Task-Training – стратегія, яка базується на тренуванні цільової моделі Text-to-SQL з нуля, часто базується на архітектурі Transformer або Mixture of Experts [6].

LLM Agent – найсучасніша стратегія, що передбачає співпрацю кількох інтелектів та використання зовнішніх інструментів для динамічної генерації, перевірки та корекції SQL-запитів. Використовує Велику Мовну Модель як свій центральний механізм для міркування та виконує складні, багатоетапні завдання. Агент здатний планувати, вибирати інструменти і самостійно коригувати свій план дій на основі спостережень. Основними компонентами є модель (LLM), пам'ять (memory) та інструменти (tools) [7].

Наприклад, LangChain/LlamaIndex – найпопулярніші програмні фреймворки, які дозволяють розробникам створювати власних LLM-агентів. Вони надають готові компоненти (модулі пам'яті, планувальники, колекції інструментів) для швидкого прототипування та розгортання агентів. Саме ці фреймворки стандартизували архітектуру LLM Agent, зробивши її доступною для стартапів та корпоративних розробників.

LLMs допомагають вирішити ключові проблеми, притаманні Text-to-SQL системам:

- **Неоднозначність мови** – користувацькі запити можуть містити лексичні та контекстні неоднозначності, а також помилки. LLMs мають бути достатньо стійкими, щоб правильно інтерпретувати такі формулювання.

- **Великі бази даних** – реальні схеми можуть містити сотні таблиць, тому модель не завжди може охопити всі зв'язки та атрибути в одному промпті, що ускладнює правильний вибір таблиць.
- **Складні SQL-запити** – генерація запитів з JOIN, вкладеними підзапитами, GROUP BY та складною логікою фільтрації є проблемною, оскільки вимагає точного розуміння структури бази даних і наміру користувача.

Висновки до розділу

У цьому розділі було комплексно розглянуто застосування технологій обробки природної мови (NLP) в інформаційних системах, зокрема аналіз даних, оптимізацію пошуку, автоматизацію завдань та покращення обслуговування клієнтів.

Особливу увагу приділено задачі Text-to-SQL, яка усуває бар'єр між природною мовою користувача та мовою баз даних, роблячи аналітику доступною для широкого кола користувачів без технічних знань. Досліджено еволюцію методів реалізації Text-to-SQL від простих правилкових систем до сучасних підходів на основі великих мовних моделей (LLMs).

Розглянуто чотири основні стратегії застосування LLMs: Prompt Engineering, Fine-Tuning, Task-Training та LLM Agent, які забезпечують ефективну генерацію SQL-запитів. Ці технології дозволяють вирішувати ключові проблеми неоднозначності мови, роботи з великими базами даних та створення складних запитів.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Методи проєктування промптів у задачі Text-to-SQL

Prompt Engineering є провідною стратегією для ефективного застосування великих мовних моделей (LLMs) у сфері Text-to-SQL. Цей підхід дозволяє істотно підвищити точність і загальну продуктивність LLM при генерації SQL-запитів, часто уникаючи необхідності в донавчанні (fine-tuning).

Суть методу полягає у створенні ретельно продуманих вхідних інструкцій (промптів), які допомагають моделі формувати коректні та релевантні відповіді.

Нижче представлені три ключові методи промпт-інжинірингу, які використовуються для Text-to-SQL:

1. **Zero-shot Prompting** – модель отримує завдання без попередніх прикладів, спираючись лише на базове формулювання проблеми, текстовий запит користувача та опис схеми бази даних [8].

Отже, LLM генерує SQL на основі свого загального досвіду та знань, здобутих під час попереднього навчання.

Перевагою такого методу є можливість миттєво застосовувати модель до нових доменів чи структур баз даних.

Однак через відсутність контекстних прикладів точність може знижуватися, а згенерований SQL-запит може бути недостатньо деталізованим чи помилковим, особливо в разі складних структур даних.

2. **Few-shot Prompting** – користувацький запит доповнюється кількома демонстраціями – приклади того, як саме формуються правильні SQL-запити [9].

У порівнянні з Zero-shot, цей підхід значно покращує продуктивність LLM, особливо при роботі зі складнішими запитам.

Цей підхід вимагає значних зусиль для ручного підбору або автоматизованого відбору високоякісних прикладів, що додає етап додаткової підготовки і знижує його універсальність при застосуванні до нових, неперевічених наборів даних.

3. **Chain of Thought Prompting** – спонукає LLM послідовно та покроково міркувати перш ніж видати остаточний результат. Це посилює її здатність до логічного синтезу інформації, що є критично важливим для комплексних завдань [10].

Метод орієнтований на ситуації, де потрібен покроковий аналіз, наприклад, у логічних, арифметичних чи символічних задачах.

Модель, за інструкцією користувача, генерує проміжні міркування, що імітують людський спосіб вирішення складних проблем – розбиття питання на менші підзадачі.

Ефективність Chain of Thought стає особливо помітною у великих моделях, але завдяки інструкційному навчанню цю техніку здатні використовувати й компактніші LLMs.

2.2 Проєктування промтів програми

Інтелектуальна система функціонує за принципом двоетапного промттування: перший промт генерує структурований вивід (зазвичай SQL) у форматі JSON, а другий промт відповідає за перетворення цього структурованого результату на мову, зрозумілу для користувача.

Перший промт є основним і призначений для спрямування Великої Мовної Моделі на генерацію валідного SQL-запиту або визначення джерела інформації. Промт містить наступні складові:

- **Опис схеми бази даних** – промт надає моделі повну схему чотирьох таблиць (category, competition, result, sportsman), включаючи назви стовпців, типи даних, а також первинні та зовнішні ключі. Це критично важливо для розуміння зв'язків між таблицями та генерації коректних операцій JOIN.
- **Додаткові відомості про дані** – надається контекст про вміст БД (результати фігурного катання) та уточнюються значення специфічних полів (наприклад, fpl, sp, fs, points). Це допомагає подолати неоднозначність природної мови та правильно інтерпретувати запити в межах домену.

- **Інструкції щодо відповіді та формат** – встановлюється конкретний формат виводу (формат JSON) із двома обов'язковими ключами: "Source" (може бути "Database" або "Internet") та "Result". Моделі вказано спершу аналізувати, чи може запит бути виконаний на основі наданої схеми БД про фігурне катання, навіть якщо запит є загальним, але може бути інтерпретований в цьому контексті (наприклад, "Розкажи про спортсменів з Японії"). Також надано приклади очікуваної поведінки та форматів відповідей для різних типів запитів.
 - Якщо відповідь стосується бази даних, "Result" повинен містити валідний SQL запит без зайвих слів чи коментарів.
 - Якщо запит не стосується бази даних (наприклад, про погоду чи політику), модель повинна надати текстову відповідь, базуючись на загальних знаннях ("Source": "Internet").

Промпт поєднує кілька стратегій для досягнення високої точності:

Few-shot Prompting – надається набір ретельно підібраних прикладів ("Користувач: [Запит]" → "Штучний інтелект: [JSON з SQL]"). Це покращує здатність LLM генерувати складний SQL та розуміти логіку JOIN операцій.

Chain of Thought – введено обов'язковий покроковий процес міркування перед генерацією відповіді: 1) Проаналізувати запит; 2) Визначити, чи стосується він даних БД; 3) Сформулювати SQL-запит.

Zero-shot елементи – від моделі очікується, що вона зможе відповісти на будь-який новий, небачений запит (тобто без специфічних навчальних даних), використовуючи лише надану схему та інструкції.

Другий промпт перетворює сирий результат (SQL-запит або результат його виконання, закодований у JSON) на читабельний текст, завершуючи цикл взаємодії з користувачем.

Промпт отримує два основні компоненти: початкове питання користувача (`{user_question}`) та структуровані дані (`{json_data}`), які містять необхідну інформацію для відповіді.

Найважливіші вимоги до фінального виводу:

- **Чіткість відповіді** – відповідь має бути сформульована українською мовою і прямо відповідати на поставлене запитання.
- **Конвертація даних** – моделі суворо заборонено відображати сирий JSON чи його структуру, всі дані мають бути повністю перетворені на природний текст.
- **Стилістична лаконічність** – вимагається надавати тільки сутність відповіді. Необхідно уникати будь-яких вступних, підсумкових, пояснювальних чи зайвих фраз (наприклад, "Згідно з даними...", "Так, звичайно..."). Відповідь має бути максимально лаконічною, але при цьому повною по суті питання.
- **Коректність мови** – окрема вимога стосується коректного відтворення імен, прізвищ та інших власних назв в українському тексті.

Таким чином, другий промпт слугує для покращення взаємодії з користувачем, перетворюючи технічні результати, згенеровані Text-to-SQL системою, на зрозумілий і доступний формат.

2.3 Структура бази даних

Розробка бази даних розпочалася з глибокого аналізу предметної області – фігурного катання, яке вважається одним із найграціозніших видів спорту. Воно структурно розділене на чотири основні дисципліни: одиночне чоловіче, одиночне жіноче, парне катання та спортивні танці на льоду.

Усі фігуристи виконують дві програми – коротку та довільну, – за які отримують оцінки від суддів. Бали нараховуються за складність і якість виконання технічних елементів, такі як стрибки, піруети та кроки. Сума балів за обидві програми визначає підсумковий результат змагань.

Щорічно проводяться численні міжнародні змагання, що збирають спортсменів з усього світу. Найпрестижнішими з них є чемпіонати Європи, чотирьох континентів та світу.

Отже, структура бази даних включає чотири ключові таблиці: "спортсмен", "змагання", "категорія" та "результат".

Column	Type
◇ id	int
◇ name	varchar(45)
◇ nationality	varchar(45)

Рисунок 2.1 – Таблиця "sportsman"

Таблиця "sportsman" призначена для зберігання основних даних про спортсменів, а саме: їхнього унікального ідентифікатора (ID), повного імені та країни, яку вони представляють.

Column	Type
◇ id	int
◇ name	varchar(45)

Рисунок 2.2 – Таблиця "category"

Таблиця "category" зберігає унікальні ідентифікатори та назви всіх категорій фігурного катання, що згадувалися раніше.

Column	Type
◇ id	int
◇ name	varchar(45)
◇ year	int

Рисунок 2.3 – Таблиця "competition"

Для кожного змагання буде зафіксовано його назву, рік проведення та унікальний ідентифікатор у таблиці "competition".

Column	Type
category_id	int
competition_id	int
fpl	int
fs	int
id	int
points	float
sp	int
sportsman2_id	int
sportsman_id	int

Рисунок 2.4 – Таблиця "result"

Таблиця "result" відповідає за зв'язок між усіма попередніми даними. Її структура включає наступні атрибути:

category_id, *competition_id*, *sportsman_id*, *sportsman2_id* – ці ідентифікатори є ключовими для встановлення відносин між таблицями;

fpl – поле для збереження фінального місця спортсмена;

sp – тут фіксується позиція, отримана за коротку програму;

fs – вказує на місце, зайняте в довільній програмі;

points – загальна кількість балів, нарахованих спортсмену.

Table	Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra
category	id	int		NO			select,insert,update,references	
category	name	varchar(45)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
competition	id	int		NO			select,insert,update,references	
competition	name	varchar(45)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
competition	year	int		NO			select,insert,update,references	
result	category_id	int		NO			select,insert,update,references	
result	competition_id	int		NO			select,insert,update,references	
result	fpl	int		NO			select,insert,update,references	
result	fs	int		NO			select,insert,update,references	
result	id	int		NO			select,insert,update,references	
result	points	float		NO			select,insert,update,references	
result	sp	int		NO			select,insert,update,references	
result	sportsman2_id	int		YES			select,insert,update,references	
result	sportsman_id	int		NO			select,insert,update,references	
sportsman	id	int		NO			select,insert,update,references	
sportsman	name	varchar(45)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
sportsman	nationality	varchar(45)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	

Рисунок 2.5 – Структура БД

Висновки до розділу

У другому розділі розглянуто методи проєктування промптів для задачі Text-to-SQL, зокрема Zero-shot, Few-shot та Chain of Thought Prompting, які дозволяють підвищити точність генерації SQL-запитів великими мовними моделями.

Детально описано двоетапну систему промптування програми: перший промпт генерує SQL-запити у форматі JSON на основі схеми бази даних, а другий перетворює результати в зрозумілий текст.

Представлено структуру бази даних для предметної області фігурного катання, яка включає чотири таблиці: "sportsman", "category", "competition" та "result", що зберігають інформацію про спортсменів, категорії змагань, турніри та їхні результати.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Основи математичного моделювання великих мовних моделей

Великі мовні моделі, такі як GPT, базуються на архітектурі трансформерів та використовують методи глибокого навчання для моделювання імовірнісного розподілу слів у тексті. Основна мета таких моделей – апроксимація умовної ймовірності наступного слова в послідовності:

$$P(w_t | w_1, w_2, \dots, w_{t-1}; \theta), \quad (3.1)$$

де w_t – слово на позиції t , а θ – вектор параметрів моделі (ваги нейронної мережі) [11].

Текстові дані подаються у вигляді токенів, які перетворюються у вектори за допомогою ембедінгових матриць:

$$x_i = E[w_i], \quad (3.2)$$

де $E \in R^{V \times d}$ – матриця ембедінгів (розмір словника V , розмірність вектора d) [12].

LLM використовують трансформери, в яких основним обчислювальним блоком є механізм багатоголової уваги (Multi-head Attention). Для кожного вхідного вектора x обчислюються ключі K , запити Q та значення V :

$$Q = X \cdot W^Q, \quad (3.3)$$

$$K = X \cdot W^K, \quad (3.4)$$

$$V = X \cdot W^V, \quad (3.5)$$

де $W^Q, W^K, W^V \in R^{d \times d_k}$ – параметри моделі.

Далі увага обчислюється як:

$$Attention(Q, K, V) = softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V \quad (3.6)$$

Математично це дозволяє моделі динамічно зважувати вплив різних слів у контексті.

Процес навчання передбачає мінімізацію функції втрат (loss function), зазвичай – крос-ентропійної втрати для задачі прогнозування слова:

$$L(\theta) = -\sum_{i=1}^T \log P(w_i | w_1, \dots, w_{i-1}; \theta), \quad (3.7)$$

де T – довжина послідовності.

Оптимізація здійснюється градієнтними методами, зокрема алгоритмом Adam (Adaptive Moment Estimation), який враховує моменти першого та другого порядку градієнтів:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla_{\theta} L(\theta) \quad (3.8)$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (\nabla_{\theta} L(\theta))^2 \quad (3.9)$$

$$\theta_t = \theta_{t-1} - \eta \cdot \left(\frac{m_t}{\sqrt{v_t + \epsilon}} \right) \quad (3.10)$$

Для запобігання перенавчанню використовуються методи регуляризації, зокрема Dropout, Weight Decay, а також розширення даних (data augmentation). Після початкового навчання LLM зазвичай проходить етап донавчання (fine-tuning), коли модель адаптується до конкретного завдання. Це можна математично описати як оптимізацію тієї ж функції втрат, але на іншій (меншій) вибірці:

$$\theta^* = \operatorname{argmin}_{\theta} L_{task}(\theta; D_{task}) \quad (3.11)$$

де D_{task} – набір даних конкретного завдання (наприклад, класифікація, узагальнення тощо) [13].

3.2 Аналіз ефективності мовних моделей

Для оцінки точності та надійності функціонування інтелектуальної пошукової системи на базі мовних моделей, було проведено серію з 30 експериментів. До порівняння залучено шість потужних мовних моделей від різних провайдерів:

- Gemini 2.5 Flash Preview
- Gemini 2.0
- GPT-4.1
- GPT-4.1-nano
- Claude Opus 4
- Claude 3.5 Haiku

Кожна з моделей отримала однакову групу із 5 запитань українською мовою, що імітують запити кінцевого користувача у сфері фігурного катання. Підібрані запити охоплюють різні типи операцій над базою даних, зокрема фільтрацію, агрегацію, сортування, обчислення максимуму/мінімуму.

Отримані відповіді оцінювалися за допомогою наступних метрик:

- ***Exact Matching Accuracy (EM)*** – вимагає, щоб згенерований SQL-запит був ідентичним еталонному. Якщо ідентичний тоді функція дорівнює 1, якщо ні – 0.
- ***Execution Accuracy (EX)*** – оцінює, чи повертає згенерований запит той самий результат, що й еталонний запит, при виконанні на базі даних. Якщо повертає тоді функція дорівнює 1, якщо ні – 0.
- ***Час генерації*** – швидкість виконання SQL-запиту.
- ***Valid Efficiency Score (VES)*** – враховує як правильність запиту, так і його ефективність (швидкість) виконання у порівнянні з еталонним. Обчислюється за формулою:

$$VES = \frac{1}{N} \sum_{i=1}^N \left(\mathbb{I}(Q_i^{gen} = Q_i^{gold}) \cdot \frac{T_{gold}}{T_{gen}} \right) \quad (3.12)$$

N – позначає загальну кількість запитів.

Q_i^{gen} – позначає згенерований SQL-запит для i -го прикладу.

Q_i^{gold} – позначає еталонний SQL-запит для i -го прикладу.

$\mathbb{I}(\cdot)$ – індикаторна функція, яка дорівнює 1, якщо згенерований SQL-запит ідентичний еталонному (тобто, $Q_i^{gen} = Q_i^{gold}$), і 0 в іншому випадку.

T_{gold} – час виконання еталонного SQL-запиту.

T_{gen} – час виконання згенерованого SQL-запиту.

- ***Вартість запиту*** – фактичні витрати на токени.

Переходимо до оцінки кожного запиту:

1. Хто переміг з чоловіків на чемпіонаті світу в 2022 році?

Еталонний запит:

```

SELECT
    s.name AS sportsman,
    s.nationality,
    r.points,
    r.sp AS short_program,
    r.fs AS free_skate
FROM result r
INNER JOIN competition comp ON r.competition_id = comp.id
INNER JOIN category cat ON r.category_id = cat.id
INNER JOIN sportsman s ON r.sportsman_id = s.id
WHERE comp.name = 'World Figure Skating Championship'
    AND comp.year = 2022
    AND cat.name = 'Man'
    AND r.fpl = 1;

```

Час виконання – 0,0009 сек.

Таблиця 3.1 – Таблиця результатів для першого запиту

Назва моделі	SQL-запит	EM	EX	Час генерації (сек)	VES	Всього токенів	Загальна вартість (\$)
1	2	3	4	5	6	7	8
Claude Opus 4	SELECT s.name FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name = 'World Figure Skating Championship' AND c.year = 2022 AND cat.name = 'Man' AND r.fpl = 1	0	1	0,0010	0	2642	0.045690
Claude 3.5 Haiku	SELECT s.name, s.nationality FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name = 'World Figure Skating Championship' AND c.year = 2022 AND cat.name = 'Man' AND r.fpl = 1	0	1	0,0012	0	2642	0.002437

Продовження таблиці 3.1

1	2	3	4	5	6	7	8
Gemini 2.5 Flash Prev	SELECT T1.name FROM sportsman AS T1 INNER JOIN result AS T2 ON T1.id = T2.sportsman_id INNER JOIN competition AS T3 ON T2.competition_id = T3.id INNER JOIN category AS T4 ON T2.category_id = T4.id WHERE T3.name = 'World Figure Skating Championship' AND T3.year = 2022 AND T4.name = 'Man' AND T2.fpl = 1	0	1	0,0010	0	1780	0.001355
Gemini 2.0 Flash	SELECT s.name FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name = 'World Figure Skating Championship' AND c.year = 2022 AND cat.name = 'Man' AND r.fpl = 1	0	1	0,0009	0	1782	0.000971
GPT 4.1	SELECT s.name FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name = 'World Figure Skating Championship' AND c.year = 2022 AND cat.name = 'Man' AND r.fpl = 1	0	1	0,0009	0	1777	0.004004
GPT 4.1 nano	SELECT s.name FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name = 'World Figure Skating Championship' AND c.year = 2022 AND cat.name = 'Man' AND r.fpl = 1	0	1	0,0009	0	1777	0.000200

2. Виведи всіх фігуристів з Японії.

Еталонний запит:

```
SELECT * FROM sportsman
WHERE nationality = 'JPN'
```

Час виконання – 0,0003 сек.

Таблиця 3.2 – Таблиця результатів для другого запиту

Назва моделі	SQL-запит	EM	EX	Час генерації (сек)	VES	Всього токенів	Загальна вартість (\$)
Claude Opus 4	SELECT * FROM sportsman WHERE nationality = 'JPN'	1	1	0,0003	1	2557	0.039975
Claude 3.5 Haiku	SELECT * FROM sportsman WHERE nationality = 'JPN'	1	1	0,0003	1	2557	0.002132
Gemini 2.5 Flash Prev	SELECT name FROM sportsman WHERE nationality = 'JPN'	0	1	0,0003	0	1723	0.001243
Gemini 2.0 Flash	SELECT * FROM sportsman WHERE nationality = 'JPN'	1	1	0,0003	1	1723	0.000888
GPT-4.1	SELECT * FROM sportsman WHERE nationality = 'JPN'	1	1	0,0003	1	1719	0.003570
GPT-4.1 nano	SELECT s.name FROM sportsman s WHERE s.nationality = 'JPN'	0	1	0,0003	0	1731	0.000183

3. Хто мав найменше балів на чемпіонаті Європи за всі роки з усіх категорій?

Еталонний запит:

```

SELECT
    r.points,
    r.year,
    c.name AS category,
    s1.name AS sportsman1,
    s2.name AS sportsman2,
    s1.nationality AS nationality
FROM (
    SELECT
        res.points,
        res.category_id,
        res.sportsman_id,
        res.sportsman2_id,
        comp.year
    FROM result res
    INNER JOIN competition comp ON res.competition_id = comp.id
    WHERE comp.name = 'European Figure Skating Championship'
    ORDER BY res.points ASC
    LIMIT 1
) r
INNER JOIN category c ON r.category_id = c.id
INNER JOIN sportsman s1 ON r.sportsman_id = s1.id
LEFT JOIN sportsman s2 ON r.sportsman2_id = s2.id;

```

Час виконання – 0,0007 сек.

Таблиця 3.3 – Таблиця результатів для третього запиту

Назва моделі	SQL-запит	EM	EX	Час генерації (сек)	VES	Всього токенів	Загальна вартість (\$)
1	2	3	4	5	6	7	8
Claude Opus 4	SELECT s.name, r.points, cat.name AS category_name, c.year FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name = 'European Figure Skating Championship' ORDER BY r.points ASC LIMIT 1	0	1	0,0008	0	2649	0.045855
Claude 3.5 Haiku	SELECT s.name, s.nationality, c.name AS competition_name, c.year, cat.name AS category, r.points FROM result r JOIN sportsman s ON r.sportsman_id = s.id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name = 'European Figure Skating Championship' ORDER BY r.points ASC LIMIT 1	0	1	0,0008	0	2674	0.002546
Gemini 2.5 Flash Prev	SELECT s.name, r.points, c.name AS competition_name, c.year FROM result r JOIN sportsman s ON r.sportsman_id = s.id JOIN competition c ON r.competition_id = c.id WHERE c.name = 'European Figure Skating Championship' ORDER BY r.points ASC LIMIT 1	0	1	0,0009	0	1822	0.001436
Gemini 2.0 Flash	SELECT s.name, MIN(r.points) AS min_points, c.name AS competition_name, c.year, cat.name AS category_name FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name LIKE '%European Figure Skating Championship%' GROUP BY s.name, c.name, c.year, cat.name ORDER BY min_points ASC LIMIT 1	0	1	0,0012	0	1783	0.000968

Продовження таблиці 3.3

1	2	3	4	5	6	7	8
GPT-4.1	SELECT s.name, cat.name AS category, c.year, r.points FROM result r JOIN sportsman s ON s.id = r.sportsman_id JOIN competition c ON c.id = r.competition_id JOIN category cat ON cat.id = r.category_id WHERE c.name = 'European Figure Skating Championship' ORDER BY r.points ASC LIMIT 1	0	1	0,0008	0	1795	0.004118
GPT-4.1 nano	SELECT s.name, r.points, c.name, c.year, cat.name FROM result r JOIN sportsman s ON r.sportsman_id = s.id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id ORDER BY r.points ASC LIMIT 1;	0	0	0,0016	0	1787	0.000203

4. Хто отримав серед жінок найбільше балів на чемпіонаті світу з усіх років?

Еталонний запит:

```
SELECT s.name, r.points, c.year
FROM result r
JOIN sportsman s ON s.id = r.sportsman_id
JOIN competition c ON c.id = r.competition_id
JOIN category cat ON cat.id = r.category_id
WHERE cat.name = 'Woman'
AND c.name = 'World Figure Skating Championship'
ORDER BY r.points DESC
LIMIT 1
```

Час виконання – 0,0008 сек.

Таблиця 3.4 – Таблиця результатів для четвертого запиту

Назва моделі	SQL-запит	EM	EX	Час генерації	VES	Всього токенів	Загальна вартість
Claude Opus 4	SELECT s.name, r.points, c.year FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name = 'World Figure Skating Championship' AND cat.name = 'Woman' ORDER BY r.points DESC LIMIT 1	0	1	0,0013	0	2650	0.045870
Claude 3.5 Haiku	SELECT s.name, MAX(r.points) as max_points FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name = 'World Figure Skating Championship' AND cat.name = 'Woman' GROUP BY s.name ORDER BY max_points DESC LIMIT 1	0	1	0,0008	0	2650	0.002446
Gemini 2.5 Flash Prev	SELECT s.name, r.points FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE cat.name = 'Woman' AND c.name = 'World Figure Skating Championship' ORDER BY r.points DESC LIMIT 1	0	1	0,0012	0	1789	0.001366
Gemini 2.0 Flash	SELECT s.name, MAX(r.points) AS max_points FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN category c ON r.category_id = c.id JOIN competition comp ON r.competition_id = comp.id WHERE c.name = 'Woman' AND comp.name = 'World Figure Skating Championship' GROUP BY s.name ORDER BY max_points DESC LIMIT 1	0	1	0,0008	0	1801	0.000994
GPT-4.1	SELECT s.name, r.points, c.year FROM result r JOIN sportsman s ON s.id = r.sportsman_id JOIN competition c ON c.id = r.competition_id JOIN category cat ON cat.id = r.category_id WHERE cat.name = 'Woman' AND c.name = 'World Figure Skating Championship' ORDER BY r.points DESC LIMIT 1	1	1	0,0008	1	1788	0.004056
GPT-4.1 nano	SELECT s.name, SUM(r.points) AS total_points FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN category c ON r.category_id = c.id WHERE c.name = 'Woman' GROUP BY s.id ORDER BY total_points DESC LIMIT 1	0	0	0,0009	0	1776	0.000198

5. Хто зайняв останнє місце у спортивних парах на чемпіонаті Європи в 2022

році?

Еталонний запит:

```
SELECT
  s1.name AS sportsman1,
  s2.name AS sportsman2,
  r.fpl AS final_placement,
  r.points
FROM result r
JOIN category cat ON r.category_id = cat.id
JOIN competition comp ON r.competition_id = comp.id
JOIN sportsman s1 ON r.sportsman_id = s1.id
JOIN sportsman s2 ON r.sportsman2_id = s2.id
WHERE cat.name = 'Pairs'
      AND comp.name = 'European Figure Skating Championship'
      AND comp.year = 2022
ORDER BY r.fpl DESC
LIMIT 1;
```

Час виконання – 0,0007 сек.

Таблиця 3.5 – Таблиця результатів для п'ятого запиту

Назва моделі	SQL-запит	EM	EX	Час генерації (сек)	VES	Всього токенів	Загальна вартість (\$)
1	2	3	4	5	6	7	8
Claude Opus 4	SELECT s1.name AS sportsman1_name, s2.name AS sportsman2_name, r.fpl FROM result r JOIN sportsman s1 ON r.sportsman_id = s1.id JOIN sportsman s2 ON r.sportsman2_id = s2.id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name = 'European Figure Skating Championship' AND c.year = 2022 AND cat.name = 'Pairs' ORDER BY r.fpl DESC LIMIT 1	0	1	0,0009	0	2684	0.048240
Claude 3.5 Haiku	SELECT s1.name AS sportsman1, s2.name AS sportsman2, r.fpl AS last_place FROM result r JOIN sportsman s1 ON r.sportsman_id = s1.id JOIN sportsman s2 ON r.sportsman2_id = s2.id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name = 'European Figure Skating Championship' AND c.year = 2022 AND cat.name = 'Pairs' ORDER BY r.fpl DESC LIMIT 1	0	1	0,0012	0	2689	0.002593

Продовження таблиці 3.5

1	2	3	4	5	6	7	8
Gemini 2.5 Flash Prev	SELECT s1.name, s2.name FROM result r JOIN sportsman s1 ON r.sportsman_id = s1.id JOIN sportsman s2 ON r.sportsman2_id = s2.id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE cat.name = 'Pairs' AND c.name = 'European Figure Skating Championship' AND c.year = 2022 ORDER BY r.fpl DESC LIMIT 1	0	1	0,0008	0	1817	0.001425
Gemini 2.0 Flash	SELECT s.name, s.nationality FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN category c ON r.category_id = c.id JOIN competition comp ON r.competition_id = comp.id WHERE c.name = 'Pairs' AND comp.name = 'European Figure Skating Championship' AND comp.year = 2022 ORDER BY r.fpl DESC LIMIT 1	0	0	0,0008	0	1806	0.000854
GPT-4.1	SELECT s1.name AS sportsman1, s2.name AS sportsman2 FROM result r JOIN category cat ON r.category_id = cat.id JOIN competition c ON r.competition_id = c.id JOIN sportsman s1 ON r.sportsman_id = s1.id LEFT JOIN sportsman s2 ON r.sportsman2_id = s2.id WHERE cat.name = 'Pairs' AND c.name = 'European Figure Skating Championship' AND c.year = 2022 AND r.fpl = (SELECT MAX(r2.fpl) FROM result r2 JOIN category cat2 ON r2.category_id = cat2.id JOIN competition c2 ON r2.competition_id = c2.id WHERE cat2.name = 'Pairs' AND c2.name = 'European Figure Skating Championship' AND c2.year = 2022)	0	1	0,0010	0	1865	0.004672
GPT-4.1 nano	SELECT s.name FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name = 'European Figure Skating Championship' AND c.year = 2022 AND cat.name = 'Pairs' AND r.fpl = (SELECT MAX(r2.fpl) FROM result r2 WHERE r2.competition_id = c.id AND r2.category_id = cat.id)	0	0	0,0009	0	1789	0.000203

Таблиця 3.6 – Таблиця результатів аналізу

Назва моделі	EM	EX	Середній час генерації (сек)	VES	Середня кількість токенів	Середня вартість (\$)
Claude Opus 4	0.20	1.00	0.00101	0.20	2636.4	0.045126
Claude 3.5 Haiku	0.20	1.00	0.00104	0.20	2642.4	0.0024308
Gemini 2.5 Flash Prev	0.00	1.00	0.00098	0.00	1794.2	0.001365
Gemini 2.0 Flash	0.20	0.80	0.00092	0.20	1779	0.000935
GPT-4.1	0.40	1.00	0.00092	0.40	1788.8	0.004084
GPT-4.1 nano	0.00	0.40	0.00108	0.00	1772	0.0001974

Отже, модель GPT-4.1 є найкращою за показниками точності та ефективності, досягнувши найвищих середніх балів EM (0,40) та VES (0,4000). Водночас, більшість потужних моделей (GPT-4.1, Claude Opus 4, Claude 3.5 Haiku, Gemini 2.5 Flash Preview) демонструють ідеальну коректність виконання (EX = 1,00), що свідчить про високу надійність у поверненні правильного результату, незалежно від синтаксичної ідентичності еталонному запиту. Gemini 2.5 Flash Preview найшвидшою та однією з найдешевших, але має найнижчий EM/VES (0,00). Модель GPT-4.1 nano є найдешевшою, але найменш надійною за коректністю.

Висновки до розділу

У даному розділі розглянуто математичні основи функціонування великих мовних моделей на базі архітектури трансформерів, зокрема механізм багатоголової уваги, процес навчання через мінімізацію крос-ентропійної функції втрат та алгоритм оптимізації Adam.

Проведено експериментальне дослідження шести сучасних мовних моделей (Gemini 2.5 Flash Preview, Gemini 2.0, GPT-4.1, GPT-4.1 nano, Claude Opus 4, Claude 3.5

Наіку) на серії тестів із запитамі українською мовою у предметній області фігурного катання з використанням метрик EM, EX, VES та вартості запитів.

Результати показали, що модель GPT-4.1 демонструє найкращі показники точності та ефективності, тоді як більшість потужних моделей досягли ідеальної коректності виконання. Модель Gemini 2.5 Flash Preview виявилася найшвидшою та найекономічнішою, а GPT-4.1 nano – найдешевшою, але з найнижчою коректністю виконання.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Мова програмування C#

Мова програмування C# є однією з найпопулярніших мов для розробки програмного забезпечення в екосистемі Microsoft. Створена на початку 2000-х років, вона поєднує потужність об'єктно-орієнтованого програмування з сучасними можливостями функціонального підходу та асинхронного виконання коду [14].

Ключові особливості C# [15]:

- Об'єктно-орієнтована парадигма з підтримкою інкапсуляції, наслідування та поліморфізму.
- Строга статична типізація з можливістю неявного визначення типів через var.
- Автоматичне управління пам'яттю через збирач сміття.
- Підтримка асинхронного програмування через async/await.
- LINQ для уніфікованої роботи з різними джерелами даних.
- Підтримка функціонального програмування через лямбда-вирази та делегати.
- Безпека типів та обробка виключень.

C# розрізняє типи-значення та посилальні типи. Типи-значення зберігаються у стеку та містять безпосередньо дані, тоді як посилальні типи зберігаються у купі, а змінні містять посилання на об'єкти. Це розмежування є важливим для розуміння продуктивності та поведінки програм.

Таблиця 4.1 – Типи даних у C#

Категорія	Типи	Опис
1	2	3
Числові цілі	byte, short, int, long	Цілі числа різного діапазону
Числові дробові	float, double, decimal	Числа з плаваючою комою
Логічний	bool	Значення true або false

Продовження таблиці 4.1

1	2	3
Символьний	char, string	Окремі символи та рядки тексту
Об'єктний	object	Базовий тип для всіх об'єктів
Nullable	int?, bool?, тощо	Типи, що можуть містити null

C# є мовою, що повністю підтримує ООП, і її дизайн значною мірою ґрунтується на чотирьох ключових принципах ООП [16], що представленні у таблиці 4.2.

Таблиця 4.2 – Принципи ООП у C#

Принцип	Реалізація	Призначення
Інкапсуляція	Модифікатори доступу (public, private, protected)	Приховування внутрішньої реалізації
Наслідування	Ключове слово : (двокрапка)	Створення ієрархії класів
Поліморфізм	Віртуальні методи (virtual, override)	Різна поведінка для споріднених типів
Абстракція	Абстрактні класи та інтерфейси	Визначення контрактів без реалізації

Інтерфейси у C# слугують шаблонами поведінки. Головна перевага інтерфейсів над абстрактними класами полягає в тому, що клас може реалізовувати множинну кількість інтерфейсів, забезпечуючи високу гнучкість архітектури. Сучасні версії C# також підтримують методи за замовчуванням в інтерфейсах.

Головна ідея ООП полягає в тому, щоб зробити код більш організованим, модульним, багаторазово використовуваним і простим в обслуговуванні.

4.2 Windows Forms - розробка графічних інтерфейсів

Windows Forms є технологією створення графічних інтерфейсів користувача для операційної системи Windows. Вона базується на обгортці навколо Windows API та надає об'єктно-орієнтований підхід до розробки десктопних застосунків [17].

Основою Windows Forms є простір імен System.Windows.Forms, який містить класи для форм, елементів управління, обробки подій та діалогових вікон. Кожна форма представлена об'єктом класу Form, який може містити різноманітні елементи управління - кнопки, текстові поля, списки та інші компоненти.

Основні компоненти архітектури:

- Form – головне вікно застосунку.
- Control – базовий клас для всіх елементів управління.
- Event – система подій для взаємодії з користувачем.
- Application – клас для управління життєвим циклом програми.

Основні елементи управління у Windows Forms – це компоненти, які ви розміщуєте на формі для створення графічного інтерфейсу користувача. Вони дозволяють користувачеві взаємодіяти з програмою.

Таблиця 4.3 – Елементи управління Windows Forms

Елементи	Клас	Призначення
1	2	3
Кнопка	Button	Виклик дій при натисканні
Текстове поле	TextBox	Введення текстових даних
Мітка	Label	Відображення статичного тексту
Список	ListBox, ComboBox	Вибір значень зі списку
Таблиця	DataGridView	Відображення табличних даних

Продовження таблиці 4.3

1	2	3
Меню	MenuStrip, ContextMenuStrip	Організація команд програми
Панель інструментів	ToolStrip	Швидкий доступ до функцій
Прогрес-бар	ProgressBar	Візуалізація прогресу операцій

Модель подій у Windows Forms базується на делегатах та паттерні спостерігач. Кожен елемент управління може генерувати різні події у відповідь на дії користувача або зміни стану. Розробник підписується на потрібні події та визначає обробники, які виконуються при їх виникненні.

Таблиця 4.4 – Типові події елементів управління

Подія	Опис	Використання
Click	Натискання лівої кнопки миші	Виконання команд через кнопки
TextChanged	Зміна тексту в полі	Валідація введених даних
KeyPress	Натискання клавіші	Обробка введення з клавіатури
Load	Завантаження форми	Ініціалізація даних при відкритті
Closing	Закриття форми	Збереження даних перед виходом
MouseMove	Рух курсору миші	Відстеження позиції курсору
Paint	Перемальовування форми	Власне малювання графіки

Отже, Windows Forms використовується для швидкого створення настільних застосунків з графічним інтерфейсом користувача. Її ключовими перевагами є простота розробки та тісна інтеграція з мовою C#. Хоча вона поступається сучасним фреймворкам у візуальній привабливості та підтримці високих роздільних здатностей, вона залишається надійною опцією для бізнес-застосунків та внутрішніх утиліт.

4.3 Бібліотеки для роботи з API штучного інтелекту

Anthropic.SDK є офіційним програмним інтерфейсом для взаємодії з моделями Claude від компанії Anthropic. Ця бібліотека надає зручні методи для надсилання запитів до API, отримання відповідей та обробки різних типів повідомлень [18]. Інтеграція з Claude дозволяє додавати до Windows Forms застосунків можливості обробки природної мови, генерації тексту та аналізу контенту.

Таблиця 4.5 – Основні класи та методи Anthropic.SDK

Клас/Метод	Призначення	Приклад використання
AnthropicClient	Головний клієнт для взаємодії з API	<pre>var client = new AnthropicClient(apiKey);</pre>
Message	Клас для представлення повідомлень у діалозі	<pre>var message = new Message { Role = "user", Content = "text" };</pre>
CreateMessageAsync()	Асинхронне створення та відправка повідомлення	<pre>var response = await client.Messages.CreateAsync(request);</pre>
MessageRequest	Об'єкт запиту з параметрами моделі	<pre>var request = new MessageRequest { Model = "claude-3-opus", MaxTokens = 1024 };</pre>
MessageResponse	Об'єкт відповіді від API	<pre>var text = response.Content[0].Text;</pre>

Простір імен OpenAI.Chat надає функціонал для роботи з моделями GPT від OpenAI. Ця бібліотека дозволяє створювати інтелектуальні асистенти у Windows Forms, системи питань-відповідей та інші застосунки на базі великих мовних моделей.

Таблиця 4.6 – Основні компоненти OpenAI.Chat

Компонент	Опис	Застосування
ChatClient	Клієнт для роботи з чат API	Ініціалізація з'єднання з API OpenAI
ChatCompletion	Об'єкт завершеної відповіді чату	Отримання згенерованого тексту
ChatMessage	Повідомлення в історії діалогу	Зберігання контексту розмови
ChatCompletionOptions	Параметри для налаштування генерації	Встановлення температури, max tokens тощо
CompleteChatAsync()	Асинхронний метод генерації відповіді	Отримання відповіді від моделі GPT

4.4 OxyPlot – візуалізація даних у Windows Forms

OxyPlot є потужною бібліотекою для створення графіків та діаграм, яка має спеціальну версію для Windows Forms. Вона дозволяє інтегрувати інтерактивні графіки безпосередньо у форми застосунків, забезпечуючи візуалізацію результатів аналізу даних [19].

Таблиця 4.7 – Компоненти OxyPlot для Windows Forms

Компонент	Призначення	Інтеграція
1	2	3
PlotView	Елемент управління для відображення графіка	Додається на форму як звичайний контрол
PlotModel	Модель даних графіка	Прив'язується до PlotView через властивість Model
LineSeries	Лінійний графік	Додається до PlotModel.Series

Продовження таблиці 4.7

1	2	3
ColumnSeries	Стовпчаста діаграма	Відображення категоріальних даних
Legend	Легенда графіка	Автоматично генерується з назв серій

Таблиця 4.8 – Методи роботи з OxyPlot

Метод/Властивість	Опис	Використання у Windows Forms
PlotView.Model	Встановлення моделі для відображення	plotView.Model = plotModel;
InvalidatePlot()	Оновлення відображення	plotView.InvalidatePlot(true);
Series.Add()	Додавання серії даних	model.Series.Add(new LineSeries());
Axes.Add()	Налаштування осей	model.Axes.Add(new LinearAxis());
ExportToPng()	Експорт графіка у зображення	Збереження результатів візуалізації

При розробці Windows Forms застосунків OxyPlot забезпечує зручну інтеграцію через елемент управління PlotView, який можна додати на форму через дизайнер або програмно. Це дозволяє створювати інтерактивні дашборди для візуалізації результатів роботи AI моделей, статистичних даних та метрик продуктивності.

4.5 Апаратне та технічне забезпечення

Для ефективної розробки та експлуатації Windows Forms з інтеграцією штучного інтелекту необхідна збалансована **апаратна конфігурація**.

- Рекомендується використовувати **чотириядерний процесор** з тактовою частотою від 2.5 ГГц, що забезпечить належну швидкість компіляції коду.

- Обсяг *оперативної пам'яті* має становити не менше **16 ГБ** для комфортної роботи середовища розробки та одночасного тестування програм. При роботі з великими базами даних доцільно збільшити обсяг до 32 ГБ для уникнення затримок.
- Використання твердотільних накопичувачів **SSD** ємністю від **256 ГБ** значно прискорює завантаження системи, компіляцію проектів та доступ до файлів.
- **Монітор з роздільною здатністю** не менше **1920x1080** пікселів забезпечує комфортну роботу з інтерфейсом Visual Studio та одночасний перегляд коду і дизайнера форм.
- **Стабільне інтернет-з'єднання** необхідне для взаємодії з API штучного інтелекту та завантаження необхідних бібліотек через NuGet.

Технічне забезпечення:

- **Операційна система Windows 10** або новіша версія забезпечує повну сумісність з інструментами розробки та підтримку сучасних технологій.
- Для виконання розроблених застосунків необхідна **платформа .NET версії 6.0** або новіше. Користувачам достатньо встановити .NET Desktop Runtime, тоді як для розробки потрібен повний .NET SDK.
- Рекомендується **MySQL Server версії 8.x** для надійного зберігання та обробки інформації.
- Для з'єднання C# програм з MySQL необхідний **MySQL Connector/.NET**, який встановлюється через менеджер пакетів NuGet.

Програмне забезпечення:

- **Visual Studio 2022** є основним середовищем для розробки Windows Forms застосунків на C#. Програма включає компілятор, налагоджувач, редактор коду з автодоповненням IntelliSense та вбудований Windows Forms Designer. Останній дозволяє створювати користувацькі інтерфейси візуально, перетягуючи елементи управління на форму та налаштовуючи їх властивості без написання коду.

- *Менеджер пакетів NuGet*, інтегрований у Visual Studio, спрощує встановлення необхідних бібліотек, таких як Anthropic.SDK для роботи з Claude, OpenAI SDK для GPT моделей та OxyPlot для візуалізації даних.
- *Система контролю версій Git* допомагає відстежувати зміни у коді та співпрацювати з іншими розробниками.
- *MySQL Workbench* є комплексним інструментом для проєктування структури бази даних, виконання SQL запитів та адміністрування сервера. Програма надає візуальний конструктор для створення таблиць та зв'язків між ними, редактор запитів з підсвічуванням синтаксису та засоби моніторингу продуктивності.

4.6 Особливості реалізації інтелектуальної системи "Figure Skating" на основі NLP та LLMs

Реалізація програмного застосунку розпочинається зі створення нового пункту меню – HELPER.

При натисканні на нього відкривається відповідний UserControl в центральній частині інтерфейсу програми.

Відкриття UserControl для пункту меню HELPER відбувається наступним чином:

```
private void btn_helper_Click(object sender, EventArgs e)
{
    SidePanel.Height = btn_helper.Height;
    SidePanel.Top = btn_helper.Top;
    sixthControl11.BringToFront();
}
```

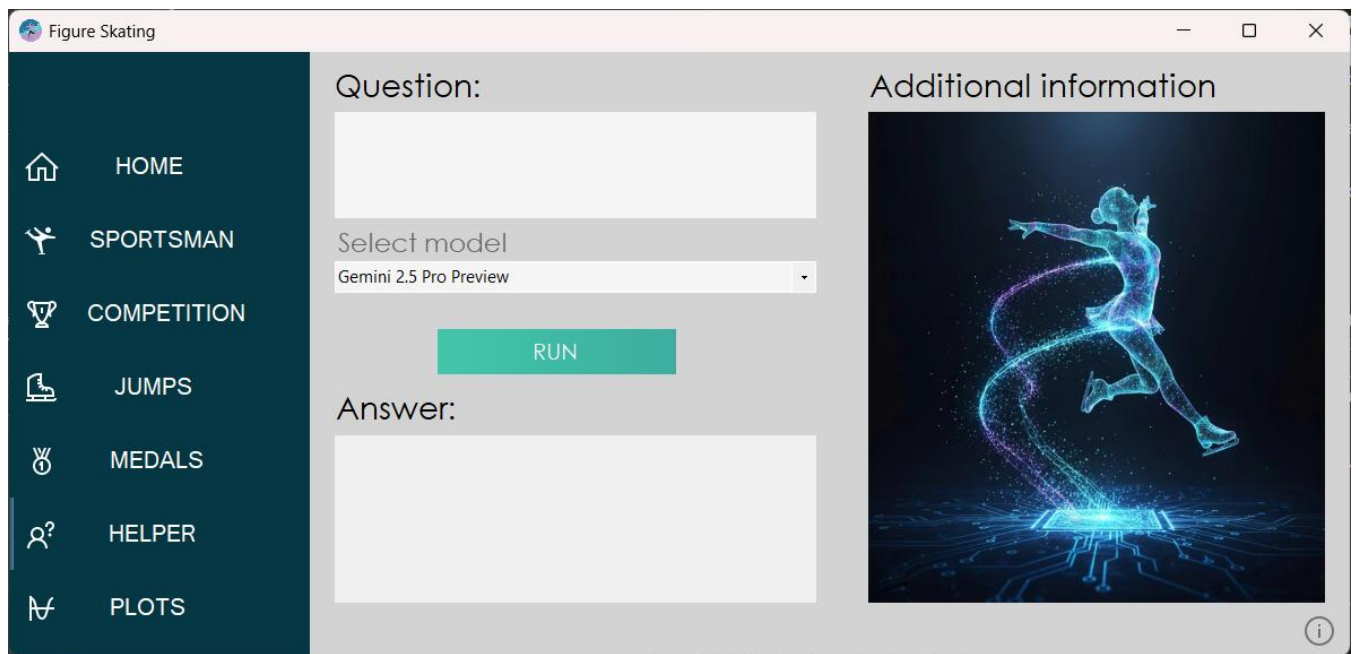


Рисунок 4.1 – Вкладка HELPER

HELPER – вкладка (див. рис. 4.1), де користувачі пишуть запитання, які автоматично перетворюються в SQL-запити за допомогою обраної мовної моделі. Потім запити виконуються над базою даних, результати зберігаються у вигляді JSON-об'єктів, і додатково форматуються мовною моделлю для виводу у вікно Answer.

Для організації інтерфейсу користувача було використано три richTextBox. Перший – для запиту користувача, другий – для відповіді і останній – для відображення інформації про метрики запиту (токени, вартість тощо).

В елементі comboBox1 можна обрати бажану мовну модель для виконання запиту (див. рис. 4.2).

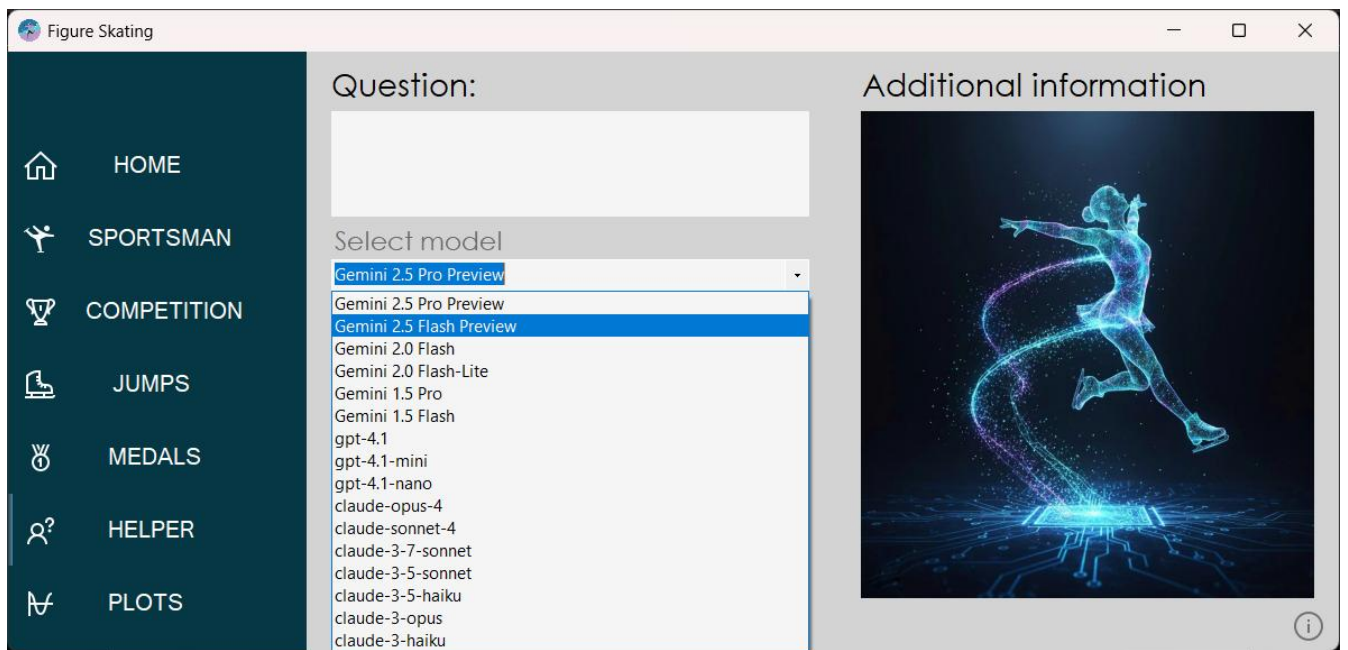


Рисунок 4.2 – Елемент comboBox

Після натиснення кнопки btn_run, під елементом comboBox буде відображатися процес виконання запиту (див. рис. 4.3).

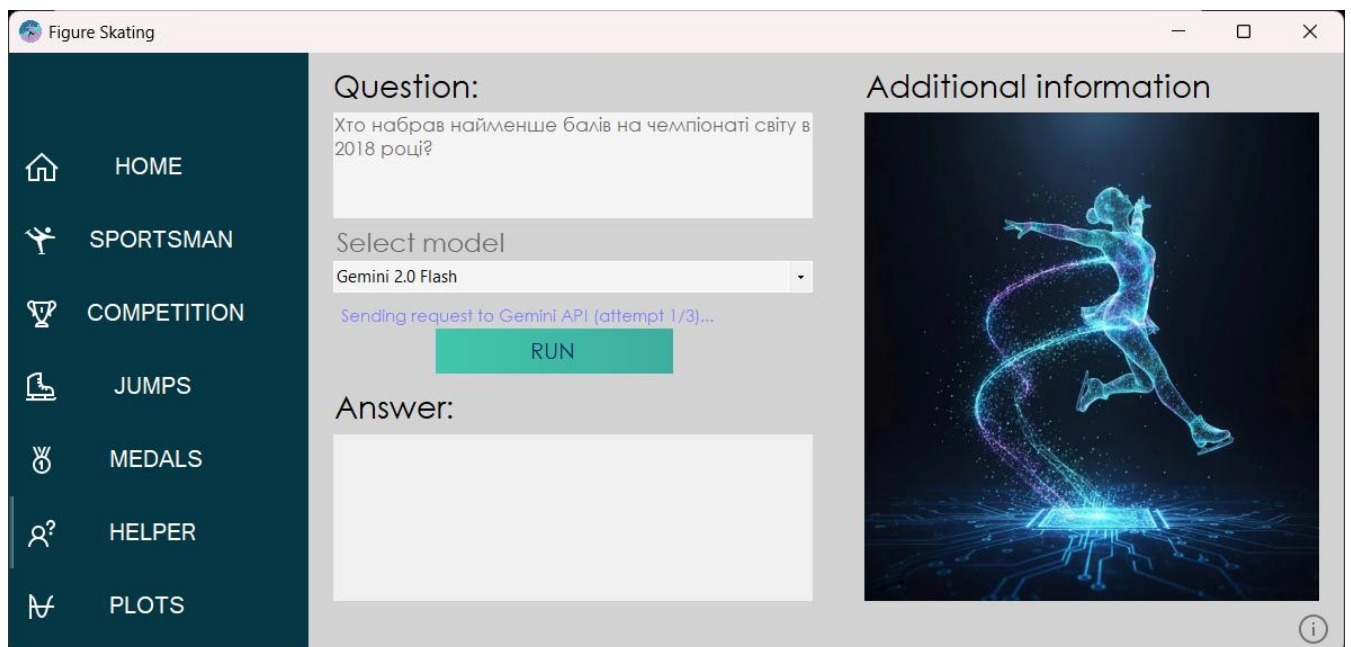


Рисунок 4.3 – Виконання запиту користувача

При натисненні на кнопку додаткова інформація, з'являється richTextBox3 (див. рис. 4.4) з даними про метрики запиту та згенерована SQL-команда.

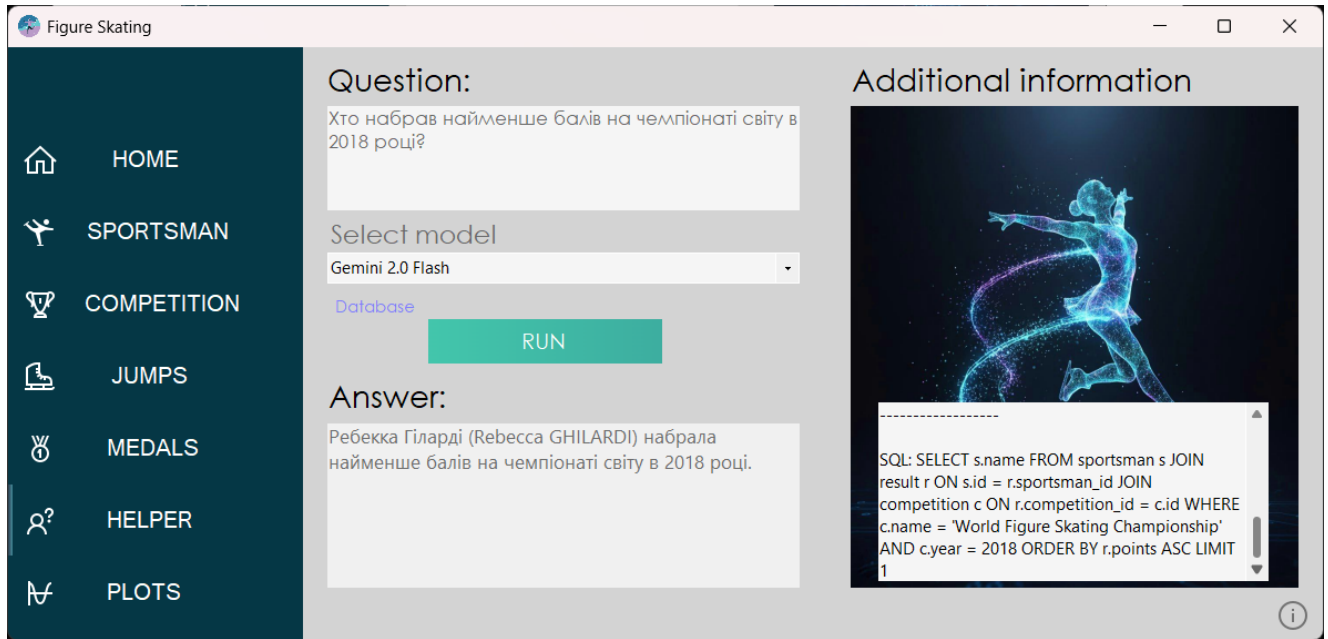


Рисунок 4.4 – Інформація про метрики запиту

Клас `AIAPIClient` – центральний клас для взаємодії з різними API штучного інтелекту (Gemini, Claude, GPT).

Містить такі основні методи:

GeminiGetSQLBasedOnUserQuestionAsync – виконує перший запит до Gemini API. Визначає джерело відповіді на запитання користувача: база даних (генерує SQL-запит) або Інтернет (загальні знання). Приймає такі вхідні дані як, `userQuestion` – запитання користувача та `model` – назва моделі Gemini.

Працює наступним чином:

1. Формує системний промпт, який містить:
 - Детальний опис схеми бази даних (таблиці `category`, `competition`, `result`, `sportsman`).
 - Опис полів та зв'язків між таблицями.
 - Інструкції щодо формату відповіді (JSON з ключами "Source" та "Result").
 - Приклади очікуваних відповідей.

2. Додає запитання користувача до історії розмови.

```
// Додає 1-ий запис з роллю user (містить системний промпт)
var prePrompt = "Ось схема таблиць в базі даних, включаючи первинні і зовнішні
ключі.\r\nTable: category\r\n\r\nCREATE TABLE...";

conversationHistory.Add(new
{
    role = "user",
    parts = new[] { new { text = prePrompt } }
});

// Додає 2-ий запис з роллю user (містить запитання користувача)
string finalPrompt = $"User: {userQuestion}";
conversationHistory.Add(new
{
    role = "user",
    parts = new[] { new { text = finalPrompt } }
});
```

3. Відправляє запит до Gemini API через HTTP POST.

```
var _apiUrl = $"https://generativelanguage.googleapis.com/v1beta/models/" +
models[model] + ":generateContent?key=AIzaSyDF12vFguqieX2UAMDHNaAbH6257xiJrfQ";
// Зберігаємо історію розмови та налаштування в єдиний анонімний об'єкт
var requestBody = new { contents = conversationHistory, tools = new {
googleSearch = new { } } };
var jsonRequest = JsonSerializer.Serialize(requestBody);
var content = new StringContent(jsonRequest, Encoding.UTF8,
"application/json");
var response = await _httpClient.PostAsync(_apiUrl, content);
```

4. Реалізує логіку повторних спроб (до 3 разів) у разі невдачі.
5. Парсить JSON-відповідь від Gemini.

```
// Парсить отриману JSON-відповідь від API (витягує текстове поле)
using var doc = JsonDocument.Parse(jsonResponse);
```

6. Очікувана відповідь – JSON-об'єкт з двома полями:

- "Source" – джерело інформації ("Database" або "Internet").
- "Result" – SQL-запит (якщо Source="Database") або текстова відповідь (якщо Source="Internet").

7. Вихідні дані – кортеж, перший параметр якого це JSON-рядок від Gemini, а другий – метрики запити.

GeminiGetAiResponseFormatterAsync – другий запит до Gemini API. Бере результат виконання SQL-запиту з БД у форматі JSON або текстову відповідь від першого виклику Gemini, якщо джерелом був "Internet" та оригінальне запитання користувача, а потім форматує їх у зрозумілу та повну текстову відповідь для кінцевого користувача. Приймає такі вхідні дані як, `userQuestion` – оригінальне запитання користувача, `jsonData` – дані у форматі JSON (результат SQL-запиту або текстова відповідь від першого виклику) та `model` – назва моделі Gemini.

Працює наступним чином:

1. Формує промпт з інструкціями для Gemini:

- Діяти як мовний асистент.
- Проаналізувати питання користувача.
- Витягти релевантні дані з JSON.
- Сформулювати лаконічну відповідь українською мовою.

2. Підставляє `userQuestion` та `jsonData` у промпт.

```
var finalPrompt = "Завдання:\n\nТи – мовний асистент. Твоя задача – взяти  
питання користувача та JSON-дані, які містять...";  
finalPrompt = finalPrompt.Replace("{user_question}",  
userQuestion).Replace("{json_data}", jsonData);  
// Додає цей finalPrompt як запис користувача до історії  
conversationHistory.Add(new  
{  
role = "user",  
parts = new[] { new { text = finalPrompt } } });
```

3. Відправляє запит до Gemini API.

```
var requestBody = new { contents = conversationHistory }; // Ств. об'єкта  
var jsonRequest = JsonSerializer.Serialize(requestBody); // Серіалізація в JSON  
// Формує JSON-тіло запиту (пакування для HTTP-запиту)  
var content = new StringContent(jsonRequest, Encoding.UTF8,  
"application/json");  
// Надсилає POST-запит до Gemini  
var response = await _httpClient.PostAsync(_apiUrl, content);
```

4. Реалізує логіку повторних спроб (до 3 разів).
5. Парсить текстову відповідь.

```
var jsonResponse = await response.Content.ReadAsStringAsync();
using var doc = JsonDocument.Parse(jsonResponse);
// Парсить отриману відповідь, витягуючи форматований текст
var responseText = doc.RootElement
    .GetProperty("candidates")[0]
    .GetProperty("content")
    .GetProperty("parts")[0]
    .GetProperty("text")
    .GetString();
```

6. Вихідні дані – форматowana текстова відповідь українською мовою, готова для відображення користувачеві.

ClaudeGetSQLBasedOnUserQuestionAsync / ClaudeGetAiResponseFormatterAsync

– аналогічні методи для роботи з Anthropic Claude API. Використовують бібліотеку AnthropicClient та MessageParameters.

GptGetSQLBasedOnUserQuestionAsync / GptGetAiResponseFormatterAsync –

аналогічні методи для роботи з OpenAI GPT API. Використовують бібліотеку ChatClient та ChatMessage.

PLOTS – вкладка, що відображає аналітичні графіки на основі даних про змагання з фігурного катання, використовуючи бібліотеку OxyPlot.

Перший графік (див. рис. 4.5) показує розподіл перемог за країнами (топ-10) та категоріями. Стовпчики складаються з сегментів різних кольорів для кожної категорії.

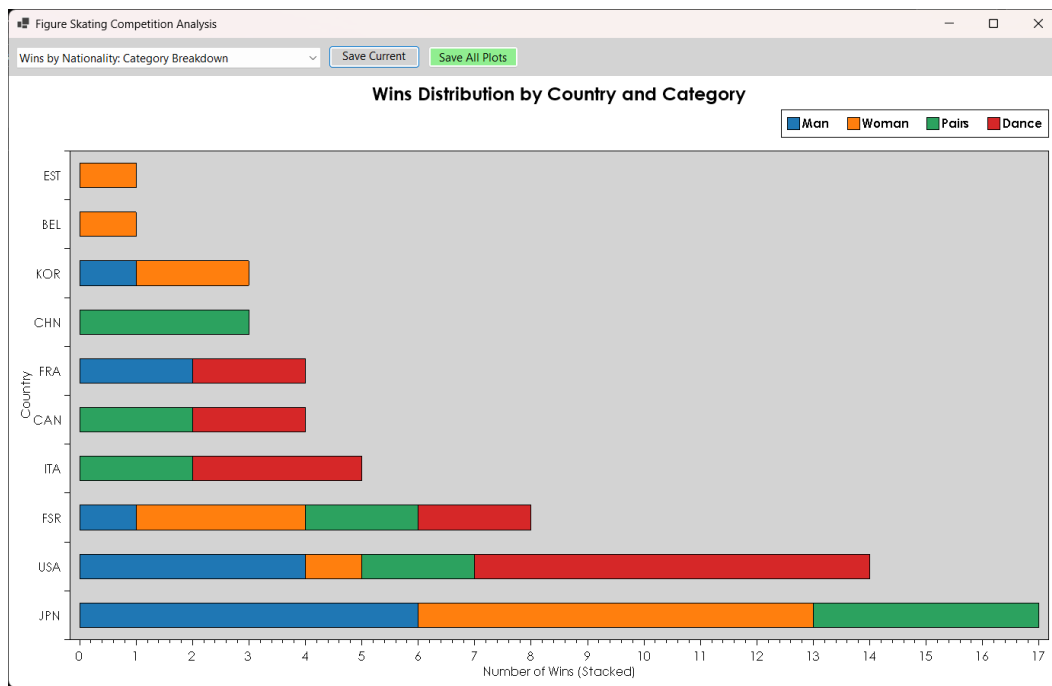


Рисунок 4.5 – Розподіл перемог за країнами

Наступний графік – стовпчикова діаграма топ-15 переможців (див. рис. 4.6). Показує спортсменів з найбільшою кількістю перемог, для спортсменів з парних категорій відображає обидва імені.

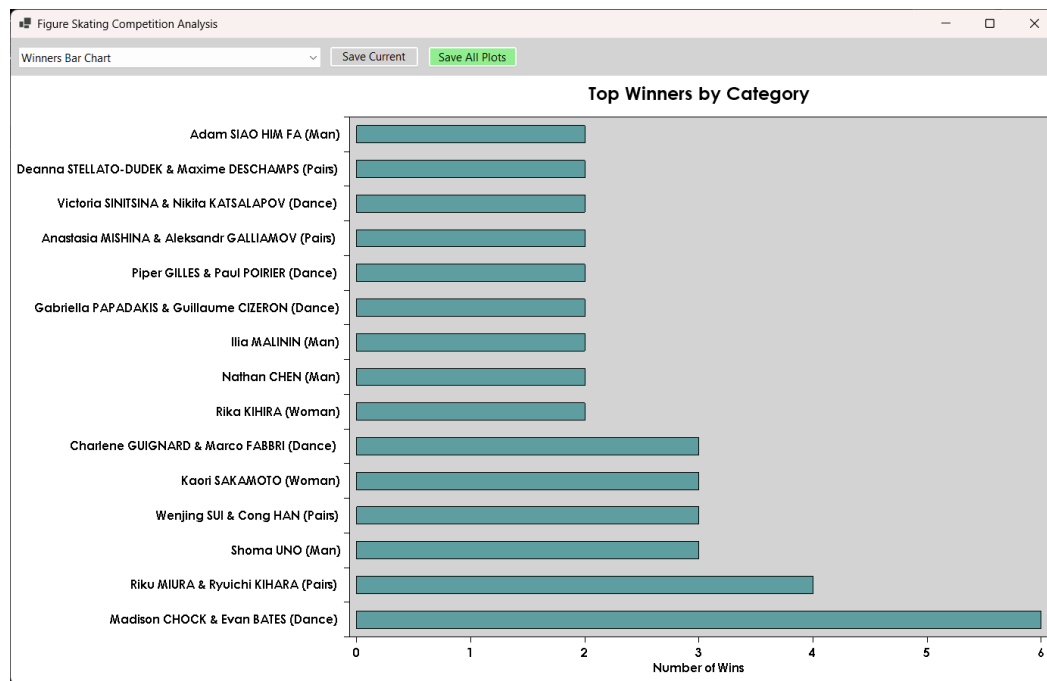


Рисунок 4.6 – Топ-15 переможців

Третій – лінійний графік прогресії балів (див. рис. 4.7). Показує зміну середніх балів за роками для кожної категорії. Кожна категорія – окрема кольорова лінія.

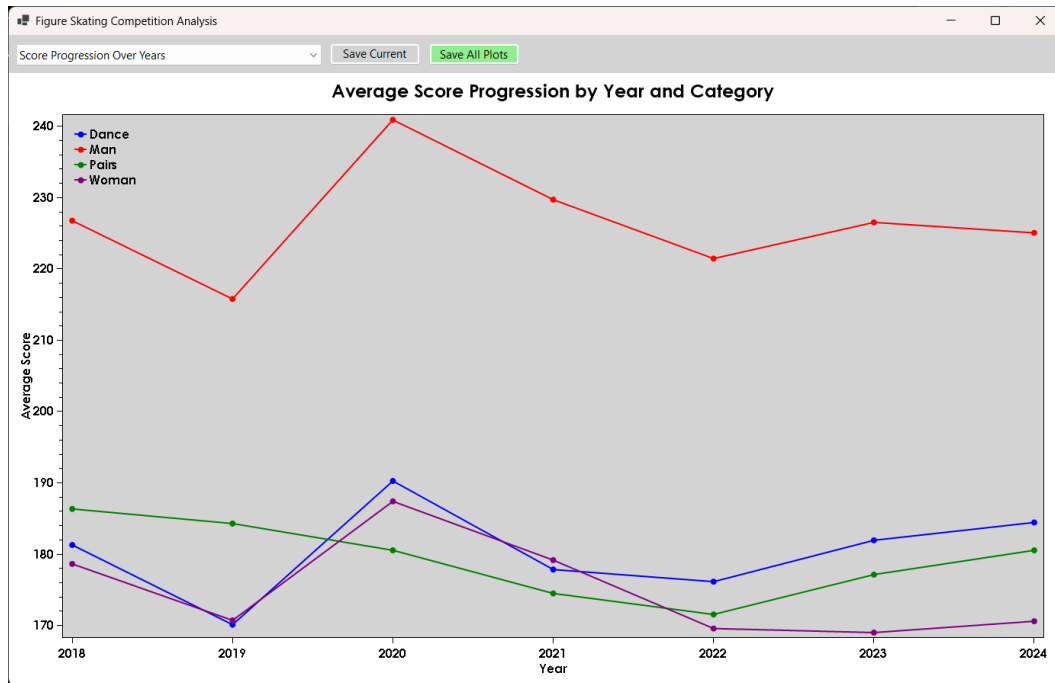


Рисунок 4.7 – Динаміка середніх балів

Наступний відображає горизонтальну стовпчикову діаграму (див. рис. 4.8), що порівнює найкращий бал (Best Score) та середній бал (Average Score) топ-15 спортсменів.

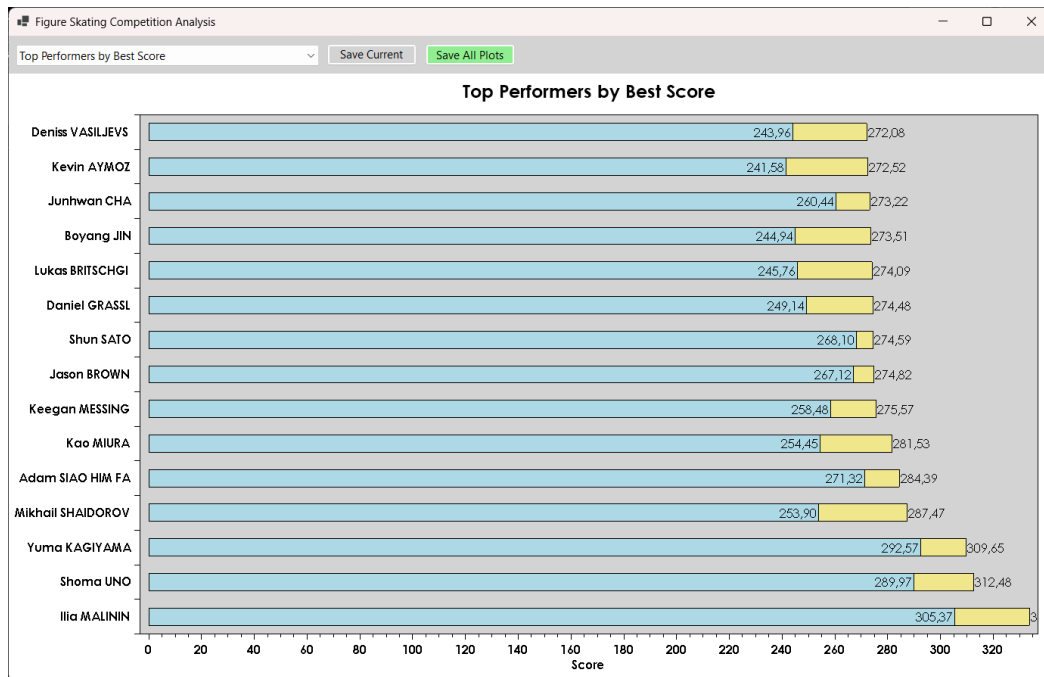


Рисунок 4.8 – Порівняння середніх та найкращих результатів

Остання діаграма з областями (див. рис. 4.9) показує кількість учасників за роками для кожної категорії. Кожна категорія має свій колір: Man (синій), Woman (рожевий), Dance (зелений) та Pairs (жовтий).

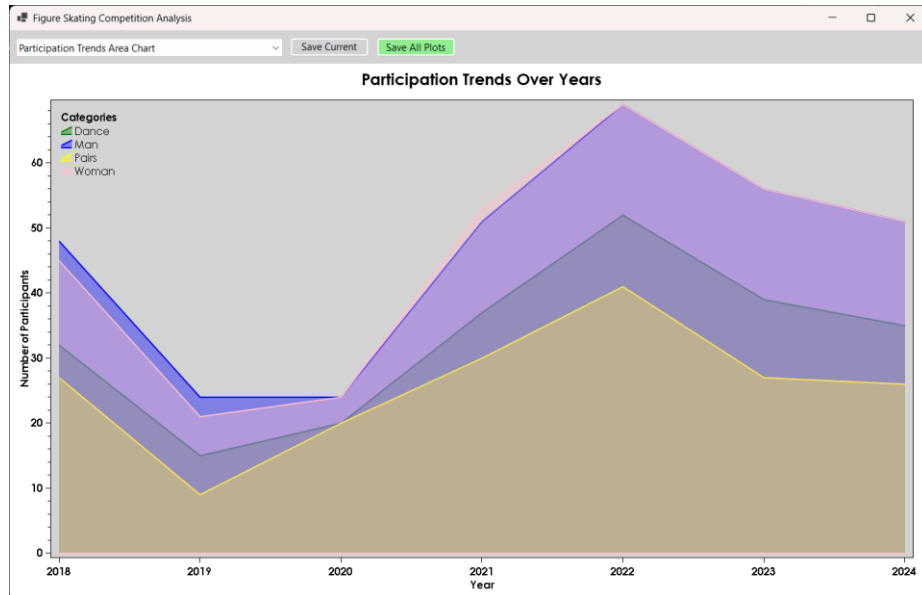


Рисунок 4.9 – Щорічна кількість учасників

Також є можливість збереження поточного графіка чи усіх загалом (див. рис. 4.10).

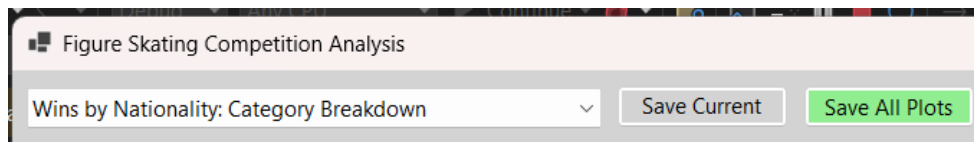


Рисунок 4.10 – Верхня панель вкладки

Висновки до розділу

У цьому розділі описано практичну реалізацію інтелектуальної системи "Figure Skating", яка використовує NLP та великі мовні моделі для автоматичної генерації SQL-запитів.

Розроблено модуль HELPER, що забезпечує взаємодію користувача з трьома провідними AI-платформами (Gemini, Claude, GPT) через уніфікований інтерфейс.

Реалізовано дворівневу архітектуру обробки запитів: перший рівень визначає джерело даних та генерує SQL-запит або текстову відповідь, другий – форматує

результати природною українською мовою.

Створено модуль візуалізації PLOTS на основі бібліотеки OxyPlot, який генерує п'ять типів аналітичних графіків для комплексного аналізу даних про змагання з фігурного катання.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Ідея стартап-проєкту полягає у створенні інтелектуальної інформаційно-пошукової системи "Figure Skating", яка використовує технології обробки природної мови (NLP) та великі мовні моделі (Large Language Models) для автоматичного формування SQL-запитів на основі запитань користувача українською мовою.

Система дозволяє користувачам отримувати інформацію про спортсменів, змагання, результати та статистику фігурного катання без необхідності знати мову SQL. Користувач формулює запитання природною мовою, а система автоматично перетворює його в SQL-запит, виконує його над базою даних та надає зрозумілу відповідь українською мовою.

Можливі напрямки застосування:

- спортивні організації та федерації фігурного катання для аналізу результатів змагань;
- тренери та спортсмени для вивчення статистики конкурентів;
- журналісти та медіа для швидкого доступу до спортивних даних;
- дослідники та аналітики для статистичних досліджень у спорті;
- вболівальники фігурного катання для отримання детальної інформації.

Основні вигоди для користувачів:

- доступ до інформації без знання SQL або складних пошукових інструментів;
- підтримка української мови для запитів та відповідей;
- можливість вибору між різними мовними моделями (Gemini, Claude, GPT);
- візуалізація даних через інтерактивні графіки та діаграми;
- інтеграція з зовнішніми джерелами (Wikipedia) для розширеної інформації;
- збереження метрик запитів для аналізу ефективності різних моделей.

Більшість спортивних баз даних мають традиційний інтерфейс пошуку з фільтрами та формами. Інтелектуальна система "Figure Skating" пропонує інший підхід

– діалог з базою даних природною мовою. Ключові відмінності включають інтеграцію трьох провідних мовних моделей AI з можливістю порівняння їх ефективності, автоматичне визначення джерела інформації (база даних чи інтернет) та комплексну аналітику з відображенням метрик запитів (токени, вартість, час виконання).

5.2 Аналіз технологічних можливостей реалізації ідей проєкту

Для реалізації проєкту використовуються наступні технології:

Windows Forms (C#) – забезпечує зручний десктопний інтерфейс користувача з можливістю інтеграції різних компонентів (DataGridView, WebView, RichTextBox). Використання .NET Framework дозволяє створювати стабільні та продуктивні застосунки для Windows.

База даних MySQL – надійна реляційна СУБД для зберігання інформації про спортсменів, змагання, результати та категорії. Підтримує складні SQL-запити та забезпечує високу швидкість обробки даних.

AI/ML інфраструктура – інтеграція з трьома провідними платформами штучного інтелекту через їх публічні API – Google Gemini API, Anthropic Claude API та OpenAI GPT API. Ці технології вже існують та доступні для використання.

HTTP-комунікація створюється за допомогою класу HttpClient, що взаємодіє з API мовних моделей, також є підтримка асинхронних запитів.

Бібліотека OxyPlot використовується для створення інтерактивних графіків та діаграм. Підтримує різні типи візуалізацій (стовпчикові, лінійні, кругові діаграми).

Бібліотека GMap.NET слугує для відображення карт з маркерами країн-учасниць змагань.

Технологічна реалізація проєкту є цілком можливою завдяки наступним факторам. По-перше, всі API мовних моделей (Google Gemini, Anthropic Claude, OpenAI GPT) мають детальну документацію та готові SDK. Платформа .NET Framework та C# забезпечують потужну екосистему бібліотек для роботи з базами даних, HTTP-запитами та візуалізацією даних. СУБД MySQL є відкритою системою з безкоштовною ліцензією.

Всі використані бібліотеки (OxyPlot, GMap.NET, HttpClient) мають ліцензії open-source або є безкоштовними.

Технології є доступними для реалізації проєкту з наступних причин. Розробка виконується на стандартному персональному комп'ютері з ОС Windows без потреби у спеціалізованому обладнанні. Visual Studio Community Edition надається безкоштовно для індивідуальних розробників. API мовних моделей пропонують платні тарифи доступні за помірну плату.

5.3 Аналіз ринкових можливостей запуску стартап-проєкту

Ринок спортивної аналітики активно розвивається, зокрема, фігурне катання має міжнародну аудиторію, федерації, тренерів та вболівальників, які потребують доступ до детальної статистики.

Потенційні клієнти включають національні федерації фігурного катання (понад 50 країн є членами ISU), спортивні медіа та видання, тренерські штаби та аналітичні центри. Українська версія системи може бути особливо затребуваною в локальному спортивному середовищі.

Останнім часом спостерігається тренд на цифровізацію спорту та використання AI для аналізу даних. Зростає попит на інтуїтивні інтерфейси для роботи з великими обсягами спортивної інформації.

Потенційні групи клієнтів:

- Професійні користувачі (спортивні організації, федерації) – потребують швидкого доступу до статистики для прийняття рішень, готові платити за професійні інструменти. Основні вимоги – висока точність даних, можливість експорту звітів, API для інтеграції з існуючими системами.
- Тренери та спортсмени – використовують дані для аналізу конкурентів та власного прогресу. Необхідний зручний інтерфейс, візуалізація трендів, порівняльний аналіз.

- Медіа та журналісти – потребують швидкий доступ до актуальної інформації для статей та репортажів. Головне це можливість швидкого пошуку фактів, експорт даних у зручних форматах.
- Вболівальники – цікавляться статистикою улюблених спортсменів, не мають технічних навичок. Необхідний простий інтерфейс, безкоштовний або доступний за низькою ціною базовий функціонал.

Зростаючий інтерес до використання AI у спорті, відсутність україномовних рішень в цій ніші та зростання цифрової грамотності серед спортивних організацій є ключовими факторами, що сприяють ринковому впровадженню.

Фактори, що перешкоджають – це вартість використання платних API мовних моделей, необхідність постійного оновлення бази даних та конкуренція з міжнародними платформами спортивної статистики.

Сильні сторони проєкту – це інноваційний підхід використання NLP для запитів, підтримка української мови, можливість порівняння різних AI-моделей, комплексна візуалізація даних та відкритість для розширення на інші види спорту.

Слабкі сторони – залежність від зовнішніх API, обмежена база даних (потрібне постійне наповнення), десктопне застосування (немає веб-версії) та потенційні витрати на токени AI-моделей.

Висновки до розділу

Проведений аналіз показує, що система "Figure Skating" має значний потенціал для ринкової комерціалізації. Існує попит з боку різних груп користувачів – від професійних спортивних організацій до вболівальників. Ринок спортивної аналітики демонструє стійке зростання, а впровадження AI-технологій у спорт є актуальним трендом.

Технологічна реалізація проєкту є цілком можливою з використанням доступних інструментів. Всі необхідні компоненти (API мовних моделей, бібліотеки візуалізації, СУБД) широко використовуються.

Проект має конкурентні переваги: підтримку української мови, можливість порівняння різних AI-моделей, комплексну візуалізацію даних. Основні ризики пов'язані з вартістю використання зовнішніх API та необхідністю постійного оновлення даних.

ВИСНОВКИ

У результаті виконання магістерської кваліфікаційної роботи створено інтелектуальну інформаційно-пошукову систему "Figure Skating" з використанням технологій NLP та великих мовних моделей (LLMs) для автоматичної генерації SQL-запитів українською мовою.

В рамках роботи проаналізовано підходи до реалізації задачі Text-to-SQL та розроблено методику проєктування промптів, яка поєднує стратегії Few-shot та Chain of Thought із детальним описом схеми бази даних. Реалізовано систему двоетапного промптування для генерації коректних SQL-запитів та форматування відповідей.

Проведене експериментальне дослідження шести мовних моделей за метриками Exact Matching (EM), Execution Accuracy (EX) та Valid Efficiency Score (VES) показало, що модель GPT-4.1 демонструє найкращі показники точності (EM = 0,40; VES = 0,4000), тоді як Gemini 2.5 Flash Preview виявилася найшвидшою та найекономічнішою.

Розроблений програмний застосунок на базі Windows Forms та C# забезпечує інтеграцію з платформами штучного інтелекту та візуалізацію даних через інтерактивні графіки.

Практична цінність роботи полягає у можливості використання системи спортивними організаціями, тренерами та журналістами для оперативного аналізу змагань і стратегічного планування. Перспективним напрямком подальших досліджень є оптимізація промптів для складних запитів та адаптація методики для інших предметних областей.

Отже, всі поставлені завдання виконано в повному обсязі. Готовий програмний продукт дозволяє користувачам без технічних знань SQL отримувати доступ до спортивної аналітики через запити, що підтверджує ефективність та практичне використання методів NLP у сучасних спортивних системах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jurafsky D., Martin J. H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Draft of 3rd edition. 2025. URL: <https://web.stanford.edu/~jurafsky/slp3/> (дата звернення: 03.12.2025).
2. Zhu X., Li Q., Cui L., Liu Y. Large Language Model Enhanced Text-to-SQL Generation: A Survey. arXiv preprint arXiv:2410.06011, 2024. URL: <https://arxiv.org/abs/2410.06011> (дата звернення: 03.12.2025).
3. Stryker C. What is a Large Language Model (LLM)?. IBM Think, 2025. URL: <https://www.ibm.com/think/topics/large-language-models> (дата звернення: 03.12.2025).
4. White J., Fu Q., Hays S., et al. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. arXiv preprint arXiv:2302.11382. 2023. URL: <https://arxiv.org/abs/2302.11382> (дата звернення: 03.12.2025).
5. Hu E. J., Shen Y., Wallis P., et al. LoRA: Low-Rank Adaptation of Large Language Models. Proceedings of ICLR, 2022.
6. Vaswani A., Shazeer N., Parmar N., et al. Attention Is All You Need. Proceedings of NeurIPS, 2017, pp. 5998-6008.
7. Wang L., Ma C., Feng X., et al. A Survey on Large Language Model based Autonomous Agents. arXiv preprint arXiv:2308.11432. 2023. URL: <https://arxiv.org/abs/2308.11432> (дата звернення: 03.12.2025).
8. Brown T. B., Mann B., Ryder N., et al. Language Models are Few-Shot Learners. Proceedings of NeurIPS, 2020, vol. 33, pp. 1877-1901.
9. Liu J., Shen D., Zhang Y., et al. What Makes Good In-Context Examples for GPT-3?. Proceedings of DeeLIO Workshop at ACL, 2022, pp. 100-114.
10. Wei J., Wang X., Schuurmans D., et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Proceedings of NeurIPS, 2022, vol. 35, pp. 24824-24837.

- 11.Radford A., Wu J., Child R., et al. Language Models are Unsupervised Multitask Learners. OpenAI Technical Report, 2019.
- 12.Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of NAACL-HLT, 2019, pp. 4171-4186.
- 13.Howard J., Ruder S. Universal Language Model Fine-tuning for Text Classification. Proceedings of ACL, 2018, pp. 328-339.
- 14.Skeet J. C# in Depth. 4th ed. Shelter Island : Manning Publications, 2019. 528 p.
- 15.Albahari J., Albahari B. C# 12 in a Nutshell: The Definitive Reference. Sebastopol : O'Reilly Media, 2023. 1062 p.
- 16.Troelsen A., Japikse P. Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming. New York : Apress, 2022. 1353 p.
- 17.Griffiths I. Programming C# 8.0: Build Windows, Web, and Desktop Applications. Sebastopol : O'Reilly, 2020. 756 p.
- 18.Anthropic API Documentation. Anthropic, 2024. URL: <https://docs.anthropic.com/> (дата звернення: 03.12.2025).
- 19.OxyPlot Documentation. OxyPlot Contributors, 2024. URL: <https://oxyplot.readthedocs.io/> (дата звернення: 03.12.2025).

ДОДАТКИ

ДОДАТОК А

Програмний код

```
using Anthropic.SDK.Messaging;
using Anthropic.SDK;
using System.Text;
using System.Text.Json;
using Message = Anthropic.SDK.Messaging.Message;
using OpenAI.Chat;
using System.Diagnostics;
namespace Services
{
    public class AIApiClient
    {
        private readonly HttpClient _httpClient; // Викор. для виконання HTTP-запитів до
API
        private Label _logLabel;
        private IDictionary<string, string> models = new Dictionary<string, string>()
        {
            {"Gemini 2.5 Pro Preview", "gemini-2.5-pro-preview-05-06" },
            {"Gemini 2.5 Flash Preview", "gemini-flash-latest" },
            {"Gemini 2.0 Flash", "gemini-2.0-flash" },
            {"Gemini 2.0 Flash-Lite", "gemini-2.0-flash-lite" },
            {"Gemini 1.5 Pro", "gemini-1.5-pro" },
            {"Gemini 1.5 Flash", "gemini-1.5-flash" },
            {"gpt-4.1", "gpt-4.1"},
            {"gpt-4.1-mini", "gpt-4.1-mini"},
            {"gpt-4.1-nano", "gpt-4.1-nano"},
            {"claude-opus-4", "claude-opus-4-20250514" },
            {"claude-sonnet-4", "claude-sonnet-4-20250514" },
        }
    }
}
```

```

// Anthropic Claude 3.7 (2025-02-19)
{ "claude-3-7-sonnet", "claude-3-7-sonnet-20250219" },
// Anthropic Claude 3.5 (2024-10-22)
{ "claude-3-5-sonnet", "claude-3-5-sonnet-20241022" },
{ "claude-3-5-haiku", "claude-3-5-haiku-20241022" },
// Anthropic Claude 3.0 (2024-02-29 / 2024-03-07)
{ "claude-3-opus", "claude-3-opus-20240229" },
{ "claude-3-haiku", "claude-3-haiku-20240307" }
};

private readonly Dictionary<string, (decimal inputPrice, decimal outputPrice)>
ModelPricing = new()
{
    // OpenAI GPT-4.1 series (from your original list - these might be custom
names/versions)
    { "gpt-4.1", (2.00m, 8.00m) }, // Per 1M tokens
input/output
    { "gpt-4.1-mini", (0.40m, 1.60m) },
    { "gpt-4.1-nano", (0.10m, 0.40m) },
    // OpenAI GPT-4o and other GPT models (current public models, prices per 1M
tokens)
    { "gpt-4o", (2.50m, 10.00m) },
    { "gpt-4o-mini", (0.15m, 0.60m) },
    { "gpt-4-turbo", (10.00m, 30.00m) },
    { "gpt-4", (30.00m, 60.00m) },
    { "gpt-3.5-turbo", (0.50m, 1.50m) },
    { "gpt-3.5-turbo-instruct", (1.50m, 2.00m) },
    // Anthropic Claude - Hypothetical & Older (from your original list or prior
additions)
    { "claude-opus-4", (15.00m, 75.00m) },
    { "claude-sonnet-4", (3.00m, 15.00m) },
    { "claude-3-7-sonnet", (3.00m, 15.00m) },
    { "claude-3-5-sonnet", (3.00m, 15.00m) },
    { "claude-3-5-haiku", (0.80m, 4.00m) },

```

```

        // Anthropic Claude 3.5 and 3 (current public models, prices per 1M tokens)
        { "claude-3.5-sonnet",          (3.00m, 15.00m) },
        { "claude-3-opus",            (15.00m, 75.00m) }, // Matches "claude-3-opus-
20240229" from image
        { "claude-3-sonnet",          (3.00m, 15.00m) },
        { "claude-3-haiku",           (0.25m, 1.25m) }, // Matches "claude-3-haiku-
20240307" from image

        // Google Gemini Models
        { "Gemini 2.5 Pro Preview",    (7.00m, 21.00m) },
        { "Gemini 2.5 Flash Preview",  (0.70m, 2.10m) },
        { "Gemini 2.0 Flash",          (0.50m, 1.50m) },
        { "Gemini 2.0 Flash-Lite",     (0.30m, 0.90m) },

        // Gemini 1.5 Series (already added, confirmed from image)
        { "Gemini 1.5 Pro",            (3.50m, 10.50m) }, // For prompts <=128K
tokens; $0.0035/1k input, $0.0105/1k output
        { "Gemini 1.5 Flash",          (0.35m, 1.05m) }, // For prompts <=128K
tokens; $0.0035/1k input, $0.00105/1k output

        // Gemini 1.0 Series
        { "gemini-1.0-pro",            (0.50m, 1.50m) } // $0.0005/1k input,
$0.0015/1k output
    };

    public AIApiClient(Label logLabel = null)
    {
        _httpClient = new HttpClient();
        _logLabel = logLabel;
    }

    private void UpdateLog(string message)
    {
        if (_logLabel != null && _logLabel.InvokeRequired)
        {
            _logLabel.Invoke(new Action(() => _logLabel.Text = message));
        }
        else if (_logLabel != null)

```

```

    {
        _logLabel.Text = message;
    }
}

// 1-ий запит до Gemini API

// Визначає чи можна відповісти на запитання користувача, використовуючи дані з
бд і згенерувати відповідний SQL-запит, чи надати відповідь з інтернету

public async Task<(string, string)> GeminiGetSQLBasedOnUserQuestionAsync(string
userQuestion, string model)
{
    var stopwatch = Stopwatch.StartNew();
    var responseBuilder = new StringBuilder();
    DateTime startTime = DateTime.Now;
    // URL для доступу до Gemini API, модель, ключ
    var _apiUrl = $"https://generativelanguage.googleapis.com/v1beta/models/" +
models[model] + ":generateContent?key= ";
    UpdateLog("Preparing request to Gemini API for SQL/Internet analysis...");
    var conversationHistory = new List<object>();
    // Додає 1-ий запис з роллю user (містить системний промпт)
    var prePrompt = "Ось схема таблицок в базі даних, включаючи первинні і
зовнішні ключі.\r\nTable: category\r\n\r\nCREATE TABLE category (\r\n    id INT NOT NULL
PRIMARY KEY,\r\n    name VARCHAR(45) NOT NULL CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci\r\n);\r\n\r\nTable: competition\r\n\r\nCREATE TABLE competition (\r\n
id INT NOT NULL PRIMARY KEY,\r\n    name VARCHAR(45) NOT NULL CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci,\r\n    year INT NOT NULL\r\n);\r\n\r\nTable:
result\r\n\r\nCREATE TABLE result (\r\n    id INT NOT NULL PRIMARY KEY,\r\n
category_id INT NOT NULL,\r\n    competition_id INT NOT NULL,\r\n    sportsman_id INT NOT
NULL,\r\n    sportsman2_id INT NULL, -- Дозволяє NULL значення, використовується для
парних категорій\r\n    fpl INT NOT NULL,\r\n    sp INT NOT NULL,\r\n    fs INT NOT
NULL,\r\n    points FLOAT NOT NULL,\r\n    FOREIGN KEY (category_id) REFERENCES
category(id),\r\n    FOREIGN KEY (competition_id) REFERENCES competition(id),\r\n
FOREIGN KEY (sportsman_id) REFERENCES sportsman(id),\r\n    FOREIGN KEY (sportsman2_id)
REFERENCES sportsman(id) -- Також посилається на sportsman\r\n);\r\n\r\nTable:
sportsman\r\n\r\nCREATE TABLE sportsman (\r\n    id INT NOT NULL PRIMARY KEY,\r\n    name
VARCHAR(45) NOT NULL CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci,\r\n    nationality
VARCHAR(45) NOT NULL CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci\r\n);\r\n\r\nОсь
додаткові відомості про дані в базі даних:\r\n\r\nБаза даних містить результати змагань з
фігурного катання. Вона включає:\r\n\r\nТаблиця 'category' (id, name) – категорії: 'Man',
'Woman', 'Pairs', 'Dance'.\r\n\r\nТаблиця 'competition' (id, name, year) – Змагання з
роками: Four Continents Figure Skating Championship, World Figure Skating Championship,
European Figure Skating Championship.\r\n\r\nТаблиця 'sportsman' (id, name, nationality)

```

- спортсмени, їхня національність вказана в такому форматі (наприклад, JPN, USA, GEO, ITA, FRA).

Таблиця 'result' містить (id, category_id, competition_id, sportsman_id, sportsman2_id, fpl, sp, fs, points) - Результати змагань: fpl (final placement) - фінальне місце. sp (short program score) - оцінка за коротку програму. fs (free skate score) - оцінка за довільну програму. points (total points) - загальний бал.

Інструкції щодо відповіді: Твоє завдання - проаналізувати запит користувача і надати відповідь у форматі єдиного JSON об'єкта. Цей JSON об'єкт повинен мати два ключі: "Source": Значення цього ключа вказує на джерело інформації. "Database": якщо відповідь базується на інформації з наданої схеми бази даних. "Internet": якщо відповідь базується на загальних знаннях, оскільки запит не стосується даних у БД. "Result": Значення цього ключа містить саму відповідь. Якщо "Source" є "Database", то "Result" повинен містити валідний SQL запит у вигляді рядка. SQL запит повинен бути без зайвих слів чи коментарів, лише чистий SQL. Якщо "Source" є "Internet", то "Result" повинен містити текстову відповідь на запит користувача, сформовану на основі твоїх загальних знань. Відповідь має бути інформативною та по суті запиту. Порядок дій: 1. Проаналізуй запит користувача. 2. Визнач, чи стосується запит інформації, яка може бути отримана з наданої схеми бази даних про фігурне катання. 3. *Особлива увага:* Якщо запит є загальним, але *може* бути інтерпретований в контексті даних про фігурне катання, що зберігаються в базі даних (наприклад, "Які є категорії в спорті?", "Хто вигравав змагання?", "Розкажи про спортсменів з Японії"), **вважай його таким, що відноситься до бази даних**. Спробуй сформулювати SQL-запит, який би відповів на це питання в контексті фігурного катання. 4. Якщо запит однозначно стосується інформації, яка може бути отримана з наданої схеми бази даних про фігурне катання, встанови "Source": "Database" і сформулюй відповідний SQL запит для поля "Result". 5. Якщо запит *очевидно* не стосується фігурного катання або інформації, яка може бути в наданій базі даних (наприклад, запити про погоду, політику, інші види спорту, не пов'язані з даними в БД, загальні знання), встанови "Source": "Internet" і сформулюй текстову відповідь для поля "Result". 6. Поверни єдиний JSON об'єкт з цими двома полями. Важливо: Спочатку завжди ретельно аналізуй, чи може запит бути виконаний на основі наданої схеми бази даних про фігурне катання, особливо враховуючи пункт 3 порядку дій. Лише якщо це неможливо, переходь до відповіді з "Source": "Internet". Переконайся, що відповідь є валідним JSON. SQL запити мають бути чистими, без додаткових пояснень чи дужок навколо всього запиту, якщо це не є частиною синтаксису самого SQL запиту.

Приклади:

Користувач: "Виведи всіх спортсменів з Франції"

Штучний інтелект: {"Source": "Database", "Result": "SELECT * FROM sportsman WHERE nationality = 'FRA'"}
Користувач: "Які спортсмени виграли World Figure Skating Championship у 2022 році в категорії 'Man'?"
Штучний інтелект: {"Source": "Database", "Result": "SELECT s.name FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name = 'World Figure Skating Championship' AND c.year = 2022 AND cat.name = 'Man' AND r.fpl = 1"}
Користувач: "Які існують категорії в змаганнях?"
Штучний інтелект: {"Source": "Database", "Result": "SELECT name FROM category"}
Користувач: "Хто з японських спортсменів показував найкращі результати?"
Штучний інтелект: {"Source": "Database", "Result": "SELECT s.name, r.points, c.name AS competition_name, c.year FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id WHERE s.nationality = 'JPN' ORDER BY r.points DESC LIMIT 10"}
Користувач: "Яка сьогодні погода в Києві?"
Штучний інтелект: {"Source": "Internet", "Result": "Щоб надати вам актуальну інформацію про погоду в Києві, мені потрібен доступ до сервісу погоди в

реальному часі. Будь ласка, перевірте спеціалізований погодні сайт або додаток.\"}\r\n\r\nКористувач: \ "Хто президент США?\ " \r\nШтучний інтелект: {"Source\": \ "Internet\ ", \ "Result\ ": \ "Станом на мої останні оновлення знань, президентом Сполучених Штатів Америки є Джо Байден. Для найактуальнішої інформації, будь ласка, перевірте надійне джерело новин.\"}\r\n\r\nКористувач: \ "Розкажи про історію Олімпійських ігор.\ " \r\nШтучний інтелект: {"Source\": \ "Internet\ ", \ "Result\ ": \ "Олімпійські ігри мають багату історію, що сягає Стародавньої Греції, де перші відомі ігри відбулися в Олімпії у 776 році до н.е. на честь бога Зевса. Вони включали різноманітні атлетичні змагання. Сучасні Олімпійські ігри були відроджені наприкінці XIX століття П'єром де Кубертеном, і перші з них відбулися в Афінах у 1896 році. З того часу вони стали головною міжнародною спортивною подією, що проводиться кожні чотири роки (літні та зимові ігри чергуються).\"}";

```
conversationHistory.Add(new
{
    role = "user",
    parts = new[] { new { text = prePrompt } }
});
// Додає 2-ий запис з роллю user (містить запитання користувача)
string finalPrompt = $"User: {userQuestion}";
conversationHistory.Add(new
{
    role = "user",
    parts = new[] { new { text = finalPrompt } }
});
UpdateLog("Creating request for Gemini API...");
// Зберігаємо історію розмови та налаштування в єдиний анонімний об'єкт
var requestBody = new { contents = conversationHistory, tools = new {
googleSearch = new { } } };
// Об'єкт перетворюється в JSON-рядок
var jsonRequest = JsonSerializer.Serialize(requestBody);
// Формуємо JSON-тіло запиту для API (вкл. історію розмови та опцію викор.
інструментів (googleSearch, її використання контролюється промптом))
var content = new StringContent(jsonRequest, Encoding.UTF8,
"application/json");

// Логіка повторних спроб
const int maxRetries = 3;
```

```

int currentRetry = 0;
while (currentRetry < maxRetries)
{
    try
    {
        UpdateLog($"Sending request to Gemini API (attempt {currentRetry +
1}/{maxRetries})...");
        // Об'єкт content відправляється на URL-адресу API за допомогою HTTP-
клієнта (POST-запит)
        var response = await _httpClient.PostAsync(_apiUrl, content);
        // Перевірка успішності запиту
        if (response.IsSuccessStatusCode)
        {
            UpdateLog("Gemini API responded successfully. Processing
response...");
            // Асинхронно зчитує тіло відповіді від API і зберігає його як єдиний довгий JSON-
рядок
            var jsonResponse = await response.Content.ReadAsStringAsync();
            // Парсить отриману JSON-відповідь від API (витягує текстове поле)
            using var doc = JsonDocument.Parse(jsonResponse);
            // Повертає отриманий текст (який, згідно з промптом - JSON
рядок типу {"Source": "...", "Result": "..."})
            var responseText = doc.RootElement
                .GetProperty("candidates")[0] // Вибираємо першу згенеровану відповідь
                .GetProperty("content") // Містить вміст відповіді
                .GetProperty("parts")[0]
                .GetProperty("text") // Витягуємо згенерований текстовий рядок
                .GetString(); // Перетв. на стандартний рядок C#
            UpdateLog("Analysis complete. Source determined.");
            stopwatch.Stop();
            var allCharactersCount = prePrompt.Length + finalPrompt.Length;
            var metricsInfo =
DisplayOpenAIOrGeminiMetrics(allCharactersCount, stopwatch.Elapsed, startTime,
responseText, model);

```

```

        return (responseText, metricsInfo);
    }
    else
    {
        // Якщо відповідь не успішна, збільшуємо лічильник спроб
        currentRetry++;
        UpdateLog($"Request failed with status code:
{response.StatusCode}. Retrying...");
        if (currentRetry >= maxRetries)
        {
            // Якщо досягнуто максимальну кількість спроб, повертаємо
повідомлення про помилку
            var errorMessage = await
response.Content.ReadAsStringAsync();
            UpdateLog($"Failed after {maxRetries} attempts.");
            return ($"Після {maxRetries} спроб не вдалося отримати
відповідь. Помилка: {errorMessage}", string.Empty);
        }
        // Чекаємо перед наступною спробою (експоненціальне
відтермінування)
        await Task.Delay(TimeSpan.FromSeconds(2));
    }
}
catch (Exception ex)
{
    // Якщо виникло виключення, зб. лічильник спроб
    currentRetry++;
    UpdateLog($"Error occurred: {ex.Message}. Retrying...");
    if (currentRetry >= maxRetries)
    {
        // Якщо досягнуто макс. к-сть спроб, повертаємо повідомлення про
помилку
        UpdateLog($"Failed after {maxRetries} attempts due to
exception.");
    }
}
}

```

```

        return ("Після {maxRetries} спроб виникла помилка:
{ex.Message}", string.Empty);
    }
    // Чекаємо перед наступною спробою (експоненціальне відтермінування)
    await Task.Delay(TimeSpan.FromSeconds(2));
}
}
UpdateLog("Could not get response after maximum retries.");
return ("Не вдалося отримати відповідь після {maxRetries} спроб.",
string.Empty);
// 2-ий запит до Gemini API
// (бере сирі дані (результат SQL-запиту у форматі JSON або почат. текстову
відповідь від першого виклику)
// та оригінальне запитання корист., і перетворює їх на укр. текстову відповідь
для кінцевого користувача)
public async Task<string> GeminiGetAiResponseFormatterAsync(string userQuestion,
string jsonData, string model)
{
    var _apiUrl = $"https://generativelanguage.googleapis.com/v1beta/models/" +
models[model] + ":generateContent?key= ";
    UpdateLog("Preparing response formatting request to Gemini API...");
    var conversationHistory = new List<object>();
    var finalPrompt = "Завдання:\n\nТи – мовний асистент. Твоя задача – взяти
питання користувача та JSON-дані, які містять інформацію для відповіді, і сформулювати
природну, зрозумілу та повну відповідь для користувача українською мовою.\n\nВхідні
дані:\n\nПитання користувача: {user_question}\n\nJSON-дані:
{json_data}\n\nІнструкції:\n\nПроаналізуй питання: Зрозумій, яку саме інформацію шукає
користувач.\n\nВитягни дані з JSON: Уважно вивчи структуру JSON та витягни з нього тільки
ті дані, які безпосередньо потрібні для відповіді на поставлене питання. Зверни увагу на
ключі та значення в JSON. Якщо в JSON є кілька об'єктів або полів, вибери найбільш
релевантні.\n\nСформулюй відповідь:\n\nНапиши відповідь українською мовою.\n\nВідповідь
має бути прямою та чітко відповідати на питання користувача.\n\nВикористай витягнуті з
JSON дані для формування змісту відповіді.\n\nПредстав інформацію у зручному для читання
форматі (наприклад, повне речення або короткий абзац).\n\nНе показуй користувачеві сам
JSON або його сиру структуру. Перетвори дані на природний текст.\n\nЯкщо JSON містить
імена, прізвища чи інші власні назви, спробуй відтворити їх коректно в українському
тексті.\n\nНадай тільки сутність відповіді. Уникай будь-яких вступних фраз (наприклад,
\"Згідно з даними...\", \"Відповідь на ваше питання...\", \"Так, звичайно...\"),
пояснень, підсумкових речень або іншої додаткової інформації, яка не є частиною прямої
відповіді. Твоя відповідь має бути максимально лаконічною, але повною по суті
питання.\n\nПриклад (для ілюстрації):\n\nПитання користувача: \"Хто зайняв останнє місце

```

на чемпіонаті Європи зі спортивних пар у 2022 році?\n\nJSON-дані: [{\"name\": \"Greta CRAFOORD\", \"name1\": \"John CRAFOORD\"}]\n\nОчікувана відповідь: \"Останнє місце на чемпіонаті Європи зі спортивних пар у 2022 році посіли Грета Крафорд (Greta CRAFOORD) та Джон Крафорд (John CRAFOORD).\" (Або схожий лаконічний варіант).\n\nВиконай завдання для наданих питання та JSON.\";

```
finalPrompt = finalPrompt.Replace("{user_question}",
userQuestion).Replace("{json_data}", jsonData);

// Додає цей finalPrompt як запис користувача до історії
conversationHistory.Add(new
{
    role = "user",
    parts = new[] { new { text = finalPrompt } }
});
UpdateLog("Creating formatting request for Gemini API...");
var requestBody = new { contents = conversationHistory }; // Ств. об'єкта
var jsonRequest = JsonSerializer.Serialize(requestBody); // Сериалізація в
JSON

// Формує JSON-тіло запиту (пакування для HTTP-запиту)
var content = new StringContent(jsonRequest, Encoding.UTF8,
"application/json");
// Логіка повторних спроб
const int maxRetries = 3;
int currentRetry = 0;
while (currentRetry < maxRetries)
{
    try
    {
        UpdateLog($"Sending formatting request to Gemini API (attempt
{currentRetry + 1}/{maxRetries})...");
        // Надсилає POST-запит до Gemini
        var response = await _httpClient.PostAsync(_apiUrl, content);
        if (response.IsSuccessStatusCode)
        {
            UpdateLog("Formatting successful. Processing final response...");
            var jsonResponse = await response.Content.ReadAsStringAsync();
```

```

using var doc = JsonDocument.Parse(jsonResponse);
// Парсить отриману відповідь, витягуючи форматований текст
var responseText = doc.RootElement
    .GetProperty("candidates")[0]
    .GetProperty("content")
    .GetProperty("parts")[0]
    .GetProperty("text")
    .GetString();
// Додає відповідь Gemini до історії розмови з рольовою моделлю
conversationHistory.Add(new
{
    role = "model",
    parts = new[] { new { text = responseText } }
});
UpdateLog("Response ready.");
return responseText;
}
else
{
    // Якщо відповідь не успішна, зб. лічильник спроб
    currentRetry++;
    UpdateLog($"Format request failed with status code:
{response.StatusCode}. Retrying...");
    if (currentRetry >= maxRetries)
    {
        // Якщо досягнуто макс. к-сть спроб, повертаємо повідомлення
        про помилку
        var errorMessage = await
response.Content.ReadAsStringAsyncAsync();
        UpdateLog($"Failed after {maxRetries} formatting attempts.");
        return $"Після {maxRetries} спроб не вдалося отримати
відповідь. Помилка: {errorMessage}";
    }
}

```

```

        // Чекаємо перед наступною спробою
        await Task.Delay(TimeSpan.FromSeconds(2));
    }
}
catch (Exception ex)
{
    // Якщо виникло виключення, зб. лічильник спроб
    currentRetry++;
    UpdateLog($"Format error occurred: {ex.Message}. Retrying...");
    if (currentRetry >= maxRetries)
    {
        // Якщо досягнуто макс. к-сть спроб, повертаємо повідомлення про помилку
        UpdateLog($"Failed after {maxRetries} formatting attempts due to exception.");
        return $"Після {maxRetries} спроб виникла помилка: {ex.Message}";
    }
    // Чекаємо перед наступною спробою
    await Task.Delay(TimeSpan.FromSeconds(2));
}
}

UpdateLog("Could not get formatted response after maximum retries.");
return $"Не вдалося отримати відповідь після {maxRetries} спроб.";
}

public async Task<(string, string)> ClaudeGetSQLBasedOnUserQuestionAsync(string
userQuestion, string model)
{
    var stopwatch = Stopwatch.StartNew();
    var responseBuilder = new StringBuilder();
    DateTime startTime = DateTime.Now;
    var apiKey = "";
    UpdateLog("Preparing request to Claude API for SQL/Internet analysis...");
    var client = new AnthropicClient(apiKey);
    var messages = new List<Message>()

```

{

```
new Message(RoleType.User, "Ось схема таблиц в базі даних, включаючи
первинні і зовнішні ключі.\r\nTable: category\r\n\r\nCREATE TABLE category (\r\n id
INT NOT NULL PRIMARY KEY,\r\n name VARCHAR(45) NOT NULL CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci\r\n);\r\n\r\nTable: competition\r\n\r\nCREATE TABLE competition (\r\n
id INT NOT NULL PRIMARY KEY,\r\n name VARCHAR(45) NOT NULL CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci,\r\n year INT NOT NULL\r\n);\r\n\r\nTable:
result\r\n\r\nCREATE TABLE result (\r\n id INT NOT NULL PRIMARY KEY,\r\n
category_id INT NOT NULL,\r\n competition_id INT NOT NULL,\r\n sportsman_id INT NOT
NULL,\r\n sportsman2_id INT NULL, -- Дозволяє NULL значення, використовується для
парних категорій\r\n fpl INT NOT NULL,\r\n sp INT NOT NULL,\r\n fs INT NOT
NULL,\r\n points FLOAT NOT NULL,\r\n FOREIGN KEY (category_id) REFERENCES
category(id),\r\n FOREIGN KEY (competition_id) REFERENCES competition(id),\r\n
FOREIGN KEY (sportsman_id) REFERENCES sportsman(id),\r\n FOREIGN KEY (sportsman2_id)
REFERENCES sportsman(id) -- Також посилається на sportsman\r\n);\r\n\r\nTable:
sportsman\r\n\r\nCREATE TABLE sportsman (\r\n id INT NOT NULL PRIMARY KEY,\r\n name
VARCHAR(45) NOT NULL CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci,\r\n nationality
VARCHAR(45) NOT NULL CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci\r\n);\r\n\r\nОсь
додаткові відомості про дані в базі даних:\r\n\r\nБаза даних містить результати змагань з
фігурного катання. Вона включає:\r\n\r\nТаблиця 'category' (id, name) – категорії: 'Man',
'Woman', 'Pairs', 'Dance'.\r\n\r\nТаблиця 'competition' (id, name, year) – Змагання з
роками: Four Continents Figure Skating Championship, World Figure Skating Championship,
European Figure Skating Championship.\r\n\r\nТаблиця 'sportsman' (id, name, nationality)
– спортсмени, їхня національність вказана в такому форматі (наприклад, JPN, USA, GEO,
ITA, FRA).\r\n\r\nТаблиця 'result' містить (id, category_id, competition_id,
sportsman_id, sportsman2_id, fpl, sp, fs, points) – Результати змагань:\r\n\r\nfpl (final
placement) – фінальне місце.\r\n\r\nsp (short program score) – оцінка за коротку
програму.\r\n\r\nfs (free skate score) – оцінка за довільну програму.\r\n\r\npoints
(total points) – загальний бал.\r\n\r\nІнструкції щодо відповіді:\r\n\r\nТвоє завдання –
проаналізувати запит користувача і надати відповідь у форматі єдиного JSON об'єкта. Цей
JSON об'єкт повинен мати два ключі:\r\n\r\n"Source": Значення цього ключа вказує на
джерело інформації.\r\n\r\n"Database": якщо відповідь базується на інформації з наданої
схеми бази даних.\r\n\r\n"Internet": якщо відповідь базується на загальних знаннях,
оскільки запит не стосується даних у БД.\r\n\r\n"Result": Значення цього ключа містить
саму відповідь.\r\n\r\nЯкщо "Source" є "Database", то "Result" повинен містити
валідний SQL запит у вигляді рядка. SQL запит повинен бути без зайвих слів чи коментарів,
лише чистий SQL.\r\n\r\nЯкщо "Source" є "Internet", то "Result" повинен містити
текстову відповідь на запит користувача, сформовану на основі твоїх загальних знань.
Відповідь має бути інформативною та по суті запиту.\r\n\r\nПорядок дій:\r\n\r\n1.
Проаналізуй запит користувача.\r\n2. Визнач, чи стосується запит інформації, яка може
бути отримана з наданої схеми бази даних про фігурне катання.\r\n3. *Особлива увага:*
Якщо запит є загальним, але *може* бути інтерпретований в контексті даних про фігурне
катання, що зберігаються в базі даних (наприклад, "Які є категорії в спорті?", "Хто
вигравав змагання?", "Розкажи про спортсменів з Японії"), **вважай його таким, що
відноситься до бази даних**. Спробуй сформулювати SQL-запит, який би відповів на це питання
в контексті фігурного катання.\r\n4. Якщо запит однозначно стосується інформації, яка
може бути отримана з наданої схеми бази даних про фігурне катання, встанови "Source":
"Database" і сформулюй відповідний SQL запит для поля "Result".\r\n5. Якщо запит
*очевидно* не стосується фігурного катання або інформації, яка може бути в наданій базі
```

даних (наприклад, запити про погоду, політику, інші види спорту, не пов'язані з даними в БД, загальні знання), встанови `"Source": "Internet"` і сформулюй текстову відповідь для поля `"Result"`.
Поверни єдиний JSON об'єкт з цими двома полями.
Важливо: Спочатку завжди ретельно аналізуй, чи може запит бути виконаний на основі наданої схеми бази даних про фігурне катання, особливо враховуючи пункт 3 порядку дій. Лише якщо це неможливо, переходь до відповіді з `"Source": "Internet"`. Переконайся, що відповідь є валідним JSON. SQL запити мають бути чистими, без додаткових пояснень чи дужок навколо всього запиту, якщо це не є частиною синтаксису самого SQL

запиту.
Приклади:
Користувач: "Виведи всіх спортсменів з Франції"
Штучний інтелект: `{"Source": "Database", "Result": "SELECT * FROM sportsman WHERE nationality = 'FRA'"}`
Користувач: "Які спортсмени виграли World Figure Skating Championship у 2022 році в категорії 'Man'?"
Штучний інтелект: `{"Source": "Database", "Result": "SELECT s.name FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id JOIN category cat ON r.category_id = cat.id WHERE c.name = 'World Figure Skating Championship' AND c.year = 2022 AND cat.name = 'Man' AND r.fpl = 1"}`
Користувач: "Які існують категорії в змаганнях?"
Штучний інтелект: `{"Source": "Database", "Result": "SELECT name FROM category"}`
Користувач: "Хто з японських спортсменів показував найкращі результати?"
Штучний інтелект: `{"Source": "Database", "Result": "SELECT s.name, r.points, c.name AS competition_name, c.year FROM sportsman s JOIN result r ON s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id WHERE s.nationality = 'JPN' ORDER BY r.points DESC LIMIT 10"}`
Користувач: "Яка сьогодні погода в Києві?"
Штучний інтелект: `{"Source": "Internet", "Result": "Щоб надати вам актуальну інформацію про погоду в Києві, мені потрібен доступ до сервісу погоди в реальному часі. Будь ласка, перевірте спеціалізований погодні сайт або додаток."}`
Користувач: "Хто президент США?"
Штучний інтелект: `{"Source": "Internet", "Result": "Станом на мої останні оновлення знань, президентом Сполучених Штатів Америки є Джо Байден. Для найактуальнішої інформації, будь ласка, перевірте надійне джерело новин."}`
Користувач: "Розкажи про історію Олімпійських ігор."
Штучний інтелект: `{"Source": "Internet", "Result": "Олімпійські ігри мають багату історію, що сягає Стародавньої Греції, де перші відомі ігри відбулися в Олімпії у 776 році до н.е. на честь бога Зевса. Вони включали різноманітні атлетичні змагання. Сучасні Олімпійські ігри були відроджені наприкінці XIX століття П'єром де Кубертеном, і перші з них відбулися в Афінах у 1896 році. З того часу вони стали головною міжнародною спортивною подією, що проводиться кожні чотири роки (літні та зимові ігри чергуються)."}"`,

```
};
```

```
string finalPrompt = $"User: {userQuestion}";
```

```
messages.Add(new Message(RoleType.User, finalPrompt));
```

```
var parameters = new MessageParameters()
```

```
{
```

```
    Messages = messages,
```

```
    MaxTokens = 1024,
```

```
    Model = models[model],
```

```
    Stream = false,
```

```

        Temperature = 1.0m,
    };
    UpdateLog("Creating request for Claude API...");
    var result = string.Empty;
    var metricsInfo = string.Empty;
    try
    {
        UpdateLog($"Sending request to Claude API ...");
        // Надсилає POST-запит до Gemini API
        var response = await client.Messages.GetClaudeMessageAsync(parameters);
        UpdateLog("Claude API responded successfully. Processing response...");
        UpdateLog("Analysis complete. Source determined.");
        result = response.Message.ToString();
        stopwatch.Stop();

        metricsInfo = DisplayClaudeMetrics(userQuestion, stopwatch.Elapsed,
startTime, response, result, model);
    }
    catch (Exception ex)
    {
        UpdateLog($"Error occurred: {ex.Message}.");
    }

    return (result, metricsInfo);
}

public async Task<string> ClaudeGetAiResponseFormatterAsync(string userQuestion,
string jsonData, string model)
{
    UpdateLog("Preparing response formatting request to Claude API...");
    var apiKey = "";
    var client = new AnthropicClient(apiKey);

    var finalPrompt = "Завдання:\n\nТи – мовний асистент. Твоя задача – взяти
питання користувача та JSON-дані, які містять інформацію для відповіді, і сформулювати

```

природну, зрозумілу та повну відповідь для користувача українською мовою.\n\nВхідні дані:\n\nПитання користувача: {user_question}\n\nJSON-дані: {json_data}\n\nІнструкції:\n\nПроаналізуй питання: Зрозумій, яку саме інформацію шукає користувач.\n\nВитягни дані з JSON: Уважно вивчи структуру JSON та витягни з нього тільки ті дані, які безпосередньо потрібні для відповіді на поставлене питання. Зверни увагу на ключі та значення в JSON. Якщо в JSON є кілька об'єктів або полів, вибери найбільш релевантні.\n\nСформулюй відповідь:\n\nНапиши відповідь українською мовою.\n\nВідповідь має бути прямою та чітко відповідати на питання користувача.\n\nВикористай витягнуті з JSON дані для формування змісту відповіді.\n\nПредстав інформацію у зручному для читання форматі (наприклад, повне речення або короткий абзац).\n\nНе показуй користувачеві сам JSON або його сиру структуру. Перетвори дані на природний текст.\n\nЯкщо JSON містить імена, прізвища чи інші власні назви, спробуй відтворити їх коректно в українському тексті.\n\nНадай тільки сутність відповіді. Уникай будь-яких вступних фраз (наприклад, "Згідно з даними...", "Відповідь на ваше питання...", "Так, звичайно..."), пояснень, підсумкових речень або іншої додаткової інформації, яка не є частиною прямої відповіді. Твоя відповідь має бути максимально лаконічною, але повною по суті питання.\n\nПриклад (для ілюстрації):\n\nПитання користувача: "Хто зайняв останнє місце на чемпіонаті Європи зі спортивних пар у 2022 році?"\n\nJSON-дані: [{"name": "Greta CRAFOORD"}, {"name1": "John CRAFOORD"}]\n\nОчікувана відповідь: "Останнє місце на чемпіонаті Європи зі спортивних пар у 2022 році посіли Грета Крафорд (Greta CRAFOORD) та Джон Крафорд (John CRAFOORD)." (Або схожий лаконічний варіант).\n\nВиконай завдання для наданих питання та JSON.";

```

var messages = new List<Message>()
{
    new Message(RoleType.User, finalPrompt),
};

finalPrompt = finalPrompt.Replace("{user_question}",
userQuestion).Replace("{json_data}", jsonData);

messages.Add(new Message(RoleType.User, finalPrompt));

var parameters = new MessageParameters()
{
    Messages = messages,
    MaxTokens = 1024,
    Model = models[model],
    Stream = false,
    Temperature = 1.0m,
};

UpdateLog("Creating formatting request for Claude API...");

var result = string.Empty;

try

```

```

    {
        UpdateLog("Formatting successful. Processing final response...");
        // Надсилає POST-запит до Gemini API
        var response = await client.Messages.GetClaudeMessageAsync(parameters);
        UpdateLog("Claude API responded successfully. Processing response...");
        UpdateLog("Analysis complete. Source determined.");
        result = response.Message.ToString();
    }
    catch (Exception ex)
    {
        UpdateLog($"Error occurred: {ex.Message}.");
    }
    return result;
}

public async Task<(string, string)> GptGetSQLBasedOnUserQuestionAsync(string
userQuestion, string model)
{
    var stopwatch = Stopwatch.StartNew();
    var responseBuilder = new StringBuilder();
    DateTime startTime = DateTime.Now;
    UpdateLog("Preparing request to Gpt API for SQL/Internet analysis...");
    ChatClient client = new(models[model], apiKey: "");
    List<ChatMessage> messages =
    [
        new UserChatMessage("Ось схема таблиц в базі даних, включаючи первинні
і зовнішні ключі.\r\nTable: category\r\n\r\nCREATE TABLE category (\r\n    id INT NOT
NULL PRIMARY KEY,\r\n    name VARCHAR(45) NOT NULL CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci\r\n);\r\n\r\nTable: competition\r\n\r\nCREATE TABLE competition (\r\n
id INT NOT NULL PRIMARY KEY,\r\n    name VARCHAR(45) NOT NULL CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci,\r\n    year INT NOT NULL\r\n);\r\n\r\nTable:
result\r\n\r\nCREATE TABLE result (\r\n    id INT NOT NULL PRIMARY KEY,\r\n
category_id INT NOT NULL,\r\n    competition_id INT NOT NULL,\r\n    sportsman_id INT NOT
NULL,\r\n    sportsman2_id INT NULL, -- Дозволяє NULL значення, використовується для
парних категорій\r\n    fpl INT NOT NULL,\r\n    sp INT NOT NULL,\r\n    fs INT NOT
NULL,\r\n    points FLOAT NOT NULL,\r\n    FOREIGN KEY (category_id) REFERENCES
category(id),\r\n    FOREIGN KEY (competition_id) REFERENCES competition(id),\r\n

```

```

FOREIGN KEY (sportsman_id) REFERENCES sportsman(id),\r\n    FOREIGN KEY (sportsman2_id)
REFERENCES sportsman(id) -- Також посилається на sportsman\r\n);\r\n\r\nTable:
sportsman\r\n\r\nCREATE TABLE sportsman (\r\n    id INT NOT NULL PRIMARY KEY,\r\n    name
VARCHAR(45) NOT NULL CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci,\r\n    nationality
VARCHAR(45) NOT NULL CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci\r\n);\r\n\r\nОсь
додаткові відомості про дані в базі даних:\r\n\r\nБаза даних містить результати змагань з
фігурного катання. Вона включає:\r\n\r\nТаблиця 'category' (id, name) – категорії: 'Man',
'Woman', 'Pairs', 'Dance'.\r\n\r\nТаблиця 'competition' (id, name, year) – Змагання з
роками: Four Continents Figure Skating Championship, World Figure Skating Championship,
European Figure Skating Championship.\r\n\r\nТаблиця 'sportsman' (id, name, nationality)
– спортсмени, їхня національність вказана в такому форматі (наприклад, JPN, USA, GEO,
ITA, FRA).\r\n\r\nТаблиця 'result' містить (id, category_id, competition_id,
sportsman_id, sportsman2_id, fpl, sp, fs, points) – Результати змагань:\r\n\r\nfpl (final
placement) – фінальне місце.\r\n\r\nsp (short program score) – оцінка за коротку
програму.\r\n\r\nfs (free skate score) – оцінка за довільну програму.\r\n\r\npoints
(total points) – загальний бал.\r\n\r\nІнструкції щодо відповіді:\r\n\r\nТвоє завдання –
проаналізувати запит користувача і надати відповідь у форматі єдиного JSON об'єкта. Цей
JSON об'єкт повинен мати два ключі:\r\n\r\n"Source\": Значення цього ключа вказує на
джерело інформації.\r\n\r\n"Database\": якщо відповідь базується на інформації з наданої
схеми бази даних.\r\n\r\n"Internet\": якщо відповідь базується на загальних знаннях,
оскільки запит не стосується даних у БД.\r\n\r\n"Result\": Значення цього ключа містить
саму відповідь.\r\n\r\nЯкщо "Source" є "Database", то "Result" повинен містити
валідний SQL запит у вигляді рядка. SQL запит повинен бути без зайвих слів чи коментарів,
лише чистий SQL.\r\n\r\nЯкщо "Source" є "Internet", то "Result" повинен містити
текстову відповідь на запит користувача, сформовану на основі твоїх загальних знань.
Відповідь має бути інформативною та по суті запиту.\r\n\r\nПорядок дій:\r\n\r\n1.
Проаналізуй запит користувача.\r\n\r\n2. Визнач, чи стосується запит інформації, яка може
бути отримана з наданої схеми бази даних про фігурне катання.\r\n\r\n3. *Особлива увага:**
Якщо запит є загальним, але *може* бути інтерпретований в контексті даних про фігурне
катання, що зберігаються в базі даних (наприклад, "Які є категорії в спорті?", "Хто
вигравав змагання?", "Розкажи про спортсменів з Японії"), **вважай його таким, що
відноситься до бази даних**. Спробуй сформувавти SQL-запит, який би відповів на це питання
в контексті фігурного катання.\r\n\r\n4. Якщо запит однозначно стосується інформації, яка
може бути отримана з наданої схеми бази даних про фігурне катання, встанови "Source":
"Database" і сформувай відповідний SQL запит для поля "Result".\r\n\r\n5. Якщо запит
*очевидно* не стосується фігурного катання або інформації, яка може бути в наданій базі
даних (наприклад, запити про погоду, політику, інші види спорту, не пов'язані з даними в
БД, загальні знання), встанови "Source": "Internet" і сформувай текстову відповідь для
поля "Result".\r\n\r\n6. Поверни єдиний JSON об'єкт з цими двома полями.\r\n\r\nВажливо:
Спочатку завжди ретельно аналізуй, чи може запит бути виконаний на основі наданої схеми
бази даних про фігурне катання, особливо враховуючи пункт 3 порядку дій. Лише якщо це
неможливо, переходь до відповіді з "Source": "Internet". Переконайся, що відповідь є
валідним JSON. SQL запити мають бути чистими, без додаткових пояснень чи дужок навколо
всього запиту, якщо це не є частиною синтаксису самого SQL
запиту.\r\n\r\nПриклади:\r\n\r\nКористувач: "Виведи всіх спортсменів з
Франції"\r\n\r\nШтучний інтелект: {"Source": "Database", "Result": "SELECT * FROM
sportsman WHERE nationality = 'FRA'"}\r\n\r\nКористувач: "Які спортсмени виграли World
Figure Skating Championship у 2022 році в категорії 'Man'?"\r\n\r\nШтучний інтелект:
{"Source": "Database", "Result": "SELECT s.name FROM sportsman s JOIN result r ON
s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id JOIN category cat ON

```

```
r.category_id = cat.id WHERE c.name = 'World Figure Skating Championship' AND c.year =
2022 AND cat.name = 'Man' AND r.fpl = 1\}\r\n\r\nКористувач: \\"Які існують категорії в
змаганнях?\\"r\nШтучний інтелект: {\\"Source\\": \\"Database\\", \\"Result\\": \\"SELECT name
FROM category\\"\}\r\n\r\nКористувач: \\"Хто з японських спортсменів показував найкращі
результати?\\"r\nШтучний інтелект: {\\"Source\\": \\"Database\\", \\"Result\\": \\"SELECT
s.name, r.points, c.name AS competition_name, c.year FROM sportsman s JOIN result r ON
s.id = r.sportsman_id JOIN competition c ON r.competition_id = c.id WHERE s.nationality =
'JPN' ORDER BY r.points DESC LIMIT 10\\"\}\r\n\r\nКористувач: \\"Яка сьогодні погода в
Києві?\\"r\nШтучний інтелект: {\\"Source\\": \\"Internet\\", \\"Result\\": \\"Щоб надати вам
актуальну інформацію про погоду в Києві, мені потрібен доступ до сервісу погоди в
реальному часі. Будь ласка, перевірте спеціалізований погодні сайт або
додаток.\\"}\r\n\r\nКористувач: \\"Хто президент США?\\"r\nШтучний інтелект: {\\"Source\\":
\\"Internet\\", \\"Result\\": \\"Станом на мої останні оновлення знань, президентом Сполучених
Штатів Америки є Джо Байден. Для найактуальнішої інформації, будь ласка, перевірте
надійне джерело новин.\\"}\r\n\r\nКористувач: \\"Розкажи про історію Олімпійських
ігор.\\"r\nШтучний інтелект: {\\"Source\\": \\"Internet\\", \\"Result\\": \\"Олімпійські ігри
мають багату історію, що сягає Стародавньої Греції, де перші відомі ігри відбулися в
Олімпії у 776 році до н.е. на честь бога Зевса. Вони включали різноманітні атлетичні
змагання. Сучасні Олімпійські ігри були відроджені наприкінці XIX століття П'єром де
Кубертенем, і перші з них відбулися в Афінах у 1896 році. З того часу вони стали головною
міжнародною спортивною подією, що проводиться кожні чотири роки (літні та зимові ігри
чергуються).\\"}),
```

```
];
```

```
string finalPrompt = $"User: {userQuestion}";
```

```
messages.Add(new UserChatMessage(finalPrompt));
```

```
UpdateLog("Creating request for Gpt API...");
```

```
var result = string.Empty;
```

```
var metricsInfo = string.Empty;
```

```
try
```

```
{
```

```
    UpdateLog($"Sending request to Gpt API ...");
```

```
    // Надсилає POST-запит до Gemini API
```

```
    ChatCompletion response = await client.CompleteChatAsync(messages);
```

```
    UpdateLog("Gpt API responded successfully. Processing response...");
```

```
    UpdateLog("Analysis complete. Source determined.");
```

```
    result = response.Content[0].Text;
```

```
    stopwatch.Stop();
```

```
    var allCharactersCount = messages.Sum(x =>
x.Content.FirstOrDefault()?.Text.Length);
```

```

        metricsInfo = DisplayOpenAIOrGeminiMetrics(allCharactersCount,
stopwatch.Elapsed, startTime, result, model);
    }
    catch (Exception ex)
    {
        UpdateLog($"Error occurred: {ex.Message}.");
    }
    return (result, metricsInfo);
}

public async Task<string> GptGetAiResponseFormatterAsync(string userQuestion,
string jsonData, string model)
{
    UpdateLog("Preparing response formatting request to Gpt API...");

    var finalPrompt = "Завдання:\n\nТи – мовний асистент. Твоя задача – взяти
питання користувача та JSON-дані, які містять інформацію для відповіді, і сформулювати
природну, зрозумілу та повну відповідь для користувача українською мовою.\n\nВхідні
дані:\n\nПитання користувача: {user_question}\n\nJSON-дані:
{json_data}\n\nІнструкції:\n\nПроаналізуй питання: Зрозумій, яку саме інформацію шукає
користувач.\n\nВитягни дані з JSON: Уважно вивчи структуру JSON та витягни з нього тільки
ті дані, які безпосередньо потрібні для відповіді на поставлене питання. Зверни увагу на
ключі та значення в JSON. Якщо в JSON є кілька об'єктів або полів, вибери найбільш
релевантні.\n\nСформулюй відповідь:\n\nНапиши відповідь українською мовою.\n\nВідповідь
має бути прямою та чітко відповідати на питання користувача.\n\nВикористай витягнуті з
JSON дані для формування змісту відповіді.\n\nПредстав інформацію у зручному для читання
форматі (наприклад, повне речення або короткий абзац).\n\nНе показуй користувачеві сам
JSON або його сиру структуру. Перетвори дані на природний текст.\n\nЯкщо JSON містить
імена, прізвища чи інші власні назви, спробуй відтворити їх коректно в українському
тексті.\n\nНадай тільки сутність відповіді. Уникай будь-яких вступних фраз (наприклад,
\"Згідно з даними...\", \"Відповідь на ваше питання...\", \"Так, звичайно...\"),
пояснень, підсумкових речень або іншої додаткової інформації, яка не є частиною прямої
відповіді. Твоя відповідь має бути максимально лаконічною, але повною по суті
питання.\n\nПриклад (для ілюстрації):\n\nПитання користувача: \"Хто зайняв останнє місце
на чемпіонаті Європи зі спортивних пар у 2022 році?\"\n\nJSON-дані: [{\"name\": \"Greta
CRAFOORD\", \"name1\": \"John CRAFOORD\"}]\n\nОчікувана відповідь: \"Останнє місце на
чемпіонаті Європи зі спортивних пар у 2022 році посіли Грета Краффорд (Greta CRAFOORD) та
Джон Краффорд (John CRAFOORD).\" (Або схожий лаконічний варіант).\n\nВиконай завдання для
наданих питання та JSON.";

    ChatClient client = new(models[model], apiKey: "");

    List<ChatMessage> messages =
    [
        new UserChatMessage(finalPrompt),

```

```

];
    finalPrompt = finalPrompt.Replace("{user_question}",
userQuestion).Replace("{json_data}", jsonData);
    messages.Add(new UserChatMessage(finalPrompt));
    UpdateLog("Creating formatting request for Gpt API...");
    var result = string.Empty;
    try
    {
        UpdateLog("Formatting successful. Processing final response...");
        ChatCompletion response = await client.CompleteChatAsync(messages);

        UpdateLog("Gpt API responded successfully. Processing response...");

        UpdateLog("Analysis complete. Source determined.");
        result = response.Content[0].Text;

    }
    catch (Exception ex)
    {
        UpdateLog($"Error occurred: {ex.Message}.");
    }
    return result;
}

private string DisplayClaudeMetrics(string input, TimeSpan responseTime, DateTime
startTime,
                                MessageResponse output, string fullResponse, string
model)
{
    var metricsBuilder = new StringBuilder();
    metricsBuilder.AppendLine("" + new string('-', 72));
    metricsBuilder.AppendLine("📊 CLAUDE RESPONSE METRICS");
    metricsBuilder.AppendLine(new string('-', 72));

```

```

// Часові метрики
metricsBuilder.AppendLine($"🕒 Response Time:
{responseTime.TotalMilliseconds:F2} ms ({responseTime.TotalSeconds:F2} seconds)");
metricsBuilder.AppendLine($"🕒 Started at: {startTime:HH:mm:ss.fff}");
metricsBuilder.AppendLine($"🕒 Completed at: {DateTime.Now:HH:mm:ss.fff}");
// Вик. токенів
if ( output.Usage != null)
{
    int inputTokens = output.Usage.InputTokens;
    int outputTokens = output?.Usage?.OutputTokens ?? 0;
    metricsBuilder.AppendLine($"🎯 Input tokens: {inputTokens}");
    metricsBuilder.AppendLine($"🎯 Output tokens: {outputTokens}");
    metricsBuilder.AppendLine($"🎯 Total tokens: {inputTokens +
outputTokens}");
    // Обчислення вартості
    if (ModelPricing.ContainsKey(model))
    {
        var (inputPrice, outputPrice) = ModelPricing[model];
        decimal inputCost = (inputTokens / 1_000_000m) * inputPrice;
        decimal outputCost = (outputTokens / 1_000_000m) * outputPrice;
        decimal totalCost = inputCost + outputCost;
        metricsBuilder.AppendLine($"💰 Input cost: ${inputCost:F6}");
        metricsBuilder.AppendLine($"💰 Output cost: ${outputCost:F6}");
        metricsBuilder.AppendLine($"💰 Total cost: ${totalCost:F6}");
    }
}
// Текстові метрики
metricsBuilder.AppendLine($"📄 Input length: {input.Length} characters");
metricsBuilder.AppendLine($"📄 Output length: {fullResponse.Length}
characters");
return metricsBuilder.ToString();
}

```

```

private string DisplayOpenAIOrGeminiMetrics(int allcharacters, TimeSpan
responseTime, DateTime startTime, string fullResponse, string model)
{
    var metricsBuilder = new StringBuilder();
    metricsBuilder.AppendLine(" + new string('-', 72));
    if (model.ToLower().Contains("gemini"))
    {
        metricsBuilder.AppendLine(" 🏠 GEMINI RESPONSE METRICS");
    }
    else
    {
        metricsBuilder.AppendLine(" 🏠 OPENAI RESPONSE METRICS");
    }
    metricsBuilder.AppendLine(new string('-', 72));
    // Часові метрики
    metricsBuilder.AppendLine($" 🕒 Response Time:
{responseTime.TotalMilliseconds:F2} ms ({responseTime.TotalSeconds:F2} seconds)");
    metricsBuilder.AppendLine($" 🕒 Started at: {startTime:HH:mm:ss.fff}");
    metricsBuilder.AppendLine($" 🕒 Completed at: {DateTime.Now:HH:mm:ss.fff}");
    metricsBuilder.AppendLine($" 📄 Input length: {allcharacters} characters");
    //metricsBuilder.AppendLine($" 📄 Output length: {totalCharacters}
characters");
    int estimatedInputTokens = EstimateTokens(allcharacters);
    int estimatedOutputTokens = EstimateTokens(fullResponse.Length);
    metricsBuilder.AppendLine($" 🎯 Estimated input tokens:
{estimatedInputTokens}");
    metricsBuilder.AppendLine($" 🎯 Estimated output tokens:
{estimatedOutputTokens}");
    metricsBuilder.AppendLine($" 🎯 Total estimated tokens: {estimatedInputTokens
+ estimatedOutputTokens}");
    // Обч. вартості
    if (ModelPricing.ContainsKey(model))
    {

```

```

        var (inputPrice, outputPrice) = ModelPricing[model];
        decimal inputCost = (estimatedInputTokens / 1_000_000m) * inputPrice;
        decimal outputCost = (estimatedOutputTokens / 1_000_000m) * outputPrice;
        decimal totalCost = inputCost + outputCost;

        metricsBuilder.AppendLine($"💰 Input cost: ${inputCost:F6}");
        metricsBuilder.AppendLine($"💰 Output cost: ${outputCost:F6}");
        metricsBuilder.AppendLine($"💰 Total estimated cost: ${totalCost:F6}");
    }

    // Текстові метрики
    int wordCount = fullResponse.Split(new[] { ' ', '\n', '\r', '\t' },
StringSplitOptions.RemoveEmptyEntries).Length;
    int lineCount = fullResponse.Split('\n').Length;
    metricsBuilder.AppendLine($"📖 Word count: {wordCount}");
    metricsBuilder.AppendLine($"📄 Line count: {lineCount}");
    metricsBuilder.AppendLine(new string('-', 72));
    return metricsBuilder.ToString();
}

private int EstimateTokens(int number)
{
    if(number == 0) return 0;
    return (int)Math.Ceiling(number / 4.0);
}
}

using MySql.Data.MySqlClient;
using System.Data;
namespace Services
{
    public class MySQLContext
    {

```

```

private readonly string _connString;
public MySQLContext(string connString)
{
    _connString = connString;
}
public DataTable ExecuteMySQLCommand(string commandString)
{
    DataTable dt = new DataTable();
    using (MySQLConnection con = new MySQLConnection(_connString))
    {
        MySqlCommand cmd = new MySqlCommand(commandString, con);
        try
        {
            con.Open();
            using (MySQLDataReader dr = cmd.ExecuteReader())
            {
                if (dr.HasRows)
                {
                    dt.Load(dr);
                }
            }
            con.Close();
        }
        catch (Exception ex)
        {
            MessageBox.Show("Неможливо з'єднати з SQL-сервером! \nПеревірте наявність Інтернету...",
                ex.Message, MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    return dt;
}

```

```

    }
}
namespace Services
{
    public class QueryData
    {
        public string Source { get; set; }
        public string Result { get; set; }
    }
}
using Newtonsoft.Json;
using Services;
namespace figure_skating
{
    public partial class SixthControl1 : UserControl
    {
        public SixthControl1()
        {
            InitializeComponent();
        }

        private readonly Dictionary<string, (decimal inputPrice, decimal outputPrice)>
ModelPricing = new()
        {
            // OpenAI GPT-4.1 series
            { "gpt-4.1", (2.00m, 8.00m) },
            { "gpt-4.1-mini", (0.40m, 1.60m) },
            { "gpt-4.1-nano", (0.10m, 0.40m) },
            // Anthropic Claude 4 (2025-05-14)
            { "claude-opus-4-20250514", (15.00m, 75.00m) },
            { "claude-sonnet-4-20250514", (3.00m, 15.00m) },
            // Anthropic Claude 3.7 (2025-02-19)
            { "claude-3-7-sonnet-20250219", (3.00m, 15.00m) },

```

```

// Anthropic Claude 3.5 (2024-10-22)
{ "claude-3-5-sonnet-20241022", (3.00m, 15.00m) },
{ "claude-3-5-haiku-20241022", (0.80m, 4.00m) },
// Anthropic Claude 3.0 (2024-02-29 / 2024-03-07)
{ "claude-3-opus-20240229", (15.00m, 75.00m) },
{ "claude-3-haiku-20240307", (0.25m, 1.25m) }
};

private async void btn_run_Click(object sender, EventArgs e)
{
    richTextBox3.Text = string.Empty;
    var model = comboBox1.Text;
    btn_run.Enabled = false;
    var userInput = richTextBox1.Text;
    if (string.IsNullOrEmpty(userInput))
    {
        btn_run.Enabled = true; // Re-enable button if input is empty
        return;
    }
    string formattedResponse = string.Empty;
    string source = string.Empty;
    string sqlResult = string.Empty;
    var client = new AIApiClient(logInfo);
    try
    {
        if (model.Contains("Gemini"))
        {
            (source, formattedResponse, sqlResult) = await
CompleteChatWithGeminiAsync(client, userInput, model);
        }
        else if (model.Contains("claude"))
        {

```

```

        (source, formattedResponse, sqlResult) = await
CompleteChatWithClaudeAsync(client, userInput, model);
    }
    else if (model.Contains("gpt"))
    {
        (source, formattedResponse, sqlResult) = await
CompleteChatWithGptAsync(client, userInput, model);
    }

    logInfo.Text = source;
    richTextBox2.Text = formattedResponse;
    if (!string.IsNullOrEmpty(sqlResult))
    {
        richTextBox3.AppendText("\nSQL: " + sqlResult);
    }
}
catch (Exception ex)
{
    logInfo.Text = $"Error: {ex.Message}";
    richTextBox2.Text = $"An error occurred: {ex.Message}";
}
finally
{
    btn_run.Enabled = true;
}
}

// Очищає рядок від Markdown (json,) та десеріалізує його в об'єкт QueryData
(string source, string result) GetSourceAndResult(string rawJson)
{
    string jsonString = rawJson
        .Replace("`json", "")
        .Replace("`", "");
}

```

```

.Trim();
try
{
    QueryData? queryDataObject =
System.Text.Json.JsonSerializer.Deserialize<QueryData>(jsonString);
    if (queryDataObject != null)
    {
        return (queryDataObject.Source, queryDataObject.Result);
    }
    else
    {
        Console.WriteLine("Deserialization returned null.");
    }
}
catch (System.Text.Json.JsonException ex)
{
    return (string.Empty, string.Empty);
}
return (string.Empty, string.Empty);
}

private async Task<(string, string, string)>
CompleteChatWithClaudeAsync(AIApiClient client, string userInput, string model)
{
    var (responseText, metricsInfo) = await
client.ClaudeGetSQLBasedOnUserQuestionAsync(userInput, model);
    var (source, result) = GetSourceAndResult(responseText);
    var serializedResult = result;
    richTextBox3.AppendText(metricsInfo);
    var sqlResult = "";
    if (source == "Database")
    {
        sqlResult = result;
    }
}

```

```

        MySQLContext context = new MySQLContext("server=localhost;
database=figureskating; user=root; password=Khrystyna2003");
        var datatable = context.ExecuteMySQLCommand(result);
        serializedResult = JsonConvert.SerializeObject(datatable);
    }

    return (source, await client.ClaudeGetAiResponseFormatterAsync(userInput,
serializedResult, model), sqlResult);
}

private async Task<(string, string, string)>
CompleteChatWithGeminiAsync(AIApiClient client, string userInput, string model)
{
    // Gemini logic
    var (responseText, metricsInfo) = await
client.GeminiGetSQLBasedOnUserQuestionAsync(userInput, model);
    var (source, result) = GetSourceAndResult(responseText);
    richTextBox3.AppendText(metricsInfo);
    var serializedResult = result;
    var sqlResult = "";
    if (source == "Database")
    {
        sqlResult = result;

        MySQLContext context = new MySQLContext("server=localhost;
database=figureskating; user=root; password=Khrystyna2003");
        var datatable = context.ExecuteMySQLCommand(result);
        serializedResult = JsonConvert.SerializeObject(datatable);
    }

    return (source, await client.GeminiGetAiResponseFormatterAsync(userInput,
serializedResult, model), sqlResult);
}

private async Task<(string, string, string)> CompleteChatWithGptAsync(AIApiClient
client, string userInput, string model)
{
    var (responseText, metricsInfo) = await
client.GptGetSQLBasedOnUserQuestionAsync(userInput, model); // Call the method and get
the responseText = await client.GptGetSQLBasedOnUserQuestionAsync(userInput, model);

```

```

        var (source, result) = GetSourceAndResult(responseText);
        richTextBox3.AppendText(metricsInfo);
        var serializedResult = result;
        var sqlResult = "";
        if (source == "Database")
        {
            sqlResult = result;
            MySQLContext context = new MySQLContext("server=localhost;
database=figureskating; user=root; password=Khrystyna2003");
            var datatable = context.ExecuteMySQLCommand(result);
            serializedResult = JsonConvert.SerializeObject(datatable);
        }
        return (source, await client.GptGetAiResponseFormatterAsync(userInput,
serializedResult, model), sqlResult);
    }
    private void SixthControl1_Load(object sender, EventArgs e)
    {
        comboBox1.SelectedIndex = 0;
    }
    private void button1_Click(object sender, EventArgs e)
    {
        richTextBox3.Visible = !richTextBox3.Visible;
    }
    private void panel3_Paint(object sender, PaintEventArgs e)
    {
    }
}
}
using OxyPlot;
using OxyPlot.Axes;
using OxyPlot.Legends;
using OxyPlot.Series;

```

```

using OxyPlot.WindowsForms;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.Menu;
usingBarItem = OxyPlot.Series.BarItem;
namespace Analysis
{
    public class GraphGenerator
    {
        private readonly DataAccess _dataAccess;
        public GraphGenerator(string connectionString)
        {
            _dataAccess = new DataAccess(connectionString);
        }
        // 1. Winners Bar Chart
        public PlotModel CreateWinnersBarChart()
        {
            var model = new PlotModel { DefaultFont = "Century Gothic",
                TitleFontWeight = 700, Title = "Top Winners by Category",
                PlotAreaBackground = OxyPlot.OxyColors.LightGray };
            var data = _dataAccess.GetWinnersStatsByCategory();
            var categoryAxis = new CategoryAxis { Position = AxisPosition.Left,
                TitleFontWeight = 700, FontWeight = 700 };
            // Configure the value axis to use integer intervals
            var valueAxis = new LinearAxis
            {
                TitleFontWeight = 700, // Set font weight for axis title
                FontWeight = 700, // Set font weight for axis labels
                Position = AxisPosition.Bottom,
                Title = "Number of Wins",
                MinimumMajorStep = 1,
                MinimumMinorStep = 1,
                MajorStep = 1,
                MinorTickSize = 0
            }
        }
    }
}

```

```

};
var series = new BarSeries
{
    Title = "Wins",
    StrokeThickness = 1,
    FillColor = OxyPlot.OxyColors.CadetBlue
};
foreach (var item in data.Take(15))
{
    if (item.Category == "Pairs" || item.Category == "Dance")
    {
        categoryAxis.Labels.Add($"{item.Name} & {item.Name2}
({item.Category})");
    }
    else
    {
        categoryAxis.Labels.Add($"{item.Name} ({item.Category})");
    }
    series.Items.Add(new BarItem { Value = item.WinsCount });
}
model.Axes.Add(categoryAxis);
model.Axes.Add(valueAxis);
model.Series.Add(series);
return model;
}
// 2. Score Progression Line Chart
public PlotModel CreateScoreProgressionChart()
{
    var model = new PlotModel { DefaultFont = "Century Gothic",
        TitleFontWeight = 700, Title = "Average Score Progression by Year and
Category",
        PlotAreaBackground = OxyPlot.OxyColors.LightGray

```

```

};

    var data = _dataAccess.GetScoreProgressionByYear();
// Create and configure the legend object
var legend = new OxyPlot.Legends.Legend
{
    LegendPlacement = OxyPlot.Legends.LegendPlacement.Inside,
    LegendPosition = OxyPlot.Legends.LegendPosition.TopLeft,
    LegendOrientation = OxyPlot.Legends.LegendOrientation.Vertical
};
// Add the legend to the model
legend.LegendFontWeight = 700;
model.Legends.Add(legend);
var xAxis = new LinearAxis
{
    Position = AxisPosition.Bottom,
    Title = "Year",
    TitleFontWeight = 700, // Set font weight for axis title
    FontWeight = 700, // Set font weight for axis labels
    MinimumMajorStep = 1,
    MinimumMinorStep = 1,
    MajorStep = 1,
    MinorTickSize = 0
};
var yAxis = new LinearAxis { Position = AxisPosition.Left, Title = "Average
Score",
    TitleFontWeight = 700,
    FontWeight = 700};
model.Axes.Add(xAxis);
model.Axes.Add(yAxis);
var categories = data.Select(x => x.Category).Distinct();
var colors = new[] { OxyColors.Blue, OxyColors.Red, OxyColors.Green,
OxyColors.Purple };

```

```

int colorIndex = 0;
foreach (var category in categories)
{
    var lineSeries = new LineSeries
    {
        Title = category,
        Color = colors[colorIndex % colors.Length],
        MarkerType = MarkerType.Circle,
        MarkerSize = 4,
        TrackerFormatString = "{0}\nYear: {2:0}\nAverage Score: {4:0.00}"
    };
    var categoryData = data.Where(x => x.Category == category).OrderBy(x =>
x.Year);
    foreach (var item in categoryData)
    {
        lineSeries.Points.Add(new DataPoint(item.Year, item.AverageScore));
    }
    model.Series.Add(lineSeries);
    colorIndex++;
}
return model;
}
// 3. Top Performers Horizontal Bar Chart
public PlotModel CreateTopPerformersChart()
{
    var model = new PlotModel { DefaultFont = "Century Gothic", Title = "Top
Performers by Best Score", PlotAreaBackground = OxyPlot.OxyColors.LightGray };
    dynamic data = _dataAccess.GetTopPerformersByPoints(15);
    // Вісь категорій для BestScore (з повними підписами)
    var categoryAxis1 = new CategoryAxis
    {
        Position = AxisPosition.Left,

```

```

    Key = "ShortLabels",
    IsAxisVisible = false,
    TitleFontWeight = 700, // Set font weight for axis title
    FontWeight = 700 // Set font weight for axis labels
};
// Вісь категорій для AverageScore (тільки ім'я або навіть пусто)
var categoryAxis2 = new CategoryAxis
{
    Position = AxisPosition.Left,
    Key = "FullLabels",
    TitleFontWeight = 700,
    FontWeight = 700
};
var valueAxis = new LinearAxis
{
    Position = AxisPosition.Bottom,
    Title = "Score",
    TitleFontWeight = 700, // Set font weight for axis title
    FontWeight = 700 // Set font weight for axis labels
};
var bestScoreSeries = new BarSeries
{
    Title = "Best Score",
    StrokeThickness = 1,
    FillColor = OxyColors.Khaki,
    YAxisKey = "ShortLabels",
    LabelPlacement = LabelPlacement.Outside,
    LabelFormatString = "{0:F2}" // округлення до сотих
};
var avgScoreSeries = new BarSeries
{
    StrokeThickness = 1,

```

```

        FillColor = OxyColors.LightBlue,
        Title = "", // приховає з легенди
        YAxisKey = "FullLabels",
        LabelPlacement = LabelPlacement.Inside,
        LabelFormatString = "{0:F2}" // округлення до сотих
    };
    foreach (var item in data)
    {
        categoryAxis1.Labels.Add($"{item.Name} ({item.Category}) bs event -
{item.CompetitionName} ({item.CompetitionYear})");
        categoryAxis2.Labels.Add(item.Name); // або тільки ім'я, або взагалі ""
якщо не треба
        bestScoreSeries.Items.Add(new BarItem { Value =
Math.Round((double)item.BestScore, 2) });
        avgScoreSeries.Items.Add(new BarItem { Value =
Math.Round((double)item.AverageScore, 2) });
        model.Axes.Add(categoryAxis1);
        model.Axes.Add(categoryAxis2);
        model.Axes.Add(valueAxis);
        model.Series.Add(bestScoreSeries);
        model.Series.Add(avgScoreSeries);
        return model;
    }
    // 4. Participation Trends Area Chart
    public PlotModel CreateParticipationTrendsChart()
    {
        var model = new PlotModel { DefaultFont = "Century Gothic", Title =
"Participation Trends Over Years", PlotAreaBackground = OxyPlot.OxyColors.LightGray };
        dynamic data = _dataAccess.GetParticipationTrends();
        var legend = new OxyPlot.Legends.Legend();
        legend.LegendTitle = "Categories";
        legend.LegendPosition = OxyPlot.Legends.LegendPosition.TopLeft;
        legend.LegendPlacement = OxyPlot.Legends.LegendPlacement.Inside;
    }
}

```

```

legend.LegendOrientation = OxyPlot.Legends.LegendOrientation.Vertical;
model.Legends.Add(legend);
var xAxis = new LinearAxis
{
    Position = AxisPosition.Bottom,
    Title = "Year",
    // Додаємо форматування осі для відображення цілих чисел
    StringFormat = "0",
    MinimumMajorStep = 1,
    MinimumMinorStep = 1,
    MajorStep = 1,
    MinorTickSize = 0,
    TitleFontWeight = 700,
    FontWeight = 700
};

var yAxis = new LinearAxis { Position = AxisPosition.Left, Title = "Number of
Participants" ,
    TitleFontWeight = 700,
    FontWeight = 700
};

model.Axes.Add(xAxis);
model.Axes.Add(yAxis);
// Оновлений словник кольорів згідно з вашим запитом
var colors = new Dictionary<string, OxyColor>
{
    { "Man", OxyColors.Blue },
    { "Woman", OxyColors.Pink },
    { "Dance", OxyColors.Green },
    { "Pairs", OxyColors.Yellow }
};

var categories = ((IEnumerable<dynamic>)data).Select(x =>
(string)x.Category).Distinct();

```

```

        foreach (var category in categories)
        {
            // Присвоюємо колір із словника. Якщо категорії немає, використовуємо
сірий за замовчуванням.
            OxyColor color = colors.ContainsKey(category) ? colors[category] :
OxyColors.Gray;
            var areaSeries = new AreaSeries
            {
                Title = category,
                Color = color,
                Fill = OxyColor.FromAColor(100, color),
                StrokeThickness = 2,
                TrackerFormatString = "{0}\nYear: {2:0}\nNumber of Participants:
{4:0}";
                var categoryData = ((IEnumerable<dynamic>)data).Where(x => x.Category ==
category).OrderBy(x => x.Year);
                foreach (var item in categoryData)
                {
                    areaSeries.Points.Add(new DataPoint((int)item.Year,
(int)item.Participants));
                }
                areaSeries.Points2.Add(new DataPoint(areaSeries.Points.First().X, 0));
                areaSeries.Points2.Add(new DataPoint(areaSeries.Points.Last().X, 0));
                model.Series.Add(areaSeries);
            }
            return model;
        }
// 5. Competition Difficulty Box Plot
public PlotModel CreateCompetitionDifficultyChart()
{
    var model = new PlotModel { DefaultFont = "Century Gothic", Title =
"Competition Difficulty (Score Spread and Standard Deviation for man)",
PlotAreaBackground = OxyPlot.OxyColors.LightGray };
    dynamic data = _dataAccess.GetCompetitionDifficulty();

```

```

        var categoryAxis = new CategoryAxis { Position = AxisPosition.Left, Title =
"Competition - Year",
            TitleFontWeight = 700,
            FontWeight = 700
        };

        var valueAxis = new LinearAxis { Position = AxisPosition.Bottom, Title =
"Score Metrics",
            TitleFontWeight = 700,
            FontWeight = 700
        };

var spreadSeries = new BarSeries
{
    Title = "Score Spread",
    FillColor = OxyColors.LightBlue,
    StrokeThickness = 1,
    LabelFormatString = "{0:F2}" // Округлення до сотих
};

var stdDevSeries = new BarSeries
{
    Title = "Standard Deviation",
    FillColor = OxyColors.LightSalmon,
    StrokeThickness = 1,
    LabelFormatString = "{0:F2}" // Округлення до сотих
};

foreach (var item in data)
{
    categoryAxis.Labels.Add($"{item.Competition} ({item.Year})");
    // Округлення значень перед додаванням
    spreadSeries.Items.Add(new BarItem { Value =
Math.Round((double)item.ScoreSpread, 2) });
    stdDevSeries.Items.Add(new BarItem { Value =
Math.Round((double)item.ScoreStdDev, 2) });
}

```

```

model.Axes.Add(categoryAxis);
model.Axes.Add(valueAxis);
model.Series.Add(spreadSeries);
model.Series.Add(stdDevSeries);
// Додавання легенди
model.Legends.Add(new Legend
{
    LegendPlacement = LegendPlacement.Outside,
    LegendPosition = LegendPosition.TopRight,
    LegendOrientation = LegendOrientation.Horizontal,
    LegendBackground = OxyColors.White, // Додаткові налаштування, якщо
потрібно
    LegendBorder = OxyColors.Black
});
return model;
}
// 6. Stacked Bar Chart of Wins by Nationality and Discipline
public PlotModel CreateWinsByNationalityStackedChart()
{
    var model = new PlotModel
    {
        DefaultFont = "Century Gothic",
        TitleFontWeight = 700,
        Title = "Wins Distribution by Country and Category",
        PlotAreaBackground = OxyPlot.OxyColors.LightGray
    };
    // 1. Отримання та попередня обробка даних
    var rawData = _dataAccess.GetWinsByNationalityAndCategory();
    // Визначаємо ТОП-10 країн за загальною кількістю перемог для відображення
    var topNationalities = rawData
        .GroupBy(d => d.Nationality)

```

```

        .Select(g => new { Nationality = g.Key, TotalWins = g.Sum(d =>
d.WinsCount) })
        .OrderByDescending(x => x.TotalWins)
        .Take(10)
        .Select(x => x.Nationality)
        .ToList();
// Фільтруємо дані лише для топ-країн
var filteredData = rawData
    .Where(d => topNationalities.Contains(d.Nationality))
    .ToList();
// 2. Налаштування осей та кольорів
var categoryNames = new List<string> { "Man", "Woman", "Pairs", "Dance" };
var colorMap = new Dictionary<string, OxyPlot.OxyColor>
{
    { "Man", OxyPlot.OxyColor.FromRgb(31, 119, 180) }, // Синій
    { "Woman", OxyPlot.OxyColor.FromRgb(255, 127, 14) }, // Помаранчевий
    { "Pairs", OxyPlot.OxyColor.FromRgb(44, 162, 95) }, // Зелений
    { "Dance", OxyPlot.OxyColor.FromRgb(214, 39, 40) } // Червоний
};
var categoryAxis = new CategoryAxis
{
    Position = AxisPosition.Left,
    Title = "Country",
    // Прибрано Labels = topNationalities
};
// Додаємо мітки, використовуючи метод AddRange()
categoryAxis.Labels.AddRange(topNationalities);
var valueAxis = new LinearAxis
{
    Position = AxisPosition.Bottom,
    Title = "Number of Wins (Stacked)",
    MinimumMajorStep = 1,

```

```

        MajorStep = 1
};
model.IsLegendVisible = true; // Залишаємо це
// Створюємо та налаштовуємо легенду
var legend = new OxyPlot.Legends.Legend
{
    LegendPlacement = OxyPlot.Legends.LegendPlacement.Outside, // Розміщення
поза область графіка
    LegendPosition = OxyPlot.Legends.LegendPosition.TopRight, // У верхньому
правому куті
    LegendOrientation = OxyPlot.Legends.LegendOrientation.Horizontal, //
Горизонтальна орієнтація
    LegendBackground = OxyPlot.OxyColors.White,
    LegendBorder = OxyPlot.OxyColors.Black,
    LegendFontWeight = 700
};
model.Axes.Add(categoryAxis);
model.Axes.Add(valueAxis);
model.Legends.Add(legend); // Додаємо об'єкт легенди до моделі
model.IsLegendVisible = true;
// 3. Створення та заповнення серій (одна серія на КАТЕГОРІЮ)
foreach (var category in categoryNames)
{
    var series = new BarSeries
    {
        Title = category,
        IsStacked = true, // Виправлено: без зірочок
        FillColor = colorMap[category],
        StrokeThickness = 1
    };
    // Заповнюємо дані для поточної категорії, дотримуючись порядку країн
    for (int i = 0; i < topNationalities.Count; i++)
    {

```

```

        string nationality = topNationalities[i];
        // Шукаємо кількість перемог для цієї країни у цій категорії
        var wins = filteredData
            .FirstOrDefault(d => d.Nationality == nationality && d.Category
== category);

        // Якщо перемог немає, додаємо 0, щоб стовпчики були вирівняні
        series.Items.Add(new BarItem { Value = wins?.WinsCount ?? 0 });
    }
    model.Series.Add(series);
}
return model;
}

// Method to save all plots
public void GenerateAllPlots(string outputDirectory)
{
    if (!System.IO.Directory.Exists(outputDirectory))
        System.IO.Directory.CreateDirectory(outputDirectory);
    var plots = new Dictionary<string, PlotModel>
    {
        { "1_Wins_by_Nationality:_Category_Breakdown",
CreateWinsByNationalityStackedChart() },
        { "2_Winners_Bar_Chart", CreateWinnersBarChart() },
        { "3_Score_Progression", CreateScoreProgressionChart() },
        { "4_Top_Performers", CreateTopPerformersChart() },
        { "5_Participation_Trends", CreateParticipationTrendsChart() }
    };
    foreach (var kvp in plots)
    {
        var pngExporter = new PngExporter { Width = 1200, Height = 800 };
        var filePath = System.IO.Path.Combine(outputDirectory, $"{kvp.Key}.png");
        pngExporter.ExportToFile(kvp.Value, filePath);
        Console.WriteLine($"Saved: {filePath}");
    }
}

```

```

        }
    }
}
using Analysis.Models;
using Dapper;
using MySql.Data.MySqlClient;
using System.Data;
using static System.ComponentModel.Design.ObjectSelectorEditor;
using System.Windows.Forms;
namespace Analysis
{
    internal class DataAccess
    {
        private readonly string _connectionString;
        public DataAccess(string connectionString)
        {
            _connectionString = connectionString;
        }
        private IDbConnection GetConnection()
        {
            return new MySqlConnection(_connectionString);
        }
        // 1. Get winners (fpl=1) count by sportsman
        public List<WinnerStats> GetWinnersStatsByCategory()
        {
            using (var conn = GetConnection())
            {
                var query = @" SELECT
                                s1.name AS Name,
                                s2.name AS Name2,
                                c.name AS Category,"

```

```

        COUNT(*) AS WinsCount
    FROM
        result AS r
    JOIN
        sportsman AS s1 ON r.sportsman_id = s1.id
    LEFT JOIN
        sportsman AS s2 ON r.sportsman2_id = s2.id
    JOIN
        category AS c ON r.category_id = c.id
    WHERE
        r.fpl = 1
    GROUP BY
        s1.name,
        s2.name,
        c.name
    ORDER BY
        WinsCount DESC
    LIMIT 20";
    return conn.Query<WinnerStats>(query).ToList();
}
}
// 2. Average scores progression by year and category
public List<ScoreProgression> GetScoreProgressionByYear()
{
    using (var conn = GetConnection())
    {
        var query = @"
            SELECT
                comp.year as Year,
                ROUND(AVG(r.points), 2) AS AverageScore,
                cat.name as Category
            FROM result r

```

```

        JOIN competition comp ON r.competition_id = comp.id
        JOIN category cat ON r.category_id = cat.id
        GROUP BY comp.year, cat.name
        ORDER BY comp.year, cat.name";
    return conn.Query<ScoreProgression>(query).ToList();
}
}
// 3. Top performers by total points
public dynamic GetTopPerformersByPoints(int limit = 15)
{
    using (var conn = GetConnection())
    {
        var query = @"
            WITH RankedResults AS (
                SELECT
                    r.sportsman_id,
                    r.category_id,
                    r.points,
                    comp.name AS CompetitionName,
                    comp.year AS CompetitionYear,
                    ROW_NUMBER() OVER(PARTITION BY r.sportsman_id,
r.category_id ORDER BY r.points DESC) AS rn
                FROM
                    result r
                JOIN
                    competition comp ON r.competition_id = comp.id
            )
            SELECT
                s.name AS Name,
                cat.name AS Category,
                MAX(t1.points) AS BestScore,
                AVG(r.points) AS AverageScore,

```

```

        COUNT(r.points) AS CompetitionsCount,
        t1.CompetitionName,
        t1.CompetitionYear
    FROM
        result r
    JOIN
        sportsman s ON r.sportsman_id = s.id
    JOIN
        category cat ON r.category_id = cat.id
    JOIN
        RankedResults t1 ON t1.sportsman_id = s.id AND
t1.category_id = cat.id AND t1.rn = 1
    GROUP BY
        s.id, s.name, cat.name, t1.CompetitionName,
t1.CompetitionYear
    HAVING
        COUNT(r.points) >= 3
    ORDER BY
        BestScore DESC
    LIMIT @limit";
    return conn.Query(query, new { limit }).ToList();
}
}
// 4. Year-over-year participation trends
public dynamic GetParticipationTrends()
{
    using (var conn = GetConnection())
    {
        var query = @"
            SELECT
                comp.year as Year,
                cat.name as Category,

```

```

        COUNT(DISTINCT r.sportsman_id) as Participants
    FROM result r
    JOIN competition comp ON r.competition_id = comp.id
    JOIN category cat ON r.category_id = cat.id
    GROUP BY comp.year, cat.name
    ORDER BY comp.year, cat.name";
    return conn.Query(query).ToList();
}
}
// 5. Competition difficulty (score spread)
public dynamic GetCompetitionDifficulty()
{
    using (var conn = GetConnection())
    {
        var query = @"
            SELECT
                comp.name as Competition,
                comp.year as Year,
                MAX(r.points) - MIN(r.points) as ScoreSpread,
                STDDEV(r.points) as ScoreStdDev
            FROM result r
            JOIN competition comp ON r.competition_id = comp.id
            JOIN category cat ON r.category_id = cat.id
            WHERE cat.name = 'Man'
            GROUP BY comp.id, comp.name, comp.year
            HAVING COUNT(*) >= 5
            ORDER BY comp.year DESC";
        return conn.Query(query).ToList();
    }
}
// 6. Wins By Nationality And Category
public List<NationalityCategoryWins> GetWinsByNationalityAndCategory()

```

```

    {
        using (var conn = GetConnection())
        {
            var query = @"
SELECT
    s1.nationality AS Nationality,
    c.name AS Category,
    COUNT(*) AS WinsCount
FROM
    result AS r
JOIN
    sportsman AS s1 ON r.sportsman_id = s1.id
JOIN
    category AS c ON r.category_id = c.id
WHERE
    r.fpl = 1 -- Фільтруємо лише переможців (перше місце)
GROUP BY
    s1.nationality,
    c.name
ORDER BY
    Nationality, WinsCount DESC"; // Заовлення для кращої візуалізації
return conn.Query<NationalityCategoryWins>(query).ToList();
        }
    }
}

namespace Analysis.Models
{
    internal class CompetitionStats
    {
        public string Competition { get; set; }
        public double AverageScore { get; set; }
    }
}

```

```

        public int ParticipantsCount { get; set; }
    }
}
using System;
using System.Collections.Generic;
using System.Text;

namespace Analysis.Models
{
    public class NationalityCategoryWins
    {
        public string Nationality { get; set; }
        public string Category { get; set; }
        public int WinsCount { get; set; }
    }
}
namespace Analysis.Models
{
    internal class NationalityStats
    {
        public string Nationality { get; set; }
        public int MedalsCount { get; set; }
        public string MedalType { get; set; }
    }
}
namespace Analysis.Models
{
    internal class ScoreProgression
    {
        public int Year { get; set; }
        public double AverageScore { get; set; }
        public string Category { get; set; }
    }
}

```

```

    }
}
namespace Analysis.Models
{
    internal class WinnerStats
    {
        public string Name { get; set; }
        public string Name2 { get; set; }
        public int WinsCount { get; set; }
        public string Category { get; set; }
    }
}
using OxyPlot;
using OxyPlot.WindowsForms;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace Analysis
{
    public partial class PlotsViewForm : Form
    {
        private ComboBox plotSelector;
        private PlotView plotView;
        private GraphGenerator generator;
        private Button saveButton;
        private Button saveAllButton;
        public PlotsViewForm(string connectionString)
        {

```

```

        generator = new GraphGenerator(connectionString);
        InitializeComponents();
        LoadFirstPlot();
    }
    private void InitializeComponents()
    {
        this.Text = "Figure Skating Competition Analysis";
        this.Size = new Size(1400, 900);
        this.StartPosition = FormStartPosition.CenterScreen;
        // Create panel for controls
        var controlPanel = new Panel
        {
            Dock = DockStyle.Top,
            Height = 50,
            BackColor = Color.LightGray
        };
        // Plot selector combo box
        plotSelector = new ComboBox
        {
            Location = new Point(10, 12),
            Size = new Size(400, 25),
            DropDownStyle = ComboBoxStyle.DropDownList
        };
        plotSelector.Items.AddRange(new string[]
        {
            "Wins by Nationality: Category Breakdown",
            "Winners Bar Chart",
            //"Medal Distribution by Nationality",
            "Score Progression Over Years",
            //"Competition Statistics Bubble Chart",
            "Top Performers by Best Score",
            //"Short Program vs Free Skate Correlation",

```

```

        "Participation Trends Area Chart"
        //"Competition Difficulty Analysis",

});
plotSelector.SelectedIndexChanged += PlotSelector_SelectedIndexChanged;
// Save current plot button
saveButton = new Button
{
    Text = "Save Current Plot",
    Location = new Point(420, 10),
    Size = new Size(120, 30)
};
saveButton.Click += SaveButton_Click;
// Save all plots button
saveAllButton = new Button
{
    Text = "Save All Plots",
    Location = new Point(550, 10),
    Size = new Size(120, 30),
    BackColor = Color.LightGreen
};
saveAllButton.Click += SaveAllButton_Click;
controlPanel.Controls.Add(plotSelector);
controlPanel.Controls.Add(saveButton);
controlPanel.Controls.Add(saveAllButton);
// Plot view
plotView = new PlotView
{
    Dock = DockStyle.Fill,
    BackColor = Color.White
};
this.Controls.Add(plotView);

```

```

        this.Controls.Add(controlPanel);
    }
    private void LoadFirstPlot()
    {
        plotSelector.SelectedIndex = 0;
    }
    private void PlotSelector_SelectedIndexChanged(object sender, EventArgs e)
    {
        PlotModel model = null;

        switch (plotSelector.SelectedIndex)
        {
            case 0:
                model = generator.CreateWinsByNationalityStackedChart();
                break;
            case 1:
                model = generator.CreateWinnersBarChart();

                break;
            case 2:
                model = generator.CreateScoreProgressionChart();

                break;
            case 3:
                model = generator.CreateTopPerformersChart();

                break;
            case 4:
                model = generator.CreateParticipationTrendsChart();
                break;
        }
        if (model != null)

```

```

    {
        plotView.Model = model;
    }
}
private void SaveButton_Click(object sender, EventArgs e)
{
    using (var saveDialog = new SaveFileDialog())
    {
        saveDialog.Filter = "PNG Image|*.png|SVG Image|*.svg";
        saveDialog.FileName = $"{plotSelector.Text.Replace(" ", "_")}.png";
        if (saveDialog.ShowDialog() == DialogResult.OK)
        {
            if (saveDialog.FilterIndex == 1) // PNG
            {
                var pngExporter = new PngExporter { Width = 1200, Height = 800 };
                pngExporter.ExportToFile(plotView.Model, saveDialog.FileName);
            }
            else // SVG
            {
                var svgExporter = new OxyPlot.SvgExporter { Width = 1200, Height
= 800 };
                svgExporter.ExportToFile(plotView.Model, saveDialog.FileName);
            }
            MessageBox.Show($"Plot saved to {saveDialog.FileName}", "Success",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
}
private void SaveAllButton_Click(object sender, EventArgs e)
{
    using (var folderDialog = new FolderBrowserDialog())
    {

```

```
folderDialog.Description = "Select folder to save all plots";
if (folderDialog.ShowDialog() == DialogResult.OK)
{try{generator.GenerateAllPlots(folderDialog.SelectedPath);
    MessageBox.Show($"All plots saved to
{folderDialog.SelectedPath}", "Success", MessageBoxButtons.OK,
MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Error saving plots: {ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}}
```