НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В. о. завідувача кафедри
_____ Артем ВОЛОКИТА
(підпис)
“__” _____2026 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”

на тему: Клієнт-серверна система обробки та прогнозування результатів командних матчів на основі статистичних метрик

Виконав: студент __4__ курсу, групи _ІО-23_
(шифр групи)

_____Васильєв Владислав Андрійович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____асистент, Хмельницький А. А.____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ас., д-р. філос. Коренко Д. В._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ - 2026 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти - перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри
_____ Артем ВОЛОКИТА
(підпис)
“__” _____2026 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Васильєва Владислава Андрійовича

1. Тема проєкту *Клієнт-серверна система обробки та прогнозування результатів командних матчів на основі статистичних метрик*
керівник проєкту Хмельницький Арсеній Андрійович, асистент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 3 червня 2026 року №2056-с

2. Термін здачі студентом закінченого проєкту *1 червня 2026 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Аналіз предметної області та постановка задачі.

Розділ 2. Вибір методів та технологій реалізації.

Розділ 3. Проектування та реалізація системи.

Розділ 4. Аналіз результатів.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) алгоритм програми, діаграма компонентів, діаграма варіантів використання.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Коренко Д. В.		

7. Дата видачі завдання «30» серпня 2025 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*5.01.2026-15.01.2026*	
2.	*Вивчення та аналіз завдання*	*16.01.2026-29.03.2026*	
3.	*Розробка архітектури та загальної структури системи*	*30.03.2026-8.04.2026*	
4.	*Розробка структур окремих підсистем*	*9.04.2026-24.04.2026*	
5.	*Програмна реалізація системи*	*25.04.2026-16.05.2026*	
6.	*Оформлення пояснювальної записки*	*17.05.2026-31.05.2026*	
7.	*Захист програмного продукту*	*23.05.2026*	
8.	*Передзахист*	*21.05.2026*	
9.	*Захист*	*15.06.2026*	

Студент-дипломник _____ Васильєв Владислав
(підпис)

Керівник проєкту _____ Арсеній Хмельницький
(підпис)

АНОТАЦІЯ

У бакалаврському дипломному проєкті розроблено клієнт-серверну систему обробки та прогнозування результатів командних матчів на основі статистичних метрик (на прикладі кіберспортивної дисципліни Dota 2).

Програмний комплекс забезпечує автоматизований збір ігрової статистики за допомогою відкритого інтерфейсу OpenDota API, структурування інформації у локальній базі даних та точний розрахунок ключових метрик. Для прогнозування результатів майбутніх протистоянь реалізовано та інтегровано алгоритми машинного навчання.

Програмний продукт створено з використанням мови програмування Python. Серверна частина та веб-інтерфейс розроблені на базі сучасного фреймворку FastAPI. Розроблений веб-додаток дозволяє користувачам наочно переглядати зібрану статистику та порівнювати ефективність і результати роботи кожного із застосованих методів машинного навчання.

ANNOTATION

In the bachelor's diploma project, a client-server system for processing and predicting team match results based on statistical metrics (using the Dota 2 esports discipline as an example) has been developed.

The software complex provides automated gathering of game statistics via the OpenDota API, structuring information in a local database, and precise calculation of key metrics. Machine learning algorithms are implemented and integrated to forecast the outcomes of future matches.

The software product is created using the Python programming language. The server-side and web interface are developed based on the modern FastAPI framework. The developed web application allows users to visually review the collected statistics and compare the effectiveness and results of each applied machine learning method.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Клієнт-серверна система	4		
			обробки та прогнозування			
			результатів командних			
			матчів на основі			
			статистичних метрик			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Пояснювальна записка	67		
	А4	***ІАЛЦ.467200.004 Д1***	Алгоритм програми	1		
	А4	***ІАЛЦ.467200.005 Д2***	Діаграма компонентів	1		
	А4	***ІАЛЦ.467200.006 Д3***	Діаграма варіантів	1		
			використання			
	А4	***ІАЛЦ.467200.007 Д4***	Текст програмного коду	72		

Зм	*Лист*	*№ докум.*	*Підп*	*Дата*	***ІАЛЦ.467200.001 ОА***		
Розроб		Васильєв В.А.			*Клієнт-серверна система обробки та прогнозування результатів командних матчів на основі статистичних метрик* Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Хмельницький А.А.					
						НТУУ “КПІ” ФІОТ ІО-23	

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Клієнт-серверна система обробки та прогнозування результатів командних матчів на основі статистичних метрик*»

Київ - 2026

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Це технічне завдання встановлює вимоги до розробки клієнт-серверної системи обробки статистики та прогнозування результатів командних матчів.

Програмний продукт застосовується для збору статистичних даних, їх аналізу та прогнозування того, яка з команд переможе. Система може використовуватися для наочного порівняння ефективності різних алгоритмів машинного навчання на практиці.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського дипломного проєкту за спеціальністю 123 «Комп'ютерна інженерія». Робота виконується на кафедрі обчислювальної техніки факультету інформатики та обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою роботи є створення системи, яка збирає ігрову статистику, обробляє її та прогнозує переможця матчу на основі складу команд.

Призначення розробки - надати користувачам зручний веб-інтерфейс, де можна не лише отримати прогноз результату, але й одразу порівняти, як із цим завданням справляються різні алгоритми машинного навчання.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є офіційна документація до використаних технологій (Python, фреймворк FastAPI, СУБД SQLite), інструкції по роботі з публічним API для збору даних, а також література та інтернет-ресурси з машинного навчання й аналізу даних.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має відповідати таким вимогам:

- Автоматизований збір матчів рейтингового режиму Dota 2 через OpenDota API з фільтрацією за якісними критеріями (ігровий режим, тривалість, рівень гравців).
- Зберігання даних матчів та довідкової інформації про героїв у реляційній базі даних.
- Розрахунок статистичних метрик: winrate та pickrate героїв, синергії пар героїв, контрпіків між героями різних команд.
- Навчання та порівняння чотирьох методів машинного навчання для прогнозування результату матчу за складом команд.
- Відображення результатів прогнозування від усіх моделей з метриками якості (Accuracy, ROC-AUC).
- Веб-інтерфейс з можливістю перегляду статистики героїв, фільтрації за атрибутом та роллю, а також введення складу команд для отримання прогнозу.
- Можливість автоматичного завантаження складу команд за ідентифікатором завершеного матчу.
- Простий та інтуїтивно зрозумілий інтерфейс користувача.

5.2. Вимоги до програмного забезпечення

- Операційна система: Windows 10/11, macOS або Linux.
- Python версії 3.10 або вище.
- Веб-браузер з підтримкою HTML5 та JavaScript (Chrome, Firefox, Edge).
- Пакети: FastAPI, uvicorn, scikit-learn, XGBoost, requests, Jinja2, numpy, python-dotenv.

5.3. Вимоги до апаратної частини

- Процесор: не менше ніж Intel Core i3 або еквівалент.
- Оперативна пам'ять: не менше ніж 8 ГБ (рекомендовано 16 ГБ для навчання моделей).
- Місце на диску: не менше ніж 5 ГБ для зберігання бази даних та моделей.
- Стабільне підключення до мережі Інтернет для збору даних через API.

6 ЕТАПИ РОЗРОБКИ

Назва етапу	Термін виконання
Аналіз предметної області	5.01.2026-29.03.2026
Проектування системи	30.03.2026-8.04.2026
Розробка модуля збору даних	9.04.2026-16.04.2026
Розробка модуля статистики	17.04.2026-24.04.2026
Розробка ML модуля	25.04.2026-1.05.2026
Розробка веб-інтерфейсу	2.05.2026-16.05.2026
Тестування системи	17.05.2026-24.05.2026
Оформлення пояснювальної записки	31.05.2026

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Клієнт-серверна система обробки та прогнозування результатів командних матчів на основі статистичних метрик*»

Київ - 2026

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД	База даних
API	Application Programming Interface - програмний інтерфейс застосунку
HTML	HyperText Markup Language - мова розмітки гіпертексту
HTTP	HyperText Transfer Protocol - протокол передачі гіпертексту
JSON	JavaScript Object Notation - текстовий формат обміну даними
LR	Logistic Regression - логістична регресія
MLP	Multilayer Perceptron - багатошаровий персептрон
MOBA	Multiplayer Online Battle Arena - жанр багатокористувацьких онлайн-ігор
ML	Machine Learning - машинне навчання
MMR	Matchmaking Rating - рейтинг підбору гравців у Dota 2
RF	Random Forest - метод випадкового лісу
ROC-AUC	Receiver Operating Characteristic - Area Under Curve - площа під кривою ROC
REST	Representational State Transfer - архітектурний стиль веб-сервісів
SQL	Structured Query Language - мова структурованих запитів
SQLite	Self-contained SQL database engine - вбудована реляційна СУБД
URL	Uniform Resource Locator - уніфікований локатор ресурсу
XGB	XGBoost - eXtreme Gradient Boosting, метод градієнтного бустингу

ВСТУП

Кіберспорт є одним із найбільш динамічно зростаючих секторів цифрової індустрії. Глобальна аудиторія кіберспортивних змагань щороку збільшується, а популярність командних онлайн-ігор виходить далеко за межі розважального середовища, формуючи окремий ринок аналітики, прогнозування та стратегічного планування. Серед провідних кіберспортивних дисциплін особливе місце займає Dota 2 - багатокористувацька командна гра жанру MOBA, розроблена компанією Valve Corporation, яка налічує понад 90 мільйонів зареєстрованих користувачів і стабільну щодобову аудиторію близько 600 тисяч одночасних гравців по всьому світу [1].

Специфікою Dota 2 є надзвичайно висока стратегічна складність: пул доступних персонажів налічує 127 одиниць, кожна з унікальним набором здібностей та ігрових ролей. Перед початком кожного рейтингового матчу команди проходять фазу драфту - почергового вибору та заборони героїв, яка визначає стратегічний потенціал обох сторін ще до початку безпосереднього протистояння. Загальна кількість можливих унікальних комбінацій складів двох команд перевищує $5{,}27 \times 10^{16}$ варіантів, що унеможливлює ручний аналіз та обумовлює необхідність застосування автоматизованих методів обробки даних.

Прогнозування результатів матчів на основі статистичних метрик є актуальною науковою задачею з практичним застосуванням. Гравці та кіберспортивні команди зацікавлені в інструментах, що допомагають оцінити силу складу команди, виявити оптимальні синергії між героями та передбачити ймовірний результат протистояння. Незважаючи на існування ряду аналітичних платформ (Dotabuff, OpenDota, Stratz), жодна з них не поєднує в єдиній системі комплексний розрахунок метрик взаємодії героїв із функціональністю прогнозування результату матчу на основі методів

машинного навчання.

Метою дипломного проєкту є розробка клієнт-серверної системи обробки та прогнозування результатів командних матчів Dota 2 на основі статистичних метрик. Для досягнення поставленої мети визначено такі завдання:

- розробити модуль автоматизованого збору даних рейтингових матчів через відкритий OpenDota API з фільтрацією за критеріями якості;
- спроектувати та реалізувати реляційну базу даних для структурованого зберігання матчів та довідкової інформації про героїв;
- розробити модуль розрахунку статистичних метрик: winrate, pickrate героїв, синергій пар героїв та контрпіків між командами;
- реалізувати та порівняти чотири методи машинного навчання, а також базовий (евристичний) підхід, що слугує точкою відліку для оцінювання їхньої ефективності;
- розробити веб-інтерфейс для відображення аналітики та надання прогнозів на основі FastAPI.

Об’єктом розробки є клієнт-серверна система аналізу та прогнозування результатів командних матчів Dota 2. Предметом розробки є методи й алгоритми обробки ігрової статистики та машинного навчання для задачі бінарної класифікації результату матчу за складом команд.

Практична цінність роботи полягає у створенні функціональної веб-системи, яка надає гравцям та аналітикам інструмент для оцінки сили драфту й порівняння ефективності різних підходів до прогнозування. Зібраний датасет обсягом 150 000 рейтингових матчів патчу 7.41 та навчені ML-моделі можуть бути використані як основа для подальших досліджень у галузі аналізу ігрових даних.

У першому розділі проведено аналіз предметної області: описано механіку гри Dota 2 та фазу драфту, обчислено комбінаторну складність задачі, здійснено огляд і порівняльний аналіз існуючих аналітичних платформ, сформульовано задачу дослідження. У другому розділі обґрунтовано вибір

методів машинного навчання та технологічного стеку, описано методологію формування вектора ознак. У третьому розділі описано проектування й реалізацію всіх компонентів системи: бази даних, модулів збору, статистики, машинного навчання та веб-інтерфейсу. У четвертому розділі наведено результати порівняння моделей, аналіз важливості ознак, динаміку точності від обсягу даних та результати тестування системи.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальний опис гри Dota 2

1.1.1 Правила та механіка гри

Dota 2 (Defense of the Ancients 2) - багатокористувацька онлайн-гра жанру MOBA, розроблена та видана компанією Valve Corporation у 2013 році [16]. Станом на 2026 рік гра залишається однією з найпопулярніших у світі: кількість зареєстрованих користувачів перевищує 90 мільйонів, а пікова одночасна аудиторія перевищує 600 тисяч гравців і під час великих турнірів сягає понад 900 тисяч осіб [1].

Базова механіка гри передбачає протистояння двох команд по п'ять гравців кожна. Команди отримують назви відповідно до розташування на карті: Radiant (світла сторона, нижній лівий кут) та Dire (темна сторона, верхній правий кут). Метою кожної команди є знищення головної споруди суперника - Ancient, розташованої в його базі.

Ігрове поле (карта) є симетричною і поділяється на три основні лінії: верхню, середню та нижню, між якими розташований ліс з нейтральними монстрами. Матч триває в середньому 35-40 хвилин, хоча тривалість може варіюватися від 15 до понад 90 хвилин залежно від рівня гравців та обраної стратегії.

Кожен гравець керує персонажем, що називається героєм. Герої мають унікальний набір характеристик, активних здібностей та пасивних ефектів. Протягом матчу герої отримують досвід та золото, що дозволяє підвищувати

рівень, вивчати нові здібності та купувати предмети для підсилення. Взаємодія героїв, предметів та ігрових механік утворює надзвичайно складну стратегічну систему.

1.1.2 Фаза драфту та її стратегічне значення

Перед початком кожного рейтингового матчу проводиться фаза вибору героїв, яка в ігровому середовищі називається драфтом (від англ. draft - відбір). Ця фаза є критично важливою складовою стратегії, оскільки визначає потенційні сильні та слабкі сторони кожної команди ще до початку протистояння.

Драфт у рейтинговому режимі Ranked All Pick відбувається у два етапи. На основі персональних списків заборони гравців система вилучає з гри 10 персонажів (по одному для кожного учасника матчу). На другому етапі команди по черзі обирають (пікають) по п'ять героїв зі списку доступних персонажів. За обраного героя може зіграти тільки один гравець - повторний вибір того самого героя обома командами неможливий.

Стратегічна цінність драфту визначається кількома факторами. По-перше, синергізм - деякі комбінації героїв підсилюють одне одного: наприклад, герої з уповільненнями дозволяють союзникам з великими area-of-effect здібностями ефективніше їх застосовувати. По-друге, контрпікінг - певні герої мають механічні переваги над конкретними суперниками через особливості своїх здібностей. По-третє, відповідність ролям - команда повинна мати збалансований склад, що покриває ключові ігрові ролі: carry (основне джерело шкоди на пізній стадії гри), support (підтримка союзників), initiator (ініціатор командних боїв) тощо.

Професійні кіберспортивні команди витрачають значні ресурси на підготовку до фази драфту: аналізують статистику суперників, розробляють комбінації героїв та готують стратегії контрпікінгу. Це свідчить про те, що результат матчу значною мірою визначається вже на стадії вибору складу.

1.1.3 Комбінаторна складність задачі вибору складу

Пул доступних героїв у Dota 2 станом на патч 7.41 налічує 127 персонажів. Кожна команда обирає 5 героїв без повторень, при цьому герої, обрані однією командою, недоступні для іншої. Кількість можливих унікальних комбінацій складів двох команд обчислюється за формулою (1.1):

$$N = C(127,5) \times C(122,5), \quad (1.1)$$

де $C(n,k)$ - біноміальний коефіцієнт, що визначає кількість способів обрати k елементів з n без повторень і без урахування порядку, який обчислюється за формулою (1.2):

$$C(n,k) = \frac{n!}{k!\,(n-k)!}, \quad (1.2)$$

Підставляючи значення $n = 127$ та $k = 5$ у формулу (1.2), отримуємо кількість варіантів складу першої команди, а для $n = 122$ - кількість варіантів складу другої:

$$C(127,5) = \frac{127!}{5! \cdot 122!} = 254\,231\,775$$

$$C(122,5) = \frac{122!}{5! \cdot 117!} = 207\,288\,004$$

Загальна кількість можливих унікальних пар складів двох команд згідно з формулою (1.1):

$$N = 254\,231\,775 \times 207\,288\,004 \approx 5{,}27 \times 10^{16}$$

Таким чином, загальна кількість можливих унікальних пар складів команд перевищує 52 квадрильйони варіантів.

Така кількість комбінацій робить вичерпний перебір усіх можливих складів неможливим навіть для сучасних обчислювальних систем. Досвідчений гравець здатний утримувати в пам’яті інформацію про декілька десятків стандартних комбінацій, однак систематичний аналіз взаємодій між усіма можливими складами команд вимагає застосування статистичних

методів та алгоритмів машинного навчання. Саме цим обумовлюється актуальність розробки автоматизованих систем аналізу й прогнозування.

1.2 Огляд існуючих аналітичних сервісів

1.2.1 Dotabuff

Dotabuff [2] є найбільш популярним сервісом статистики для Dota 2 з аудиторією понад 10 мільйонів зареєстрованих користувачів. Платформа надає детальну статистику по героях (winrate, pickrate, банрейт у розбивці за рейтинговими категоріями та патчами), аналіз профілів гравців (рейтинг, статистика по героях, останні матчі) і загальну мета-аналітику поточного патчу.

Серед переваг Dotabuff слід виділити зручний інтерфейс, актуальність даних та широку функціональність безкоштовної версії. Проте сервіс має суттєве обмеження: він надає виключно дескриптивну статистику й не пропонує жодних інструментів прогнозування результатів матчів чи рекомендацій щодо вибору оптимального складу. Розширені функції (детальна аналітика матчів, порівняння гравців) доступні лише за платною підпискою.

1.2.2 OpenDota

OpenDota [3] - відкрита платформа з вихідним кодом, що надає доступ до бази даних матчів Dota 2 через публічний API. На відміну від Dotabuff, OpenDota орієнтована насамперед на розробників та дослідників даних, надаючи програмний доступ до детальної інформації про матчі, гравців і героїв.

Публічний API OpenDota дозволяє отримувати дані про мільйони матчів

без реєстрації, хоча й з обмеженням на кількість запитів (60 запитів на хвилину без API-ключа). Веб-інтерфейс платформи за функціональністю поступається Dotabuff, однак відкритість API робить OpenDota основним джерелом даних для наукових досліджень у галузі аналізу Dota 2. У даній роботі OpenDota використовується як єдине джерело даних для наповнення бази.

1.2.3 Stratz

Stratz [4] - аналітична платформа з розширеними можливостями візуалізації та аналізу даних. Серед унікальних функцій сервісу: аналіз синергій героїв у розбивці за позиціями, відстеження ефективності предметів, детальна аналітика командних боїв і графіки динаміки переваги протягом матчу.

Stratz надає глибший аналіз порівняно з конкурентами, однак більшість розширених функцій доступні лише за платною підпискою. Публічний API Stratz має суттєвіші обмеження порівняно з OpenDota, що ускладнює його використання для масового збору даних. Функціональність прогнозування результатів матчів на платформі також відсутня.

1.2.4 Порівняльний аналіз та недоліки існуючих рішень

Узагальнений порівняльний аналіз розглянутих сервісів наведено в таблиці 1.1.

Таблиця 1.1 - Порівняльний аналіз аналітичних сервісів Dota 2

Критерій	Dotabuff	OpenDota	Stratz	Розроблена система
Winrate / Pickrate героїв	Так	Так	Так	Так
Синергії героїв	Частково	Ні	Так	Так
Контрпіки	Частково	Ні	Так	Так
Прогнозування результату	Ні	Ні	Ні	Так

Кінець таблиці 1.1

Критерій	Dotabuff	OpenDota	Stratz	Розроблена система
Порівняння ML методів	Ні	Ні	Ні	Так
Безкоштовний доступ	Частково	Так	Частково	Так

Аналіз існуючих рішень виявив спільний недолік усіх розглянутих платформ: жодна з них не поєднує в єдиній системі комплексний розрахунок статистичних метрик взаємодії героїв (синергій та контрпіків) з функціональністю прогнозування результату матчу на основі методів машинного навчання. Крім того, жоден із сервісів не надає можливості порівняння ефективності різних алгоритмів прогнозування, що є важливим як з наукової, так і з практичної точки зору.

1.3 Постановка задачі

1.3.1 Формулювання задачі класифікації

На основі проведеного аналізу предметної області задача роботи формулюється так: розробити автоматизовану систему, що на основі складу двох команд (драфту) передбачає переможця матчу Dota 2.

З погляду машинного навчання це задача бінарної класифікації. Нехай $X \in \mathbb{R}^n$ - вектор ознак, що описує склад обох команд, а $y \in \{0,1\}$ - результат матчу (1 - перемога Radiant, 0 - перемога Dire). Необхідно побудувати функцію $f: \mathbb{R}^n \to \{0,1\}$, яка мінімізує помилку класифікації на незалежній тестовій вибірці.

Ключовою особливістю поставленої задачі є обмеженість вхідних даних: прогноз будується виключно на основі інформації, доступної до початку матчу, тобто лише на складі команд. Такий підхід, на відміну від прогнозування за динамічними метриками матчу (золото та досвід за хвилинами), дозволяє отримати прогноз одразу після завершення фази драфту.

Теоретична межа точності для pick-based прогнозу в Dota 2 оцінюється дослідниками на рівні 57-60% [5, 6]. Решта варіативності результатів визначається індивідуальною майстерністю гравців, мікрорішеннями під час матчу та випадковими факторами, які не можуть бути передбачені на основі лише складу команд. Це принципово відрізняє задачу від детермінованих задач класифікації і є важливим контекстом для інтерпретації результатів.

1.3.2 Вхідні та вихідні дані системи

Вхідними даними системи є ідентифікатори десяти героїв - по п'ять для кожної команди. Кожен герой має унікальний числовий ідентифікатор у системі Dota 2 (від 1 до 155 з пропусками для видалених персонажів). Порядок вибору героїв у межах команди не враховується.

На основі вхідних даних система формує вектор ознак, що включає:

- категоріальний компонент - бінарне one-hot кодування присутності кожного героя в кожній команді;
- числовий компонент - агреговані статистичні метрики: середній, максимальний та мінімальний winrate героїв команд, середній pickrate, різниці відповідних показників між командами, а також метрики синергій та контрпіків.

Вихідними даними системи є ймовірність перемоги команди Radiant, обчислена кожною з реалізованих моделей, та бінарний прогноз переможця. Додатково система відображає метрики якості кожної моделі (Accuracy, ROC-AUC), отримані на тестовій вибірці під час навчання.

1.3.3 Критерії оцінювання якості

Для оцінювання ефективності реалізованих методів машинного навчання використовуються дві основні метрики.

Accuracy (точність класифікації) - частка правильно класифікованих матчів серед усіх матчів тестової вибірки. Для бінарної класифікації accuracy обчислюється за формулою (1.3):

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}, \quad (1.3)$$

де TP - кількість правильно передбачених перемог Radiant; TN - правильно передбачених перемог Dire; FP - хибно передбачених перемог Radiant; FN - хибно передбачених перемог Dire.

ROC-AUC (площа під кривою ROC) [17] - метрика оцінювання якості бінарного класифікатора, що визначається як площа під кривою ROC. Криву ROC будують шляхом зміни порогу класифікації від 0 до 1 та відображення залежності між часткою правильно визначених позитивних прикладів (True Positive Rate) і часткою хибно визначених негативних прикладів (False Positive Rate). Значення ROC-AUC = 0,5 відповідає випадковому прогнозу, ROC-AUC = 1,0 - ідеальному класифікатору. Перевагою ROC-AUC порівняно з Accuracy є стійкість до дисбалансу класів, що є актуальним для даної задачі (розподіл результатів 53,4% / 46,6% на користь Radiant).

Додатковим критерієм оцінювання є порівняння з базовою моделлю (Baseline), яка не використовує машинного навчання: прогноз визначається виключно на основі середнього winrate героїв кожної команди.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі проведено детальний аналіз предметної області - кіберспортивної дисципліни Dota 2. Розглянуто основні правила та механіку гри, описано стратегічне значення фази драфту як ключового етапу, що визначає потенціал команди ще до початку матчу.

Обчислено комбінаторну складність задачі вибору складу: загальна кількість можливих унікальних пар складів команд перевищує $5{,}27 \times 10^{16}$ варіантів, що унеможливлює ручний аналіз і обумовлює необхідність застосування автоматизованих методів обробки даних.

Проведено огляд та порівняльний аналіз трьох провідних аналітичних платформ - Dotabuff, OpenDota і Stratz. Встановлено, що жодна з існуючих систем не поєднує комплексний розрахунок метрик взаємодії героїв із функціональністю прогнозування результату матчу на основі методів машинного навчання.

Сформульовано задачу роботи як задачу бінарної класифікації: за складом двох команд передбачити переможця матчу. Визначено вхідні та вихідні дані системи, а також критерії оцінювання якості реалізованих моделей (Accuracy та ROC-AUC) з обґрунтуванням їх вибору.

РОЗДІЛ 2
ВИБІР МЕТОДІВ ТА ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ

2.1 Методи машинного навчання для бінарної класифікації

Вибір методів машинного навчання для задачі прогнозування результату матчу за складом команд визначається специфікою вхідних даних. Вектор ознак містить переважно бінарні категоріальні ознаки (one-hot кодування складу команд) та відносно невелику числову компоненту зі статистичними метриками взаємодій. Такий тип даних має конкретні наслідки для поведінки різних алгоритмів: лінійні методи можуть не виявити нелінійних взаємодій між героями, деревовидні методи природно працюють із бінарними ознаками без нормалізації, а нейронні мережі вимагають додаткової регуляризації.

Крім того, задача має дисбаланс класів 53,4% на користь Radiant, що пов’язано з асиметрією карти. Цей дисбаланс невеликий, але достатній, щоб вплинути на поведінку деяких методів без коригування ваг класів. У роботі реалізовано й порівняно п’ять підходів, що охоплюють весь спектр складності - від детерміністичного правила без навчання до нейронної мережі. Такий діапазон дозволяє зробити обґрунтовані висновки про те, які властивості алгоритмів є визначальними для цієї конкретної задачі.

2.1.1 Базова модель (Baseline)

Перш ніж застосовувати методи машинного навчання, необхідно визначити базовий рівень прогнозування - нижню межу якості, відносно якої оцінюватиметься ефективність складніших алгоритмів. Базова модель є детерміністичним правилом класифікації, що не потребує навчання й тому позбавлене ризику перенавчання. Якщо ML-модель не перевищує цей рівень,

вона не має практичної цінності незалежно від своєї складності.

У даній роботі базова модель визначає переможця, порівнюючи середні winrate героїв обох команд. Winrate кожного героя - це частка матчів, у яких команда з цим героєм перемогла; він є накопиченою статистикою ефективності персонажа в поточному мета-грі. Команда, де цей показник у середньому вищий, отримує прогноз перемоги. Ймовірність перемоги команди Radiant обчислюється за формулою (2.1):

$$P(\text{Radiant}) = \frac{\overline{WR}_R}{\overline{WR}_R + \overline{WR}_D}, \quad (2.1)$$

де $\overline{WR}_R$ - середній winrate п’яти героїв команди Radiant; $\overline{WR}_D$ - середній winrate п’яти героїв команди Dire. Якщо $P(\text{Radiant}) \geq 0{,}5$, прогнозується перемога Radiant, інакше - Dire.

Базова модель є змістовним предиктором, а не тривіальним правилом: вона спирається на реальну статистику ефективності героїв у поточному патчі й вловлює головну закономірність - команди, що обирають сильніших у мета-грі персонажів, виграють частіше. Саме тому baseline є надійним еталоном: якщо ML-модель показує гірший результат, це сигнал, що вона не змогла виявити нічого понад простим усередненням winrate.

2.1.2 Логістична регресія

Логістична регресія є одним із фундаментальних методів статистичного навчання [18] для задач бінарної класифікації. Незважаючи на назву, метод є класифікатором: він моделює ймовірність приналежності об’єкта до класу через лінійну комбінацію ознак, трансформовану сигмоїдною функцією. Логістична регресія є природним першим кроком після baseline, оскільки перевіряє, чи є задача лінійно роздільною в поточному просторі ознак.

Для вектора ознак $x \in \mathbb{R}^n$ модель обчислює ймовірність перемоги Radiant за формулою (2.2):

$$P(y = 1 \mid x) = \sigma(w^T x + b) = \frac{1}{1+e^{-(w^T x+b)}}, \quad (2.2)$$

де $w \in \mathbb{R}^n$ - вектор ваг, що відповідає кожній ознаці; b - зсув (bias); w^T - транспонований вектор ваг; $\sigma(\cdot)$ - сигмоїдна функція, що відображає будь-яке дійсне число у відрізок $(0,1)$. Геометрично модель будує гіперплощину у просторі ознак, що розділяє матчі з перемогою Radiant від матчів з перемогою Dire. Усі вхідні приклади, що опинилися по один бік від цієї гіперплощини, отримують однаковий прогноз незалежно від того, наскільки далеко вони від неї розташовані.

Параметри w і b знаходяться шляхом мінімізації функції логістичних втрат з L2-регуляризацією, наведеної у формулі (2.3):

$$J(w, b) = -\frac{1}{m}\sum_{i=1}^{m}[\, y_i \ln P_i + (1 - y_i)\ln(1 - P_i)\,] + \lambda \parallel w \parallel^2, \quad (2.3)$$

де m - розмір тренувальної вибірки; $y_i \in \{0,1\}$ - реальна мітка i-го прикладу; P_i - передбачена ймовірність для i-го прикладу; $\lambda = 1/C$ - коефіцієнт регуляризації. Перший член функції є логарифмічною правдоподібністю - він штрафує модель за невпевнені або неправильні прогнози. Другий член $\lambda \parallel w \parallel^2$ є L2-регуляризацією, що штрафує великі значення ваг.

У реалізації обрано $C = 0{,}1$, що відповідає $\lambda = 10$ - відносно сильній регуляризації. Це рішення є принциповим з урахуванням специфіки задачі. Вхідний простір містить 400 бінарних ознак, більшість яких є слабкими предикторами, оскільки кожна ознака несе дуже мало інформації окремо. Без достатньої регуляризації модель перенавчилася б на випадкових комбінаціях героїв, що зустрілися у тренувальній вибірці, але не є реальними закономірностями. Сильна регуляризація змушує модель спиратися виключно на ознаки, що стабільно корелюють з результатом по всьому датасету.

Числові ознаки нормалізуються за допомогою StandardScaler перед навчанням, оскільки логістична регресія є чутливою до масштабу вхідних даних. StandardScaler приводить кожну числову ознаку до нульового

середнього й одиничної дисперсії. Без нормалізації ознаки з різними масштабами отримують некоректні відносні ваги: градієнт по ознаці з великим діапазоном значень домінуватиме у процесі оптимізації, що погіршує збіжність і якість моделі. Категоріальні бінарні ознаки вже перебувають у діапазоні $\{0,1\}$ і нормалізації не потребують.

Логістична регресія відрізняється високою інтерпретованістю: ваговий коефіцієнт w_i безпосередньо відображає вплив ознаки x_i на прогноз. Позитивне значення w_i збільшує передбачувану ймовірність перемоги Radiant при зростанні x_i, негативне - зменшує. Це дозволяє після навчання проаналізувати, які ознаки вектора є найбільш значущими для лінійного класифікатора.

2.1.3 Випадковий ліс (Random Forest)

Random Forest є ансамблевим методом машинного навчання, запропонованим Лео Брейманом у 2001 році [7]. Метод вирішує фундаментальну проблему одиночного дерева рішень: одне дерево здатне точно описати тренувальні дані, але через надмірну чутливість до шуму у вибірці схильне до перенавчання - воно запам’ятовує специфічні приклади, а не узагальнені закономірності. Random Forest долає цю проблему через ансамблювання множини некорельованих дерев.

Алгоритм побудови ансамблю базується на двох техніках зниження дисперсії. Перша - bagging (Bootstrap Aggregating) [19]. Кожне з T дерев навчається на незалежній bootstrap-вибірці S_t розміром N, отриманій вибіркою з поверненням із тренувального набору. У середньому кожне дерево бачить приблизно 63,2% унікальних прикладів, а решта 36,8% є дублікатами. Приклади, що не потрапили до bootstrap-вибірки, утворюють Out-of-Bag (OOB) множину, яку можна використати для оцінки узагальнюючої здатності без окремої валідаційної вибірки.

Друга техніка - випадковий підпростір ознак. При побудові кожного вузла дерева випадково обирається підмножина розміром $\sqrt{n}$ ознак, де n - загальна кількість ознак. Для $n = 413$ це приблизно 20 ознак на кожне розбиття. Вузол розбивається за ознакою й порогом, що максимально зменшують критерій нечистоти Джині, наведений у формулі (2.4):

$$\text{Gini} = 1 - \sum_k p_k^2, \quad (2.4)$$

де p_k - частка прикладів класу k у вузлі. Вузол з чистим розподілом (усі приклади одного класу) має $\text{Gini} = 0$, вузол з рівним розподілом - $\text{Gini} = 0{,}5$.

Завдяки різним bootstrap-вибіркам і різним підмножинам ознак прогнози різних дерев є частково некорельованими. Фінальна ймовірність перемоги Radiant обчислюється усередненням по всіх деревах за формулою (2.5):

$$P_{RF}(y = 1 \mid x) = \frac{1}{T}\sum_{t=1}^{T} h_t(x), \quad (2.5)$$

де $T = 300$ - кількість дерев; $h_t(x) \in \{0,1\}$ - прогноз t-го дерева. Усереднення некорельованих прогнозів суттєво зменшує дисперсію ансамблю порівняно з одиночним деревом: за умови, що кожне дерево помиляється незалежно від інших, дисперсія усередненого прогнозу зменшується пропорційно до T.

Вибір гіперпараметрів є конкретним інженерним рішенням. Максимальна глибина дерева обмежена 15 рівнями. Без цього обмеження дерева ростуть до повного запам’ятовування тренувальних прикладів, що призводить до перенавчання навіть при усередненні. Мінімальна кількість прикладів у листовому вузлі - 5 - запобігає побудові вузлів, що спираються на одиничні приклади.

Параметр class_weight='balanced' коригує ваги класів обернено пропорційно до їх частот у тренувальній вибірці. Клас 1 (Radiant, 53,4%) отримує меншу вагу, ніж клас 0 (Dire, 46,6%), що компенсує дисбаланс і дозволяє моделі однаково точно класифікувати обидва класи. Без коригування модель схилялася б прогнозувати переважно перемогу Radiant.

Random Forest не потребує нормалізації ознак, оскільки дерева рішень при розбитті вузла порівнюють значення ознаки з порогом, а не використовують абсолютні значення. Монотонні перетворення вхідних даних (наприклад, масштабування) не змінюють оптимальний поріг розбиття. Паралельне навчання дерев реалізоване через параметр n_jobs=-1, що використовує всі доступні ядра процесора.

Важливою властивістю Random Forest є можливість обчислення важливості ознак. Для кожної ознаки розраховується середнє зменшення критерію Джині по всіх вузлах і деревах, де ця ознака використовувалася для розбиття. Ця метрика застосовується в розділі 4 для аналізу інформативності вектора ознак і виявлення найбільш значущих предикторів.

2.1.4 Градієнтний бустинг (XGBoost)

XGBoost (eXtreme Gradient Boosting) реалізує алгоритм градієнтного бустингу, запропонований Фрідманом (2001) [8] і оптимізований Тяньці Ченом та Карлосом Гестрином (2016) [9]. На відміну від Random Forest, де T дерев будуються незалежно й паралельно, градієнтний бустинг будує дерева послідовно: кожне наступне дерево навчається виправляти помилки ансамблю попередніх дерев.

Ідея бустингу полягає в тому, що навіть слабкі класифікатори (неглибокі дерева) можна послідовно поєднати у сильний. Алгоритм починає з константного прогнозу $F_0(x)$. На кожній ітерації t обчислюються псевдозалишки - негативний градієнт функції втрат відносно поточного прогнозу $F_{t-1}(x)$. Нове дерево r_t навчається апроксимувати ці псевдозалишки. Оновлення ансамблю відбувається за формулою (2.6):

$$F_t(x) = F_{t-1}(x) + \eta\, r_t(x), \tag{2.6}$$

де η - швидкість навчання (learning rate), що контролює внесок кожного нового дерева. Мала швидкість навчання ($\eta = 0{,}01$) означає, що кожне дерево

вносить невеликий крок у напрямку зменшення похибки. Це вимагає більшої кількості ітерацій, але забезпечує кращу узагальнюючу здатність.

XGBoost вирізняється тим, що мінімізує регуляризований функціонал, наведений у формулі (2.7):

$$\text{Obj} = \sum_i L\left(y_i, F(x_i)\right) + \sum_t \Omega\left(r_t\right), \quad (2.7)$$

де L - функція втрат (логістична для бінарної класифікації); $\Omega(r_t) = \gamma\, T_t + \frac{\lambda}{2} \parallel w_t \parallel^2$ - регуляризаційний член, що штрафує складність кожного дерева. Тут T_t - кількість листів t-го дерева; w_t - ваги листів; γ і λ - гіперпараметри. Включення регуляризації безпосередньо у функцію оптимізації є ключовою відмінністю XGBoost від класичного GBDT і однією з причин його практичної ефективності.

При побудові кожного нового дерева XGBoost використовує апроксимацію Тейлора другого порядку для функції втрат, що дозволяє точніше оцінити оптимальне значення ваги кожного листа й значення критерію для вибору розбиття. Це є математично суворішим підходом, ніж використання лише градієнта (першого порядку) у класичному GBDT.

Вибір гіперпараметрів обумовлений необхідністю запобігти перенавчанню при послідовному навчанні. Обрано 500 дерев з малою швидкістю навчання $\eta = 0{,}01$ - велика кількість слабких учнів дає кращу узагальнюючу здатність, ніж мала кількість сильних. Максимальна глибина дерева 6 є меншою, ніж у Random Forest (15): послідовний бустинг більш схильний до перенавчання на шумі, тому глибину обмежено жорсткіше. Параметри subsample = 0.7 і colsample_bytree = 0.7 вводять стохастичність, схожу на bagging, - кожне дерево навчається на 70% рядків і 70% ознак, що зменшує кореляцію між деревами. Сильна L2-регуляризація reg_lambda = 10 додатково стримує складність листів.

2.1.5 Багатошаровий персептрон (MLP)

Багатошаровий персептрон (MLP, Multilayer Perceptron) є базовою архітектурою штучної нейронної мережі прямого поширення. Теорема про універсальну апроксимацію (Хорнік, 1989) [10] стверджує, що MLP з одним прихованим шаром достатньої ширини може апроксимувати будь-яку неперервну функцію на компактній множині з довільною точністю. На практиці це означає, що мережа теоретично здатна виявляти як завгодно складні нелінійні залежності між ознаками.

Архітектура мережі у роботі складається з вхідного шару (413 нейронів, що відповідає розмірності вектора ознак), двох прихованих шарів (128 та 64 нейрони) і вихідного шару (1 нейрон із сигмоїдною активацією). Поступове зменшення ширини шарів ($413 \to 128 \to 64 \to 1$) реалізує принцип ієрархічного стиснення: кожен шар виділяє все більш абстрактні ознаки, зменшуючи розмірність представлення.

Для прихованих шарів обрано функцію активації ReLU (Rectified Linear Unit) [21], визначену формулою (2.8):

$$\mathrm{ReLU}(z) = \max(0, z), \tag{2.8}$$

Функція активації є принципово важливою складовою нейронної мережі: без нелінійної активації будь-яка кількість послідовних лінійних шарів еквівалентна одному лінійному шару, і мережа не може апроксимувати нелінійні функції. ReLU обрано замість класичних sigmoid або tanh з кількох причин. Похідна ReLU дорівнює 1 при $z > 0$ і 0 при $z < 0$, тоді як похідні sigmoid і tanh прямують до нуля при великих $|z|$ - це явище називається зникаючим градієнтом (vanishing gradient). При зворотному поширенні помилки градієнт множиться на похідну активаційної функції на кожному шарі, і при використанні sigmoid або tanh він стає настільки малим, що ваги ранніх шарів практично не оновлюються. ReLU вирішує цю проблему, оскільки градієнт не зникає для активних нейронів ($z > 0$). Крім того,

обчислення ReLU тривіальне (просте порівняння з нулем), що прискорює і прямий, і зворотний проходи.

Навчання здійснюється методом зворотного поширення помилки (backpropagation) з оптимізатором Adam (Adaptive Moment Estimation) [20]. На відміну від звичайного стохастичного градієнтного спуску (SGD), де однакова швидкість навчання застосовується до всіх параметрів, Adam адаптує швидкість навчання індивідуально для кожного параметра на основі перших двох моментів градієнта. Перший момент (середнє градієнта) дозволяє рухатися стабільно у правильному напрямку, а другий момент (середній квадрат градієнта) масштабує кроки відповідно до кривизни функції втрат. Це забезпечує стабільнішу і швидшу конвергенцію, ніж SGD.

Для запобігання перенавчанню застосовуються два незалежних механізми. L2-регуляризація з коефіцієнтом $\alpha = 0{,}01$ додає до функції втрат штрафний член, пропорційний квадрату норми ваг. Це змушує мережу спиратися на більшу кількість ознак, а не концентруватися на кількох, - розподілене представлення є більш узагальнюваним. Рання зупинка навчання (early stopping) зупиняє процес, якщо значення функції втрат на валідаційній вибірці (10% тренувальних даних) не покращується протягом 20 епох підряд. Це запобігає перенавчанню на пізніх ітераціях, коли мережа починає запам’ятовувати окремі тренувальні приклади.

Числові ознаки нормалізуються StandardScaler перед навчанням. Нейронні мережі, на відміну від деревовидних методів, є чутливими до масштабу вхідних даних: ваги ініціалізуються малими випадковими значеннями, і при великій різниці в масштабах ознак градієнти по різних вагах матимуть несумірні значення, що ускладнює оптимізацію.

2.1.6 Порівняльний аналіз методів

Вибір саме цього набору методів обумовлений необхідністю охопити весь спектр підходів: від простого детерміністичного правила (Baseline) та лінійного методу (LR) до потужних ансамблевих методів (RF, XGBoost) і нейронної мережі (MLP). Такий підбір дозволяє провести змістовне порівняння й зробити обґрунтовані висновки про оптимальний підхід для даної задачі. Теоретичні характеристики кожного методу, що є важливими для даної задачі, наведено в таблиці 2.1.

Таблиця 2.1 - Порівняльний аналіз методів машинного навчання

Критерій	Baseline	LR	RF	XGBoost	MLP
Тип моделі	Правило	Лінійна	Ансамбль	Ансамбль	Нейромережа
Нелінійність	Ні	Ні	Так	Так	Так
Потребує нормалізації	Ні	Так	Ні	Ні	Так
Інтерпретованість	Висока	Висока	Середня	Середня	Низька
Швидкість навчання	Миттєво	Висока	Середня	Середня	Середня
Ризик перенавчання	Відсутній	Низький	Низький	Середній	Середній
Feature importance	Ні	Так	Так	Так	Ні

Жоден з методів не є апріорно найкращим - вибір залежить від конкретних властивостей задачі й даних. Практичне порівняння на реальному датасеті завжди є ціннішим, ніж теоретичний аналіз. Результати порівняння наведено в розділі 4.

2.2 Вибір технологічного стеку

Вибір технологій для реалізації системи визначається трьома критеріями: наявністю зрілої екосистеми для машинного навчання й аналізу даних, можливістю побудови всіх компонентів у рамках єдиного стеку, що мінімізує складність інтеграції, та відкритістю і безкоштовністю компонентів.

Мова програмування Python. Python [15] обрано як основну мову для реалізації всіх компонентів системи. Вибір обумовлений домінуванням Python у галузі машинного навчання та аналізу даних. Екосистема бібліотек scikit-learn, XGBoost і NumPy надає готові оптимізовані реалізації всіх необхідних алгоритмів, що дозволяє зосередитися на вирішенні предметної задачі, а не на низькорівневій реалізації математичних операцій. Наявність веб-фреймворку FastAPI дозволяє розробити серверну частину тією самою мовою, що й ML-компонента, без необхідності міжмовної інтеграції. Вбудований модуль sqlite3 стандартної бібліотеки забезпечує взаємодію з базою даних без додаткових залежностей. Використовується версія Python 3.10+.

Фреймворк FastAPI. FastAPI [12] є сучасним асинхронним веб-фреймворком для Python, побудованим на Starlette та Pydantic [22]. На відміну від Flask, який є синхронним і при обробці одного запиту блокує сервер, FastAPI побудований на ASGI (Asynchronous Server Gateway Interface), що дозволяє конкурентно обробляти кілька запитів. Для цього проєкту, де перше обчислення статистики є ресурсомістким, асинхронна архітектура дозволяє серверу залишатися чутливим при паралельних запитах. Вбудована підтримка Jinja2-шаблонів [23] через клас Jinja2Templates дозволяє реалізувати серверний рендеринг HTML без додаткових залежностей. Декларативний синтаксис визначення роутерів через декоратори (@app.get, @app.post) забезпечує читабельність коду. Сервер запускається через uvicorn [24] - ASGI-сервер, оптимізований для FastAPI.

База даних SQLite. SQLite [13] обрано як систему управління базою даних для зберігання матчів і довідкової інформації. SQLite є вбудованою реляційною СУБД, що зберігає всі дані в єдиному файлі та не потребує окремого серверного процесу. Вибір між SQLite і повноцінними СУБД (PostgreSQL, MySQL) визначається масштабом задачі: при 150 000 рядків і сценарії, де один процес пише матчі раз на кілька діб, а веб-сервер читає статистику, SQLite є оптимальним рішенням. Відсутність необхідності налаштування й адміністрування СУБД, портабельність (весь датасет в одному файлі dota2.db) та вбудована підтримка в Python через модуль sqlite3 є додатковими перевагами.

Бібліотеки машинного навчання. Scikit-learn [11] є основною бібліотекою машинного навчання для Python, що надає уніфікований інтерфейс fit / predict / predict_proba для навчання й оцінювання алгоритмів класифікації. У проєкті використовуються: LogisticRegression, RandomForestClassifier та MLPClassifier для реалізації відповідних методів, train_test_split для розбивки вибірки, StandardScaler для нормалізації числових ознак, accuracy_score та roc_auc_score для оцінювання якості. XGBoost є окремою бібліотекою [9], що реалізує сумісний зі scikit-learn інтерфейс через клас XGBClassifier. NumPy [14] використовується для матричних операцій при побудові вектора ознак. Pickle - стандартний Python-модуль для серіалізації навчених моделей та допоміжних об’єктів у бінарний формат .pkl.

OpenDota API як джерело даних. OpenDota [3] надає відкритий REST API з відповідями у форматі JSON. Для збору обрано ендпоінт GET /publicMatches, що повертає до 100 матчів за один запит і вже містить усі потрібні поля: результат, тривалість, ранг та склади команд. Альтернативний GET /matches/{match_id} повертає лише один матч і вимагав би окремого запиту на кожен. Тому, навіть попри те що сувора фільтрація відсіює більшість отриманих матчів, пакетний ендпоінт значно ефективніший для масового збору. Для взаємодії з API використовується бібліотека requests [25].

2.3 Формування вектора ознак

Формування інформативного вектора ознак є ключовим кроком у побудові ML-системи. Якість ознак визначає верхню межу точності будь-якого алгоритму [26] - навіть найсучасніша модель не може компенсувати відсутність інформації у вхідних даних. У даній задачі вхідними даними є лише 10 ідентифікаторів героїв, тому задача полягає у максимально ефективному перетворенні цієї обмеженої інформації на ознаки, що несуть статистичний сигнал про результат матчу. Вектор ознак складається з двох принципово різних компонент.

Категоріальні ознаки - one-hot кодування. Перша компонента представляє склад команд у вигляді бінарного one-hot вектора. Для кожного з 200 можливих ідентифікаторів героїв виділяється по дві бінарні ознаки: одна для присутності героя у команді Radiant, інша - для Dire. Позиції $0 \dots 199$ відповідають героям Radiant: vec[hero_id] $= 1$, якщо герой обраний Radiant, і 0 інакше. Позиції $200 \dots 399$ аналогічно кодують склад Dire. Загальна розмірність - 400 бінарних ознак. Кожен матч має рівно 10 одиниць у векторі - по 5 для кожної команди.

Розбиття на дві окремі частини (а не єдиний вектор зі значеннями $+1/-1$) обрано для сумісності з деревовидними методами (RF, XGBoost), які при побудові вузла порівнюють значення ознаки з порогом. Для бінарних ознак $\{0,1\}$ поріг 0,5 відповідає умові «герой присутній / відсутній». При кодуванні $\{+1, -1\}$ один і той самий герой кодувався б значеннями з різним знаком залежно від команди, що ускладнило б інтерпретацію умов розбиття.

Числові ознаки - статистичні метрики. Друга компонента містить 13 числових ознак, що агрегують статистику взаємодії героїв. Усі метрики обчислюються виключно на тренувальній підвибірці. Це принципова методологічна вимога: якби статистику рахували по всьому датасету, тестові матчі опосередковано впливали б на параметри моделі - явище, відоме як витік

даних (data leakage). Витік даних призводить до завищених оцінок якості моделі під час тестування, але до гіршої реальної ефективності при застосуванні на нових даних.

Winrate та pickrate героїв визначаються за формулами (2.9) та (2.10):

$$WR(h) = \frac{\text{wins}(h)}{\text{games}(h)}, \tag{2.9}$$

$$PR(h) = \frac{\text{games}(h)}{\text{total\_matches}}, \tag{2.10}$$

де $\text{wins}(h)$ - кількість матчів, у яких команда з героєм h перемогла; $\text{games}(h)$ - загальна кількість ігор героя h у тренувальній вибірці; total_matches - розмір тренувальної вибірки. Герої з менш ніж 5 іграми отримують нейтральне значення $WR(h) = 0{,}5$, щоб уникнути статистично незначущих оцінок.

Для кожної команди T обчислюються агреговані показники: $\overline{WR}(T) = \frac{1}{5}\sum WR(h_i)$ - середній winrate як узагальнений показник сили команди; $WR_{\max}(T) = \max WR(h_i)$ - максимальний winrate, що виявляє наявність одного дуже сильного героя; $WR_{\min}(T) = \min WR(h_i)$ - мінімальний winrate, що виявляє слабку ланку команди; $\overline{PR}(T) = \frac{1}{5}\sum PR(h_i)$ - середній pickrate, що відображає популярність складу. Різниці між командами $\overline{WR}_R - \overline{WR}_D$ і $\overline{PR}_R - \overline{PR}_D$ є найбільш прямими сигналами для прогнозу.

Баєсівське згладжування статистик взаємодії. Синергія пари героїв (h_1, h_2) - це їхній спільний winrate, коли вони грають в одній команді. Контрпік (h_R, h_D) - winrate Radiant, коли герой h_R протистоїть герою h_D з команди Dire. Наївне обчислення wins/games призводить до статистичних артефактів: пара, що зустрілася мало разів, може мати winrate 0% або 100%, що є наслідком малої вибірки, а не реальним показником синергії чи контрпіку.

Для згладжування застосовується баєсівська оцінка з апріорним розподілом [27] за формулою (2.11):

$$WR_{\text{smooth}}(\text{pair}) = \frac{\text{wins(pair)}+\text{prior}}{\text{games(pair)}+10}, \quad (2.11)$$

Апріорний внесок розрізняється для двох типів взаємодії, що відображає їхню природу. Для синергій апріор центрується на глобальному winrate, $\text{prior} = WR_{\text{global}} \cdot 10$, де WR_{global} - частка перемог Radiant у тренувальній вибірці ($\approx 0{,}534$), оскільки синергія оцінюється з боку команди. Для контрпіків апріор центрується на нейтральному значенні 0,5 ($\text{prior} = 5$), бо winrate у протистоянні «герой проти героя» симетричний відносно 0,5. Константа 10 у знаменнику визначає силу апріорного розподілу: при $\text{games(pair)} = 0$ оцінка дорівнює апріорному значенню; при $\text{games(pair)} = 10$ вага апріору й реальних даних однакова; при $\text{games(pair)} \gg 10$ оцінка наближається до реального winrate пари. Такий підхід є стандартним прийомом у статистичному аналізі рідкісних подій.

Синергія команди обчислюється як середнє по всіх $C(5,2) = 10$ унікальних парах за формулою (2.12):

$$SYN(T) = \frac{1}{10}\sum_{i<j} WR_{\text{smooth}}(h_i, h_j), \quad i,j = 1 \dots 5, \quad (2.12)$$

Контрпік Radiant проти Dire обчислюється по всіх $5 \times 5 = 25$ парах за формулою (2.13):

$$CTR = \frac{1}{25}\sum_{h_R \in R,\ h_D \in D} WR_{\text{smooth}}(h_R, h_D), \quad (2.13)$$

Аналіз важливості ознак Random Forest у розділі 4 підтвердить, що CTR (ознака Radiant_Counter_Dire) є найінформативнішою у всьому векторі. Разом CTR і дві ознаки SYN забезпечують понад 43% сукупної важливості моделі при тому, що займають лише 3 позиції з 413 у векторі ознак. Це підтверджує правильність рішення включити агреговані метрики взаємодій до вектора ознак.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі обґрунтовано вибір методів машинного навчання та технологічного стеку для реалізації системи прогнозування результатів матчів Dota 2.

Розглянуто п’ять підходів до прогнозування результату матчу. Для кожного методу детально описано математичну основу та алгоритм роботи: сигмоїдну трансформацію і L2-регуляризацію для логістичної регресії, механізм bagging та випадкового підпростору ознак для Random Forest, послідовний бустинг з регуляризованим функціоналом для XGBoost, ієрархічне стиснення з активацією ReLU і ранньою зупинкою для MLP. Для кожного методу обґрунтовано вибір гіперпараметрів з урахуванням специфіки вхідних даних - переважно бінарного простору ознак та дисбалансу класів. Проведено порівняльний аналіз теоретичних характеристик методів. Практичні результати порівняння наведено в розділі 4.

Обґрунтовано вибір технологічного стеку: Python як основна мова розробки з бібліотеками scikit-learn та XGBoost для ML-компонента, FastAPI для серверної частини, SQLite для зберігання даних, OpenDota API як джерело ігрової статистики.

Описано методологію формування вектора ознак розмірністю 413: 400 бінарних ознак one-hot кодування складу команд та 13 числових агрегованих метрик (winrate, pickrate, синергії та контрпіки). Для синергій і контрпіків застосовано баєсівське згладжування за формулою (2.11), що усуває статистичні артефакти при малій кількості спільних матчів пари. Принцип обчислення метрик виключно на тренувальних даних запобігає витоку інформації на тестову вибірку.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Архітектура системи

3.1.1 Загальна структура клієнт-серверної системи

При проектуванні системи ключовим принципом було чітке розділення відповідальності між компонентами та їхня максимальна незалежність. Кожен модуль виконує строго визначену функцію і взаємодіє з іншими лише через чітко визначені інтерфейси - файли на диску та HTTP-запити. Такий підхід забезпечує можливість незалежного тестування, модифікації й заміни окремих компонентів без впливу на решту системи.

Систему реалізовано за трирівневою клієнт-серверною архітектурою. Рівень збору даних включає два автономних скрипти: collector.py для безперервного збору матчів через OpenDota API та heroes.py для одноразового завантаження довідника героїв. Обидва скрипти є повністю незалежними від веб-сервера й запускаються окремо. Рівень бізнес-логіки складається з модуля stats.py, що обчислює статистичні метрики за запитом, та модуля ml.py, який реалізує повний пайплайн машинного навчання. Рівень представлення реалізований через FastAPI-сервер (main.py), що обробляє HTTP-запити від браузера й повертає готові HTML-сторінки через Jinja2-шаблони.

Принципово важливим архітектурним рішенням є розділення офлайн- і онлайн-частин системи. Скрипти collector.py і ml.py запускаються вручну та зберігають результати на диск - перший наповнює базу даних матчами, другий навчає моделі й зберігає їх у директорії models/. Веб-сервер у процесі роботи не збирає дані й не навчає моделі - він лише читає збережені результати та обчислює статистику. Завдяки цьому час відповіді сервера не залежить від

часу навчання моделей, і сервер може бути запущений одразу після підготовки даних.

3.1.2 Опис модулів та їх взаємодія

Взаємодія між модулями організована виключно через файлову систему: collector.py записує матчі в dota2.db, ml.py читає з dota2.db і записує .pkl файли у models/, main.py читає з dota2.db і з models/. Між модулями немає прямих викликів функцій, і вони не залежать від внутрішньої реалізації одне одного. Така архітектура дозволяє замінювати або вдосконалювати будь-який модуль незалежно від решти. Наприклад, модуль ml.py можна повністю переписати, змінивши алгоритми навчання, - і це ніяк не вплине на collector.py чи main.py, доки зберігається формат .pkl файлів. Спільні налаштування (шлях до бази, параметри збору, адреса API) винесено в окремий модуль config.py, що читає їх із файлу .env, тож зміна конфігурації не потребує редагування коду.

Структуру файлів проєкту наведено в таблиці 3.1.

Таблиця 3.1 - Структура файлів проєкту

Файл / Директорія	Призначення	Режим роботи
collector.py	Збір матчів через OpenDota API	Офлайн
heroes.py	Завантаження довідника героїв	Офлайн (одноразово)
config.py / .env	Конфігурація системи	Спільний
stats.py	Розрахунок статистичних метрик	Онлайн (на запит)
ml.py	Навчання ML-моделей та генерація прогнозів	Офлайн / Онлайн
main.py	FastAPI-сервер, визначення роутерів	Онлайн
dota2.db	SQLite база даних з матчами і героями	Спільний ресурс
models/	Збережені .pkl файли моделей і статистик	Спільний ресурс
templates/	Jinja2 HTML-шаблони сторінок	Онлайн

3.2 Проектування бази даних

База даних є центральним сховищем системи, через яке взаємодіють модулі збору й бізнес-логіки. Вибір SQLite обумовлено відповідністю масштабу задачі: для 150 000 рядків і переважно читаючих навантажень SQLite забезпечує достатню продуктивність без необхідності налаштування окремого серверного процесу. Уся база зберігається в єдиному файлі dota2.db, що спрощує резервне копіювання й перенесення проєкту.

База даних містить дві таблиці з різним призначенням: matches для зберігання даних матчів, що використовуються у навчанні ML-моделей і розрахунку статистики, та heroes для довідкової інформації, яка використовується лише у веб-інтерфейсі.

Таблиця matches є центральним елементом бази даних. Її схему наведено в таблиці 3.2.

Таблиця 3.2 - Схема таблиці matches

Поле	Тип	Призначення
id	INTEGER PRIMARY KEY	Ідентифікатор матчу з OpenDota (match_id)
radiant_win	INTEGER NOT NULL	Результат: 1 - перемога Radiant, 0 - Dire
duration	INTEGER NOT NULL	Тривалість матчу в секундах
avg_rank_tier	INTEGER	Середній ранг гравців (60 = Ancient, 70 = Divine)
num_rank_tier	INTEGER	Кількість гравців з визначеним рангом
game_mode	INTEGER	Ігровий режим (22 = Ranked All Pick)
start_time	INTEGER	Unix-timestamp початку матчу
r1, r2, r3, r4, r5	INTEGER	Ідентифікатори п’яти героїв Radiant
d1, d2, d3, d4, d5	INTEGER	Ідентифікатори п’яти героїв Dire

Зберігання піків героїв у вигляді окремих колонок r1-r5 та d1-d5 є свідомим проектним рішенням замість альтернатив (JSON-рядок або окрема таблиця picks). При завантаженні рядків для навчання моделей Python отримує ідентифікатори героїв безпосередньо як елементи кортежу без потреби в додатковому парсингу JSON. При виконанні аналітичних запитів (наприклад, пошуку всіх матчів, де грав герой з певним id) можна використовувати стандартні умови SQL: WHERE r1 = 74 OR r2 = 74 OR ..., що є ефективнішим, ніж пошук у JSON-рядку. Первинний ключ таблиці відповідає match_id з OpenDota, що гарантує унікальність записів: оператор INSERT OR IGNORE при збереженні нових матчів автоматично ігнорує дублікати.

Таблиця heroes містить довідкову інформацію, що завантажується одноразово через скрипт heroes.py. Її схему наведено в таблиці 3.3.

Таблиця 3.3 - Схема таблиці heroes

Поле	Тип	Призначення
id	INTEGER PRIMARY KEY	Ідентифікатор героя в системі Dota 2
name	TEXT NOT NULL	Технічна назва (npc_dota_hero_axe)
localized_name	TEXT NOT NULL	Відображувана назва (Axe)
primary_attr	TEXT	Основний атрибут: str / agi / int / all
attack_type	TEXT	Тип атаки: Melee / Ranged
roles	TEXT	Ролі через кому (Carry, Support, Nuker)
img	TEXT	URL зображення героя на сервері OpenDota

Ролі героя зберігаються як рядок через кому (наприклад, Carry,Initiator,Durable) замість окремої таблиці зв’язку many-to-many. Це свідоме спрощення схеми: ролі є статичними довідковими даними, що не беруть участі в ML-обчисленнях і потрібні лише для фільтрації у веб-інтерфейсі. Розбиття рядка через split(',') виконується в Python при підготовці

даних для Jinja2-шаблону. Нормалізація схеми до третьої нормальної форми в цьому випадку ускладнила б код без будь-яких практичних переваг.

3.3 Модуль збору даних

Модуль collector.py реалізує автоматизований збір рейтингових матчів Dota 2 через публічний API OpenDota. Якість і репрезентативність зібраного датасету безпосередньо впливають на якість навчання ML-моделей, тому модуль реалізує суворі критерії фільтрації.

Для збору обрано ендпоінт GET /publicMatches OpenDota API, що повертає до 100 матчів за один HTTP-запит. Ключова технічна перевага порівняно з детальним ендпоінтом GET /matches/{match_id} полягає в тому, що /publicMatches повертає всю потрібну інформацію (склад команд, результат, тривалість, ранг) одразу для 100 матчів. Через суворі критерії якості значна частина отриманих матчів відсіюється, тому для накопичення цільового обсягу потрібно багато запитів; проте кожен запит дає до 100 матчів-кандидатів, тоді як /matches/{match_id} вимагав би окремого запиту для кожного матчу без отримання додаткових потрібних даних. Це робить пакетний ендпоінт значно ефективнішим для масового збору.

Пагінація реалізована через параметр less_than_match_id: кожний наступний запит отримує матчі з ідентифікатором, меншим за мінімальний у попередньому батчі. Параметр mmr_descending=1 пріоритизує матчі з вищим рейтингом гравців. При повторному запуску скрипт завантажує множину вже збережених ідентифікаторів через SELECT id FROM matches і пропускає їх при обробці - це забезпечує ідемпотентність збору й можливість відновлення після переривання.

Фільтрація якісних матчів реалізована у функції is_quality(), яка перевіряє кожен матч за шістьма критеріями. Перший - ігровий режим Ranked All Pick (game_mode = 22). Виключаються турбо-матчі (mode 23), де тривалість матчу вдвічі коротша, а баланс героїв суттєво відрізняється від стандартного

рейтингового формату, а також усі аркадні режими з нестандартними правилами. Другий - середній ранг гравців Ancient або вище (avg_rank_tier ≥ 60). Ранг кодується двозначним числом: перша цифра - bracket (6 = Ancient, 7 = Divine, 8 = Immortal), друга - кількість зірок у межах bracket. Гравці Ancient і вище мають достатній рівень розуміння гри для усвідомленого вибору героїв, що робить їхні піки репрезентативнішими для аналізу синергій і контрпіків. Третій критерій - не менше п’яти гравців з визначеним рангом (num_rank_tier ≥ 5): це виключає матчі, де більшість учасників є новими або анонімними акаунтами. Четвертий - тривалість від 20 до 90 хвилин: виключаються покинуті матчі та аномально довгі. П’ятий - наявність визначеного результату (radiant_win не є None). Шостий - повний і коректний склад обох команд: рівно 10 ненульових ідентифікаторів героїв без повторень.

Для підвищення надійності збору при тривалій роботі реалізовано механізм відновлення пагінації. Лічильник empty_streak відстежує кількість послідовних батчів, у яких не було збережено жодного матчу, - це відбувається, коли скрипт досягає зони архіву з матчами, що не проходять фільтрацію. При досягненні значення 5 пагінація скидається до найновіших матчів, і збір відновлюється. Це запобігає зависанню скрипту й гарантує безперервну роботу.

3.4 Модуль статистики

Модуль stats.py реалізує обчислення аналітичних метрик, що відображаються у веб-інтерфейсі. На відміну від статистик у ML-модулі, які обчислюються один раз при навчанні, stats.py виконує обчислення за зверненням до відповідних сторінок. Для уникнення надлишкових обчислень результати кешуються в оперативній пам’яті й перераховуються лише при зміні кількості матчів у базі (перевірка виконується через швидкий запит COUNT). Таке рішення поєднує дві вимоги: статистика залишається

актуальною й оновлюється без перезапуску сервера при поповненні бази, але водночас не перераховується марно на кожен запит.

3.4.1 Розрахунок winrate та pickrate

Функція get_hero_stats() завантажує всі матчі з таблиці matches в оперативну пам’ять і обчислює для кожного героя кількість ігор і перемог. Ключова особливість розрахунку - врахування сторони команди: якщо герой грав за Radiant і Radiant переміг, це зараховується як перемога героя; якщо за Dire і Dire перемогла - теж перемога. Такий підхід нівелює природний дисбаланс 53,4% / 46,6% між командами й дозволяє порівнювати ефективність героїв незалежно від їхньої переважної сторони карти.

Функція get_top_heroes() повертає N героїв з найвищим winrate або pickrate з мінімальним порогом кількості ігор (min_picks). Поріг необхідний для виключення статистично незначущих результатів: герой з 3 іграми і 3 перемогами має winrate 100%, що є наслідком малої вибірки, а не реальним показником ефективності. Функції get_duration_buckets() та get_winrate_by_rank() обчислюють розподіл тривалості матчів і winrate Radiant за рангами для відображення у вигляді графіків. Загальний winrate Radiant і середня тривалість матчу обчислюються окремими функціями get_radiant_winrate() та get_avg_duration() безпосередньо засобами SQL.

3.4.2 Обчислення синергій та контрпіків

Функція get_hero_pairs() знаходить пари героїв з найвищим спільним winrate. Для кожного матчу перебираються всі $C(5,2) = 10$ унікальних пар героїв кожної команди. Пара ідентифікується відсортованим кортежем (min_id, max_id), що виключає дублювання - пара (74, 14) і (14, 74) є однією й тією самою. Функція повертає топ синергій з мінімальним порогом спільних

ігор. Без цього порогу результати були б спотворені рідкісними парами, що зустрілися 1-2 рази й отримали екстремальний winrate.

Функція get_counters() обчислює для конкретного героя список персонажів, що найефективніше йому протидіють. Для кожного матчу, де обраний герой присутній, фіксуються герої протилежної команди й результат матчу - чи перемогла команда суперника. Накопичена статистика дозволяє ранжувати контерів за ефективністю. Такий підхід враховує не лише часті протистояння, а й реальну ефективність у цих протистояннях.

3.5 Модуль машинного навчання

Модуль ml.py реалізує повний пайплайн машинного навчання - від завантаження даних до збереження навчених моделей на диск. Це найвідповідальніший компонент системи, оскільки його коректність напряму визначає якість прогнозів. Методологічний порядок операцій у пайплайні є критичним: будь-яке порушення призводить до некоректної оцінки якості моделей. Пайплайн складається із семи послідовних кроків, між якими існують строгі залежності.

Крок 1 - завантаження даних. Функція load_rows() зчитує всі рядки з таблиці matches у вигляді списку кортежів. Кожен кортеж містить мітку результату та 10 ідентифікаторів героїв.

Крок 2 - стратифікована розбивка вибірки. Через train_test_split з параметром stratify вибірка ділиться на тренувальну (80%, 120 000 матчів) і тестову (20%, 30 000 матчів). Параметр stratify гарантує, що пропорція класів 53,4% / 46,6% зберігається в обох підвибірках. Без стратифікації при випадковій розбивці пропорції можуть відрізнятися, що ускладнює коректне порівняння моделей.

Крок 3 - обчислення статистик. Функції compute_hero_winrates(), compute_pickrates() і compute_interactions() обчислюють winrate, pickrate, синергії та контрпіки виключно на тренувальній підвибірці. Цей крок є

методологічно критичним і обов'язково виконується після розбивки. Якби статистику обчислювали до розбивки або по всьому датасету, виникав би витік даних: тестові матчі впливали б на статистику, яка потім використовується для побудови ознак тих самих тестових матчів. Це призвело б до штучно завищених оцінок якості.

Крок 4 - побудова векторів ознак. Функція rows_to_features() перетворює рядки БД у пари матриць: X_num (числові ознаки, 13 стовпців) та X_cat (категоріальні one-hot ознаки, 400 стовпців). Розділення на дві матриці необхідне для наступного кроку.

Крок 5 - нормалізація числових ознак. StandardScaler навчається на X_num з тренувальної підвибірки (метод fit) і застосовується до числових матриць обох підвибірок (метод transform). До категоріальних бінарних ознак нормалізація не застосовується - значення {0,1} вже перебувають в однаковому масштабі.

Крок 6 - об'єднання матриць. Через numpy.hstack числові й категоріальні матриці об'єднуються горизонтально, утворюючи фінальний вектор ознак розмірністю 413 для кожної підвибірки.

Крок 7 - навчання та оцінювання моделей. Кожна з чотирьох ML-моделей навчається на X_train_final і оцінюється на X_test_final. Для кожної моделі обчислюються Accuracy, ROC-AUC та per-class метрики (precision, recall, F1 для класів Radiant і Dire), а для найкращої моделі будується матриця помилок (аналізується в розділі 4). Навчені моделі разом з об'єктом scaler і словниками статистик зберігаються у директорії models/ через pickle. При зверненні до предиктора FastAPI завантажує ці файли й будує вектор ознак за ідентичною логікою - це гарантує, що прогнози генеруються з тими самими параметрами нормалізації, що використовувалися при навчанні.

3.6 Веб-інтерфейс

3.6.1 Структура роутерів FastAPI

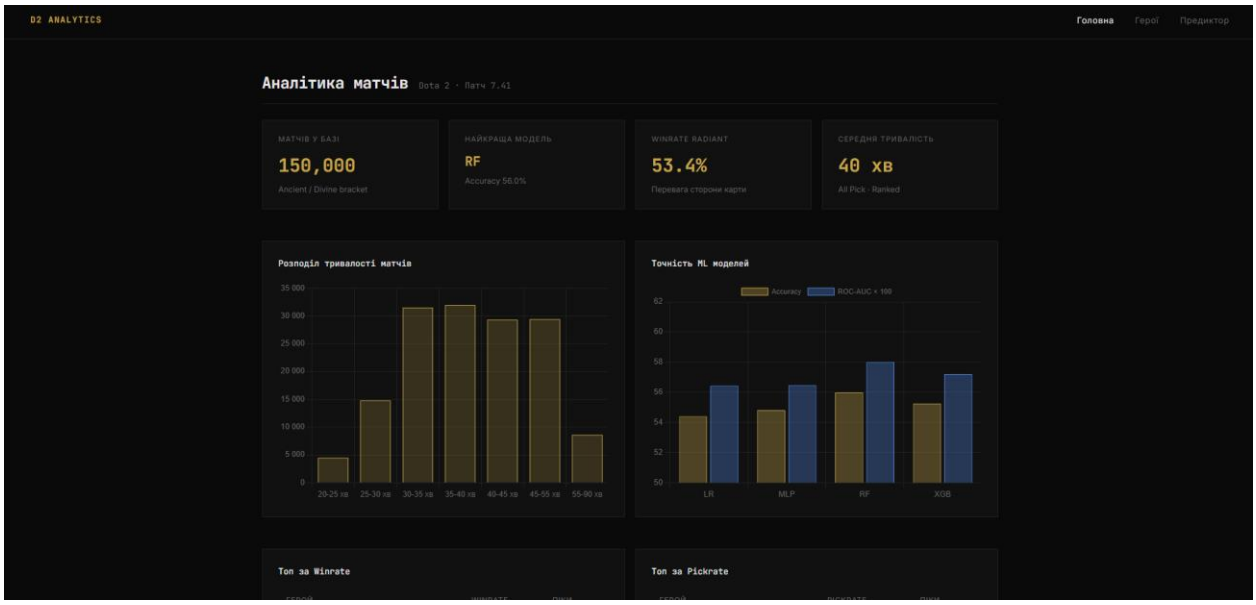
Веб-інтерфейс реалізовано на базі FastAPI із серверним рендерингом HTML через Jinja2-шаблони. Підхід серверного рендерингу обрано замість SPA (Single Page Application) фреймворків, оскільки він не потребує окремого фронтенд-сервера, складних налаштувань збірки чи окремого API для передачі даних. Усі дані обчислюються на сервері й передаються у шаблон як Python-об’єкти, що є найпростішим і найнадійнішим рішенням для проєкту такого масштабу. Для графіків використовується бібліотека Chart.js [28], підключена через CDN.

Сервер реалізує п’ять роутерів. GET / обробляє головну сторінку: завантажує топ героїв за winrate і pickrate через stats.py, розподіл тривалості матчів, результати ML-моделей з results.pkl та рендерить index.html. GET /heroes повертає сторінку статистики всіх 127 героїв з підтримкою фільтрації через GET-параметри attr і role. GET /predict відображає форму вибору складу команд. POST /predict обробляє відправлену форму, перевіряє унікальність вибору (кожен герой може бути обраний лише один раз), викликає ml.predict() і повертає сторінку з результатами прогнозування. GET /predict/by-match приймає параметр match_id, перевіряє його коректність, звертається до OpenDota API для отримання складу завершеного матчу й автоматично заповнює форму.

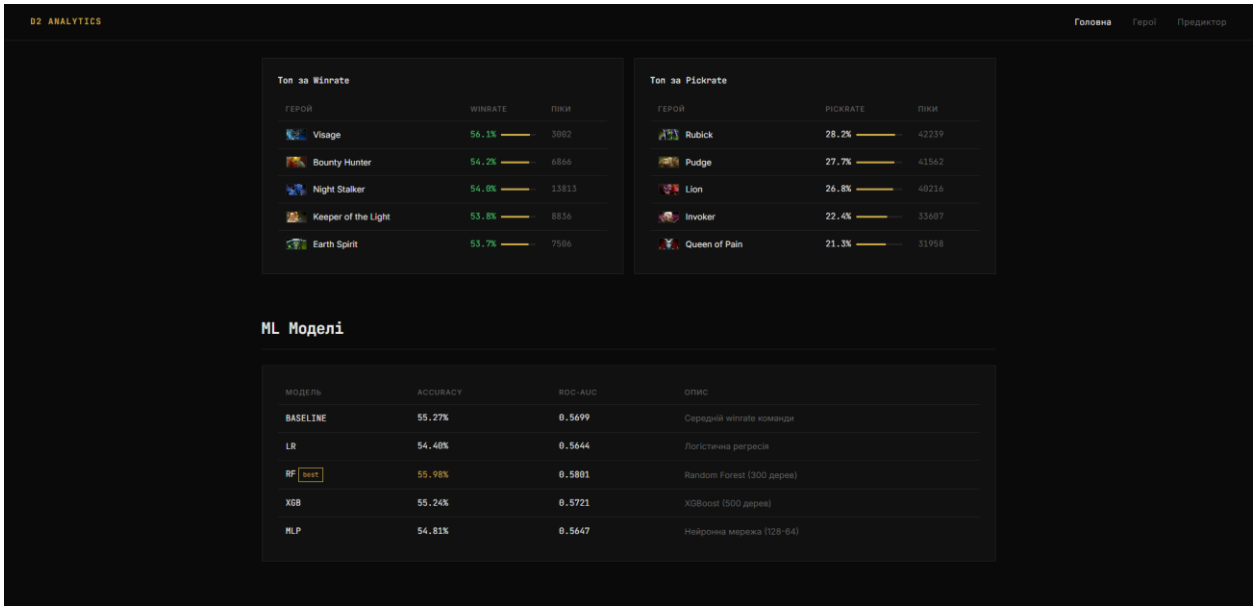
3.6.2 Опис сторінок системи

Головна сторінка системи надає загальний огляд зібраної аналітики й результатів моделей. Верхня частина містить чотири інформаційні картки: загальна кількість матчів у базі, назва та accuracy найкращої моделі, загальний

winrate команди Radiant і середня тривалість матчу. Нижче розміщено два інтерактивні графіки, побудовані через Chart.js: стовпчаста діаграма розподілу тривалості матчів за інтервалами (по 5 хвилин, з укрупненням для тривалих матчів) і діаграма порівняння accuracy та ROC-AUC всіх моделей. У нижній частині розташовані таблиці топ-5 героїв за winrate та pickrate із зображеннями персонажів. Скріншот головної сторінки наведено на рисунку 3.1.



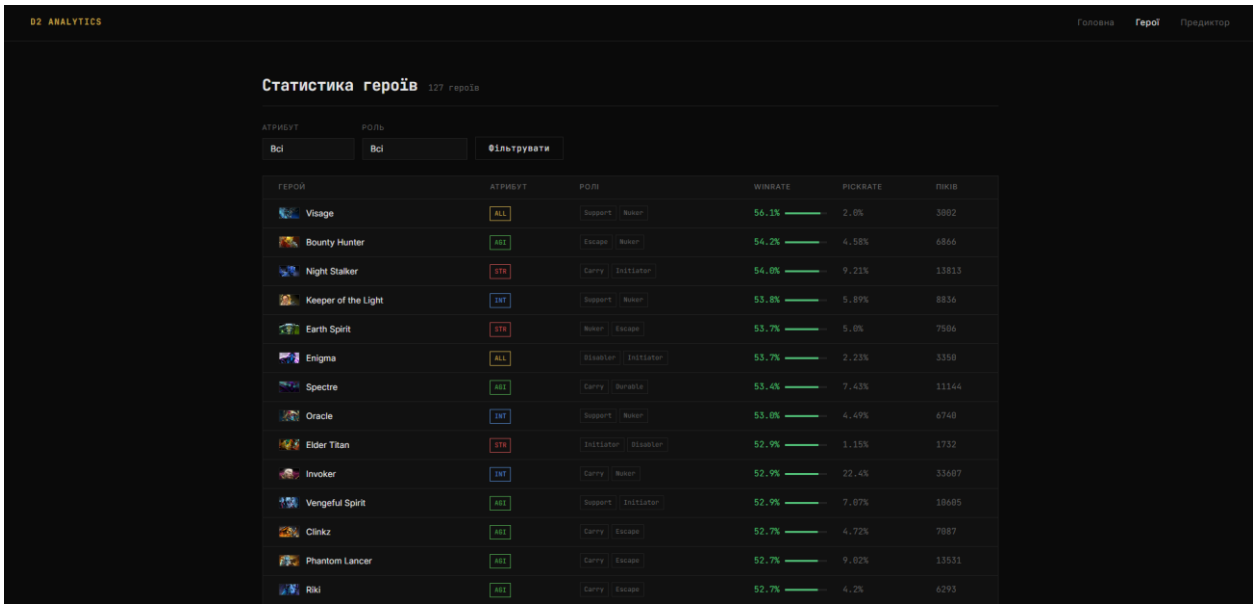
а)



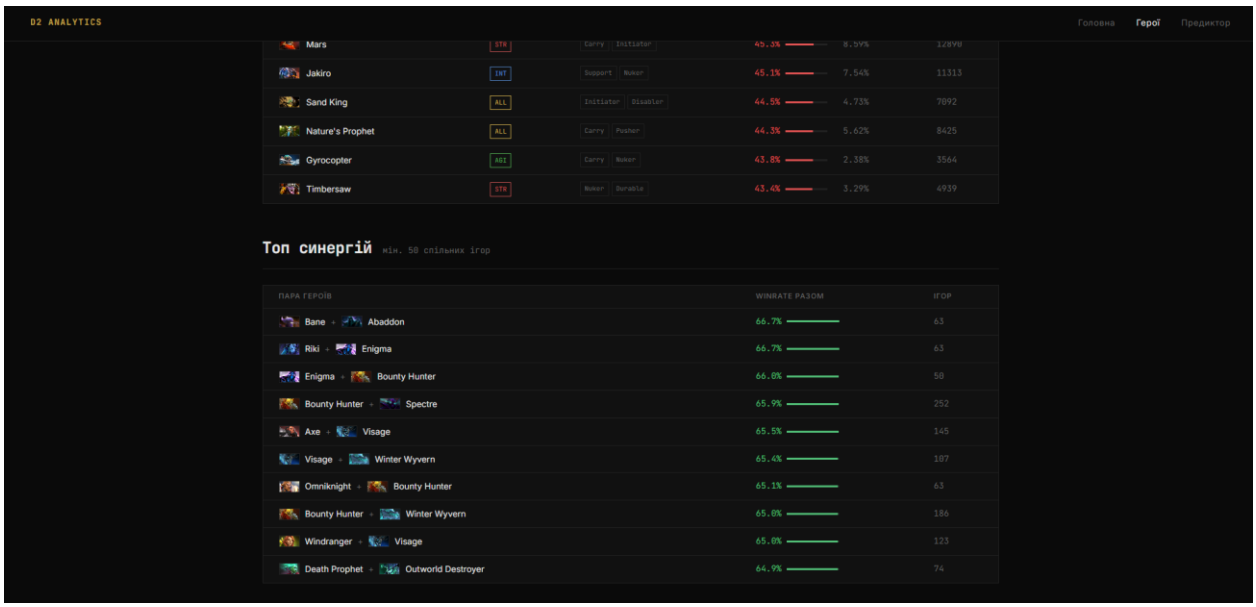
б)

Рисунок 3.1 - Головна сторінка веб-інтерфейсу системи: а - верхня частина; б - нижня частина

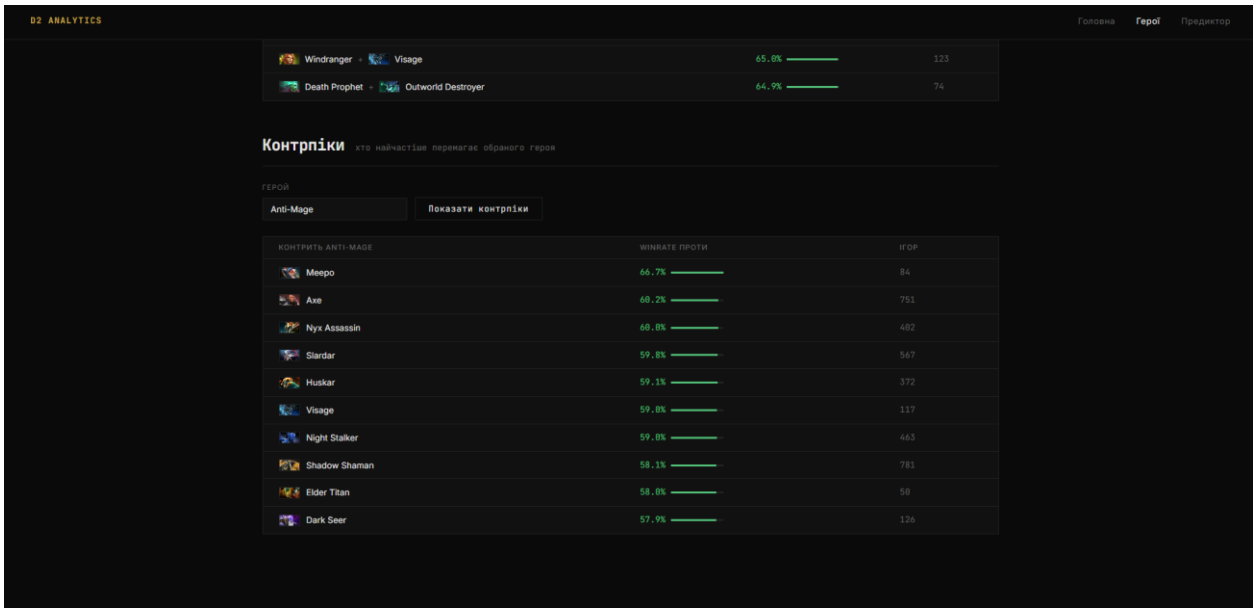
Сторінка статистики героїв відображає повну таблицю всіх 127 персонажів гри з можливістю фільтрації за основним атрибутом (Strength, Agility, Intelligence, Universal) та ігровою роллю (Carry, Support, Nuker, Disabler тощо). Таблиця містить колонки: зображення героя, назва, атрибут, ролі, winrate з кольоровим індикатором (зелений при winrate вище 52%, червоний - нижче 48%) та кількість матчів. Кольоровий індикатор winrate дозволяє швидко ідентифікувати сильних і слабких героїв поточного патчу без читання конкретних значень. Нижче основної таблиці розташована таблиця топ-10 синергій пар героїв з мінімальним порогом 50 спільних матчів. Під нею розміщено блок контрпіків: користувач обирає героя зі списку, і система виводить персонажів, які найчастіше його перемагають, із зазначенням winrate проти обраного героя та кількості спільних ігор. Скріншот сторінки героїв наведено на рисунку 3.2.



а)



б)

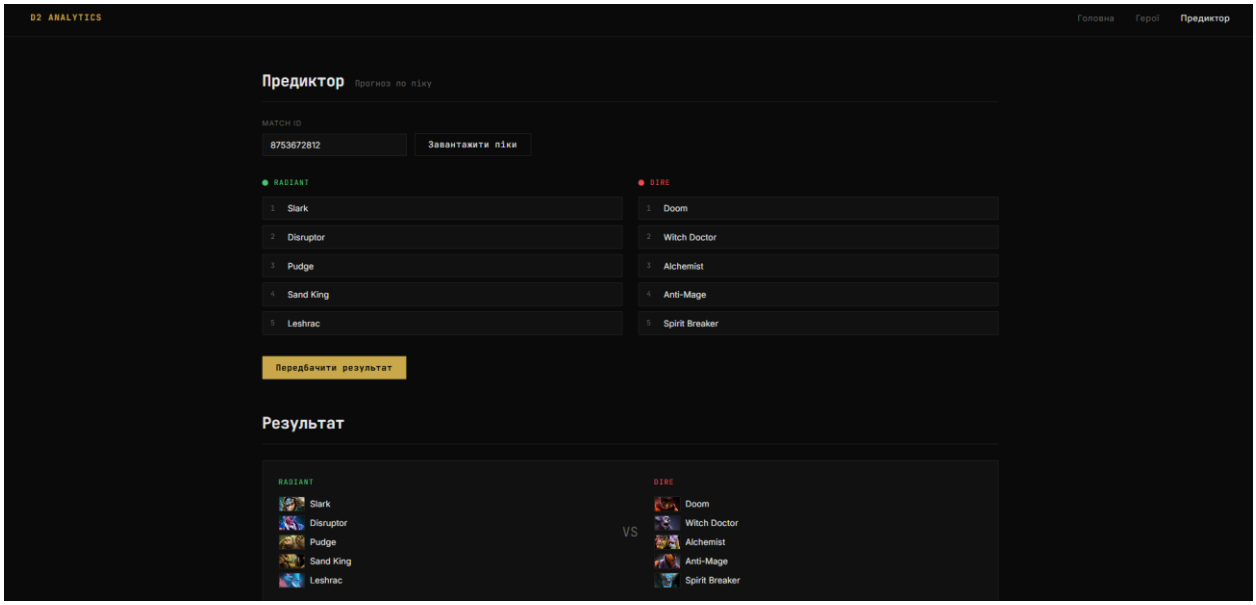


в)

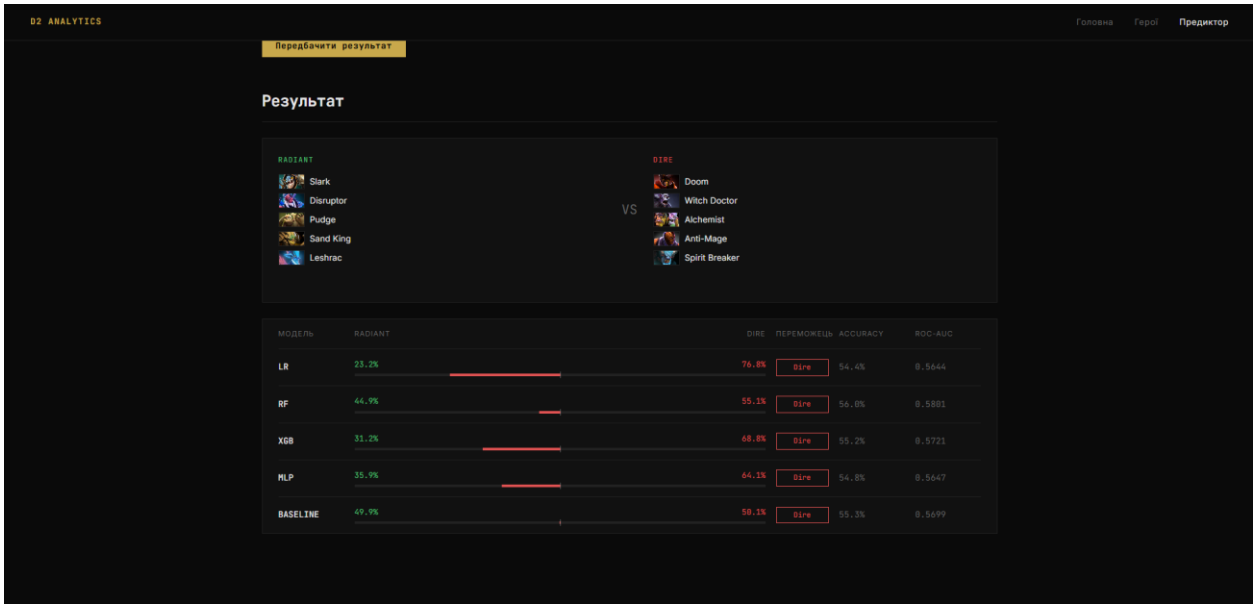
Рисунок 3.2 - Сторінка статистики героїв: а - таблиця героїв;
б - закінчення таблиці героїв і блок синергій; в - блок контрпіків.

Сторінка предиктора є основним функціональним елементом системи. Вона надає два способи введення складу команд. Перший - ручний вибір через десять випадаючих списків (по п’ять для кожної команди), де кожен містить усіх 127 героїв в алфавітному порядку. Другий - автоматичне заповнення за ідентифікатором матчу: користувач вводить числовий match_id будь-якого завершеного матчу, і система звертається до OpenDota API, щоб отримати склад команд і автоматично заповнити форму. Після відправки форми на тій

самій сторінці відображається блок результатів: зображення обраних героїв обох команд і порівняльна таблиця прогнозів від усіх п’яти моделей з відсотком ймовірності перемоги Radiant у вигляді прогресбару й бінарним прогнозом переможця. Скріншот предиктора наведено на рисунку 3.3.



а)



б)

Рисунок 3.3 - Сторінка предиктора з результатами прогнозування: а - верхня частина; б - нижня частина

3.6.3 Інструкція користувача

Розгортання системи виконується у п’ять послідовних кроків. По-перше, встановити залежності командою pip install -r requirements.txt та, за потреби, налаштувати параметри системи у .env (шлях до бази, цільова кількість матчів, критерії фільтрації). По-друге, запустити збір даних командою python collector.py і дочекатися накопичення щонайменше 50 000 матчів - рекомендований обсяг 150 000. По-третє, завантажити довідник героїв командою python heroes.py. По-четверте, навчити ML-моделі командою python ml.py - процес займає кілька хвилин залежно від обсягу датасету й потужності процесора. По-п’яте, запустити веб-сервер командою uvicorn main:app --reload і відкрити браузер за адресою http://localhost:8000.

Для перегляду статистики героїв необхідно перейти до розділу «Герої» через навігаційне меню. Фільтрація виконується через випадаючі списки «Атрибут» і «Роль» над таблицею, після чого слід натиснути «Фільтрувати». Кнопка «Скинути» повертає повний список. Для перегляду контрпіків конкретного героя його обирають у списку внизу сторінки й натискають «Показати контрпіки».

Для отримання прогнозу необхідно перейти до розділу «Предиктор». Перший спосіб - увести ідентифікатор завершеного матчу в поле Match ID і натиснути «Завантажити піки»: форма автоматично заповниться складом обох команд. Другий спосіб - вручну обрати по п’ять героїв для кожної команди. Після натискання «Передбачити результат» відображається порівняльна таблиця прогнозів від усіх моделей. Якщо один герой обраний для обох команд, система відображає повідомлення про помилку й не виконує прогноз.

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі описано проектування та реалізацію всіх компонентів клієнт-серверної системи аналізу та прогнозування матчів Dota 2.

Обрана трирівнева архітектура з повним розділенням офлайн-підготовки даних і онлайн-обслуговування запитів забезпечує швидкий відгук сервера незалежно від обсягу датасету. Взаємодія між модулями виключно через файлову систему гарантує їхню незалежність і можливість модифікації без впливу на решту системи.

Спроектовано базу даних із двома таблицями. Зберігання піків героїв у вигляді окремих колонок r1-r5 та d1-d5 забезпечує зручність SQL-запитів і ефективність при побудові ML-векторів.

Реалізовано модуль збору з фільтрацією за шістьма критеріями якості й механізмом автоматичного відновлення пагінації. Модуль статистики обчислює метрики за запитом з кешуванням, що забезпечує актуальність даних без перезапуску сервера й водночас усуває надлишкові обчислення. Модуль машинного навчання реалізує коректний семикроковий пайплайн, де обчислення статистик виконується виключно на тренувальних даних після розбивки вибірки, що запобігає витоку інформації. Веб-інтерфейс надає три функціональні сторінки з підтримкою автоматичного завантаження складу за ідентифікатором матчу.

РОЗДІЛ 4
АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Характеристика зібраного датасету

Збір даних проводився протягом кількох діб за допомогою скрипту collector.py через публічний API OpenDota. У результаті сформовано датасет обсягом 150 000 рейтингових матчів Dota 2 патчу 7.41. Усі матчі пройшли шестиетапну фільтрацію, описану в розділі 3, і відповідають критеріям якості: ігровий режим Ranked All Pick, рівень Ancient і вище, тривалість від 20 до 90 хвилин, повний склад обох команд. Загальну характеристику датасету наведено в таблиці 4.1.

Таблиця 4.1 - Загальна характеристика зібраного датасету

Показник	Значення
Загальна кількість матчів	150 000
Патч гри	7.41
Ігровий режим	Ranked All Pick (game_mode = 22)
Мінімальний ранг гравців	Ancient (avg_rank_tier ≥ 60)
Тривалість матчів	від 20 до 90 хвилин
Перемоги Radiant	80 140 (53,4%)
Перемоги Dire	69 860 (46,6%)
Тренувальна вибірка	120 000 матчів (80%)
Тестова вибірка	30 000 матчів (20%)

Розподіл результатів 53,4% на користь Radiant є добре відомою особливістю Dota 2 і пов’язаний з асиметрією карти. Команда Radiant розташована у нижньому лівому куті карти й має тактичну перевагу: кращий огляд ключових об’єктів (Рошан, аутпости) і зручніше розташування бази відносно центральної лінії. Цей дисбаланс є стабільним протягом багатьох

патчів і враховується при навчанні всіх моделей через параметри class_weight та scale_pos_weight.

Розбивка на тренувальну й тестову вибірки виконана з параметром stratify, що гарантує збереження пропорції 53,4% / 46,6% в обох підвибірках. Це важлива умова коректного порівняння: якби розподіл класів відрізнявся між підвибірками, метрики accuracy і ROC-AUC для різних моделей були б непорівнянними. Розподіл тривалості матчів наведено на рисунку 4.1.

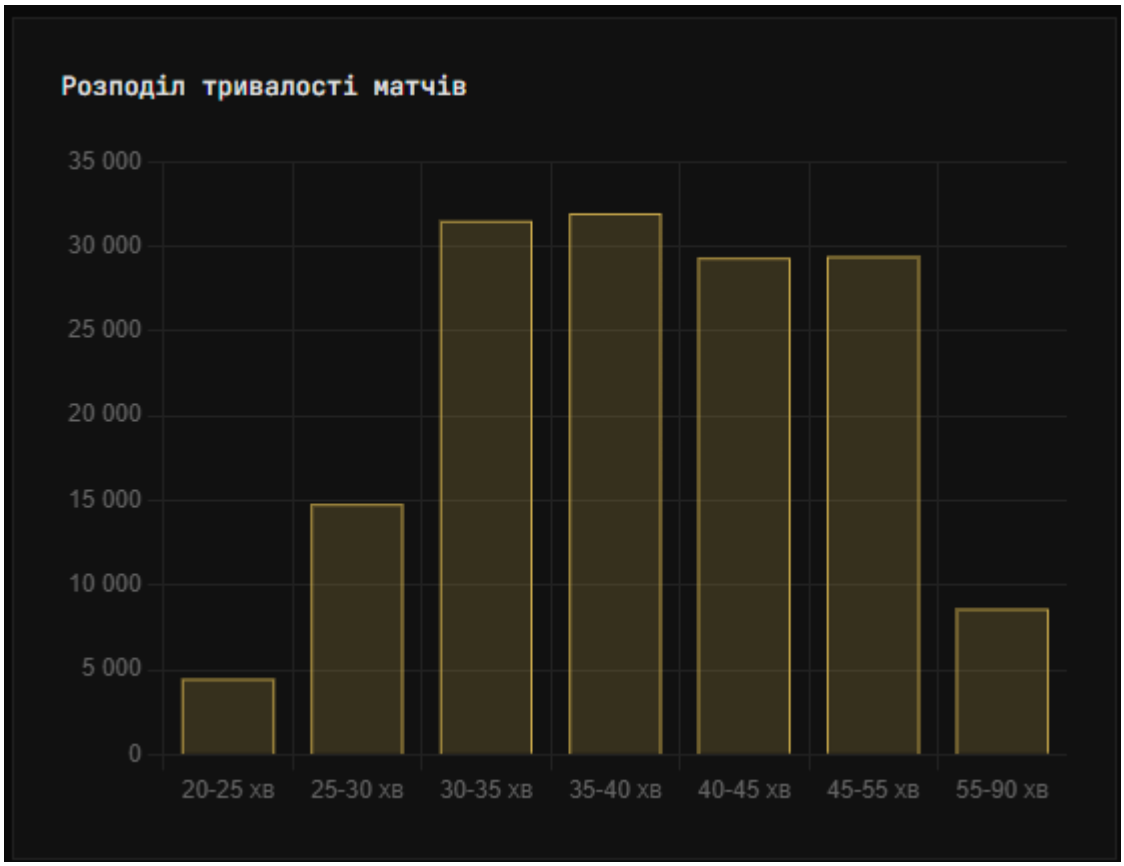


Рисунок 4.1 - Розподіл тривалості матчів датасету

4.2 Порівняння методів машинного навчання

4.2.1 Результати за Accuracy та ROC-AUC

Після навчання всіх моделей на тренувальній вибірці 120 000 матчів та оцінювання на незалежній тестовій вибірці 30 000 матчів отримано результати, наведені в таблиці 4.2. Усі моделі навчалися на однаковому векторі ознак розмірністю 413 і оцінювалися на тій самій тестовій вибірці, що забезпечує коректність порівняння.

Таблиця 4.2 - Порівняння методів машинного навчання на тестовій вибірці (30 000 матчів)

Модель	Accuracy	ROC-AUC
Baseline	55,27%	0,5699
Логістична регресія (LR)	54,40%	0,5644
Random Forest (RF)	55,98%	0,5801
XGBoost (XGB)	55,24%	0,5721
Нейронна мережа (MLP)	54,81%	0,5647

Найкращий результат за обома метриками продемонстрував метод Random Forest: accuracy 55,98% та ROC-AUC 0,5801. Він є єдиним методом, що стабільно перевершує базову модель за обома метриками. Перевага над baseline (+0,71% accuracy і +0,01 ROC-AUC) підтверджує, що ансамбль дерев рішень здатний виявляти нелінійні залежності між складом команди й результатом матчу, яких базова модель не бачить. Зокрема, RF ефективно використовує умовні взаємодії між ознаками: наприклад, синергія конкретної пари героїв може бути корисною лише тоді, коли середній winrate команди перевищує певний поріг, - такі умовні залежності природно виражаються через розгалуження в деревах, але не можуть бути описані лінійною функцією.

XGBoost за accuracy (55,24%) незначно поступається baseline (55,27%), але перевищує його за ROC-AUC (0,5721 проти 0,5699). Це означає, що XGBoost краще калібрує ймовірності, хоча за порогом 0,5 дає дещо гірші бінарні рішення, ніж baseline. Відставання XGBoost від RF, попри традиційно кращі результати цього методу на змаганнях з ML, пояснюється переважно бінарним характером вхідних ознак: 400 із 413 ознак є бінарними значеннями {0,1}. При такому типі даних bagging (RF) є ефективнішим, ніж послідовний бустинг (XGBoost), оскільки бустинг чутливіший до шуму в бінарних ознаках на ранніх ітераціях.

Логістична регресія (54,40%) і MLP (54,81%) поступаються базовій моделі за accuracy, що є важливим аналітичним висновком. Базова модель безпосередньо використовує різницю середніх winrate команд - найпряміший числовий сигнал. Логістична регресія із сильною регуляризацією $C = 0{,}1$ не може повністю засвоїти цей сигнал через присутність 400 слабких бінарних ознак, що ускладнюють оптимізацію. MLP демонструє схожу поведінку - оптимізація нейронної мережі на переважно бінарних даних є складнішим завданням, ніж на неперервних. Обидва методи поступаються baseline і за ROC-AUC (LR: 0,5644 проти 0,5699; MLP: 0,5647 проти 0,5699), що додатково підтверджує: задача є нелінійною й вимагає саме ансамблевих методів, здатних моделювати взаємодії між ознаками. Порівняльний графік метрик усіх моделей наведено на рисунку 4.2.

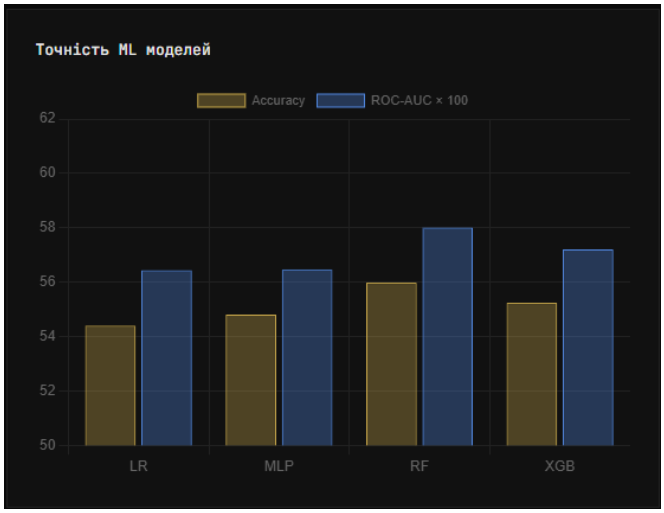


Рисунок 4.2 - Порівняння метрик якості ML-моделей

4.2.2 Матриця помилок та класова збалансованість

Інтегральні метрики не показують, чи не схиляється модель до прогнозування переважаючого класу через дисбаланс 53,4% / 46,6%. Щоб це перевірити, для найкращої моделі (Random Forest) побудовано матрицю помилок на тестовій вибірці 30 000 матчів (таблиця 4.3).

Таблиця 4.3 - Матриця помилок Random Forest (тестова вибірка, 30 000 матчів)

	Прогноз Dire	Прогноз Radiant
Факт Dire	7 759 (TN)	6 213 (FP)
Факт Radiant	6 992 (FN)	9 036 (TP)

На основі матриці обчислено метрики окремо для кожного класу (таблиця 4.4).

Таблиця 4.4 - Precision, Recall та F1 по класах (Random Forest)

Клас	Precision	Recall	F1
Dire (0)	0,5260	0,5553	0,5403
Radiant (1)	0,5926	0,5638	0,5778

Ключовим є показник recall, який відображає частку правильно виявлених перемог кожної сторони. Для Dire він становить 0,5553, для Radiant - 0,5638; різниця менша за один відсотковий пункт. Така збалансованість підтверджує, що коригування ваг класів (class_weight='balanced') є ефективним: модель розпізнає перемоги обох сторін майже однаково добре й не ігнорує менш представлений клас Dire. Це додатково видно з розподілу прогнозів - модель передбачила перемогу Dire у 14 751 матчі та Radiant у 15 249, тобто близько до співвідношення 50/50, а не до базового 53,4% / 46,6%.

Вища precision для класу Radiant (0,5926 проти 0,5260) пояснюється природною перевагою базової частоти: оскільки перемог Radiant у вибірці

більше, кожен прогноз «Radiant» статистично частіше виявляється правильним. Усі чотири метрики перебувають у діапазоні 0,53-0,59, що узгоджується з отриманою точністю 55,98% та теоретичною межею pick-based прогнозу й свідчить про помірну, але стабільну прогностичну здатність моделі вище рівня випадкового вгадування.

4.2.3 Аналіз важливості ознак

Random Forest надає можливість кількісно оцінити внесок кожної ознаки у прогноз через середнє зменшення критерію нечистоти Джині при розбитті вузлів по всіх деревах ансамблю. Чим більше розбиття за даною ознакою зменшує нечистоту в середньому, тим важливішою вважається ознака. Топ-15 найважливіших ознак за результатами навчання наведено в таблиці 4.5.

Таблиця 4.5 - Важливість ознак Random Forest (топ-15)

Ранг	Ознака	Важливість	Тип
1	Radiant_Counter_Dire	22,24%	Числова
2	Radiant_Synergy	10,88%	Числова
3	Dire_Synergy	10,48%	Числова
4	Diff_WinRate	6,67%	Числова
5	Radiant_WinRate_Mean	4,89%	Числова
6	Dire_WinRate_Mean	4,74%	Числова
7	Dire_WinRate_Min	2,45%	Числова
8	Radiant_PickRate	2,44%	Числова
9	Radiant_WinRate_Min	2,39%	Числова
10	Diff_PickRate	2,34%	Числова
11	Dire_PickRate	2,32%	Числова
12	Radiant_WinRate_Max	2,20%	Числова
13	Dire_WinRate_Max	2,11%	Числова
14	Hero_R_74 (Invoker)	0,23%	Категоріальна
15	Hero_R_86 (Rubick)	0,22%	Категоріальна

Результати аналізу виявляють кілька принципових закономірностей, що підтверджують правильність обраної методології формування вектора ознак.

По-перше, ознака Radiant_Counter_Dire є найінформативнішою з важливістю 22,24% - вона майже вдвічі перевищує наступну за значущістю ознаку. Ця ознака агрегує взаємодію між конкретними героями різних команд по всіх $5 \times 5 = 25$ парах, що робить її найбільш комплексним показником відповідності складів. Цей результат підтверджує ключовий принцип Dota 2: хто кого контрить у фазі драфту має більший вплив на результат, ніж індивідуальна сила героїв.

По-друге, три ознаки взаємодій між героями - CTR (22,24%), SYN(R) (10,88%) та SYN(D) (10,48%) - разом забезпечують 43,60% сукупної важливості моделі. Ці три ознаки займають лише 3 позиції з 413 у векторі (0,73% розмірності), але несуть майже половину всієї інформації, яку використовує модель. Це є сильним аргументом на користь включення статистик взаємодій до вектора ознак.

По-третє, усі 13 числових ознак разом забезпечують близько 76% важливості, тоді як 400 категоріальних ознак - лише близько 24%. Кожна категоріальна ознака вносить у середньому лише 0,06% важливості. Це свідчить, що факт присутності конкретного героя у складі є відносно слабким предиктором без урахування його взаємодій з іншими персонажами. Проте сукупний внесок категоріальних ознак (24%) є достатнім, щоб виправдати їхнє включення, - вони дозволяють моделі враховувати специфіку окремих героїв, чого числові агрегати не можуть.

Invoker (hero_id = 74, ознака Hero_R_74) і Rubick (hero_id = 86, ознака Hero_R_86) займають перші місця серед категоріальних ознак. Invoker - герой із 26 унікальними заклинаннями, що потребує значного досвіду для ефективної гри. Rubick - підтримуючий герой, здатний красти заклинання ворогів. Обидва є одними з найпопулярніших і водночас найзалежніших від рівня гравця персонажів, тож їхня присутність у складі є значущим сигналом про стиль і рівень команди.

4.3 Динаміка точності від обсягу даних

Для дослідження залежності якості від обсягу навчальних даних усі п'ять підходів оцінювалися при різних обсягах навчальної підвибірки при фіксованій тестовій вибірці 30 000 матчів. Незмінність тестової вибірки робить результати різних рядків безпосередньо порівнянними. Експеримент дозволяє оцінити, чи є поточний обсяг даних достатнім, і показує, наскільки швидко кожен метод покращується зі зростанням навчальної вибірки. Результати наведено в таблиці 4.6.

Таблиця 4.6 - Динаміка accuracy методів залежно від обсягу навчальної вибірки

Навчальних матчів	Baseline	LR	RF	XGB	MLP
10 000	53,98%	51,71%	53,45%	52,15%	52,23%
40 000	55,18%	52,90%	54,88%	53,40%	52,62%
80 000	55,23%	53,89%	55,75%	54,29%	54,45%
120 000	55,27%	54,40%	55,98%	55,24%	54,81%

Усі моделі оцінювалися на одній і тій самій фіксованій тестовій вибірці 30 000 матчів. Рядок 120 000 збігається з основними результатами таблиці 4.2, оскільки відповідає навчанню на повному обсязі.

Аналіз динаміки виявляє кілька важливих закономірностей.

При малому обсязі навчання (10 000 матчів) найкращим виявляється baseline (53,98%), а всі моделі машинного навчання йому поступаються: даних ще недостатньо, щоб виявити закономірності, складніші за просте усереднення winrate команд. Зі зростанням навчальної вибірки ML-методи покращуються швидше за baseline і поступово його переганяють.

Random Forest є найкращою моделлю при повному обсязі (55,98%) і переганяє baseline приблизно з 80 000 матчів (55,75% проти 55,23%). Найбільший приріст RF припадає на ранній діапазон (53,45% → 54,88% при

зростанні від 10 до 40 тисяч), після чого зростання сповільнюється (55,75% → 55,98% від 80 до 120 тисяч). Водночас показник ROC-AUC для RF зростає монотонно - від 0,5465 до 0,5801, - тобто навіть коли приріст accuracy сповільнюється, збільшення даних продовжує покращувати калібровку ймовірностей. Спадні прирости свідчать, що при поточному наборі з 413 ознак модель наближається до насичення, і подальше збільшення вибірки без розширення ознак дасть лише незначне покращення.

Baseline зростає найменше серед усіх методів (+1,29% від 10 до 120 тисяч) і найшвидше виходить на плато: від 40 до 120 тисяч його точність майже не змінюється (55,18% → 55,27%). Це пояснюється тим, що winrate героїв стабілізується вже на помірних обсягах, і нові матчі мало змінюють накопичену статистику.

Логістична регресія демонструє суттєвий приріст (+2,69%), але стабільно поступається baseline на всіх обсягах. При малій вибірці 400 слабких бінарних ознак вносять значний шум в оцінку параметрів, що подавляється сильною регуляризацією; при збільшенні даних закон великих чисел дозволяє виявити навіть слабкі статистичні сигнали через ці ознаки. Проте LR так і не досягає рівня baseline, що підтверджує принципове обмеження лінійних методів для нелінійної задачі.

XGBoost демонструє найбільший приріст серед усіх методів (+3,09%): при 10 тисячах матчів він поступається навіть baseline (52,15%), але при 120 тисячах майже наздоганяє RF (55,24%). Це відповідає теоретичним очікуванням - послідовний бустинг потребує більшої кількості прикладів для стабілізації, ніж паралельний bagging, оскільки кожне нове дерево навчається на залишках попередніх, які при малих вибірках містять багато шуму.

MLP зростає рівномірно (+2,58%), але теж не перевищує baseline: на малих вибірках механізм ранньої зупинки спрацьовує надто рано через швидке перенавчання на валідаційній частині.

Загальна тенденція: зі зростанням навчальної вибірки розрив між найкращим і найгіршим методом скорочується з 2,27% (при 10 тисячах) до

1,17% (при 120 тисячах). Це свідчить, що при достатньому обсязі даних результат визначається переважно набором ознак, а не вибором алгоритму. Усі методи стабілізуються у вузькому діапазоні приблизно 54-56%, що узгоджується з теоретичною межею pick-based прогнозу.

4.4 Тестування системи

Функціональне тестування системи проводилося на трьох рівнях: тестування окремих компонентів в ізоляції, тестування інтеграції між компонентами й тестування веб-інтерфейсу з реальними вхідними даними.

Тестування модуля збору. Перевірялася коректність фільтрації за всіма шістьма критеріями. Для кожного критерію добиралися матчі, що його порушують: нерейтингові матчі (game_mode = 1), матчі рівня Archon (avg_rank_tier = 40), матчі тривалістю 12 хвилин, матчі з менш ніж 5 визначеними рангами. Перевірено, що жоден із таких матчів не потрапив до бази. Ідемпотентність збору перевірена запуском скрипту двічі поспіль: кількість рядків у таблиці matches після другого запуску збіглася з першим, оскільки INSERT OR IGNORE ігнорує дублікати. Відсутність дублікатів підтверджена SQL-запитом: SELECT COUNT(*) і SELECT COUNT(DISTINCT id) повернули однакове значення 150 000.

Тестування модуля статистики. Winrate кількох популярних героїв порівнювався з даними публічної платформи Dotabuff для патчу 7.41. Відхилення в межах 1-2% є очікуваним і пояснюється різним обсягом вибірки та часовими рамками матчів. Наприклад, герой Pudge (персонаж-танк, відомий своїм гачком, що захоплює ворогів на великій відстані) стабільно входить до топ-3 за pickrate в обох системах. Коректність механізму баєсівського згладжування перевірена на прикладах рідкісних пар героїв: при менш ніж 10 спільних іграх їхній synergy winrate перебуває поблизу глобального winrate 0,534, а не набуває екстремальних значень.

Тестування ML-модуля. Відтворюваність результатів перевірена повторним запуском ml.py з random_state = 42: accuracy і ROC-AUC збіглися до останньої цифри. Коректність запобігання витоку даних перевірена порівнянням двох сценаріїв. У першому статистики winrate обчислювалися на всьому датасеті до розбивки, у другому - лише на тренувальній підвибірці після розбивки. У першому сценарії accuracy RF на тестовій вибірці штучно завищувалася приблизно на 0,3-0,5%, що підтверджує методологічну важливість правильного порядку операцій, описаного в розділі 3.5.

Тестування предиктора. Тестування проводилося на реальних матчах із відомим результатом через функцію завантаження за match_id. Для кількох завершених матчів вводився ідентифікатор, система завантажувала склад і генерувала прогноз, що порівнювався з фактичним результатом. Перевірено коректність обробки граничних випадків: введення тексту замість числа повертає повідомлення про помилку валідації; введення неіснуючого ідентифікатора - повідомлення, що матч не знайдено; вибір одного героя для обох команд - відповідне попередження без виконання прогнозу. Скріншот предиктора наведено на рисунку 4.3.

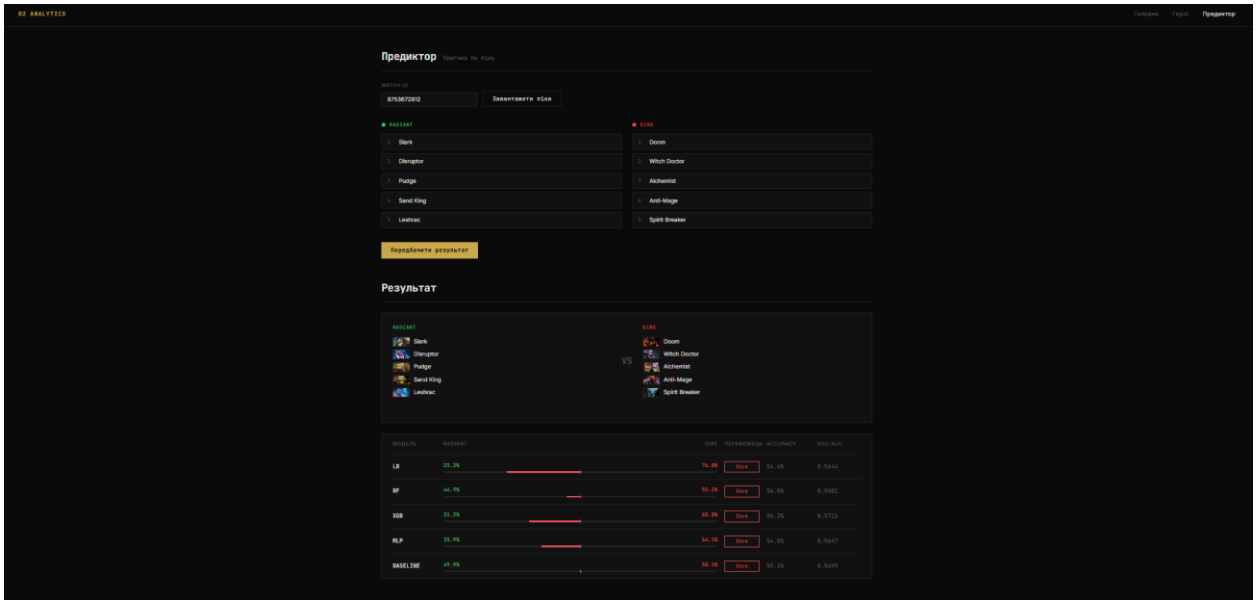


Рисунок 4.3 - Приклад роботи предиктора на реальному матчі

4.5 Результати та напрямки подальшого розвитку

Отримана точність 55,98% для Random Forest є закономірним результатом для задачі прогнозування виключно на основі складу команд без статистики гравців. Для порівняння: дослідження Kalyanaraman (2014) [5], яке прогнозувало результат матчу Dota 2 лише за піками команд, отримало точність близько 57% на значно меншій вибірці (~10 000 матчів). Conley та Perry (2013) [6] провели один із перших порівняльних аналізів методів машинного навчання для прогнозування та рекомендації героїв у Dota 2. Вищі показники точності в літературі досягаються при включенні ігрової статистики матчу та індивідуальних показників гравців, що є принципово ширшим набором вхідних даних, ніж лише склад команд.

Теоретична межа для pick-based прогнозу в Dota 2 оцінюється дослідниками на рівні 57-60%. Ця межа визначається тим, що значна частина результату матчу залежить від індивідуальної майстерності гравців і мікрорішень під час гри - факторів, які принципово не можуть бути передбачені на основі лише складу команд. Отримане значення 55,98% наближається до нижньої межі, незначне відставання закономірне з огляду на обмежений набір ознак.

Аналіз важливості ознак підтвердив правильність обраної методології формування вектора: числові метрики взаємодій (43,6% важливості при 0,73% розмірності) суттєво інформативніші за категоріальні one-hot ознаки. Це вказує на головний напрямок подальшого вдосконалення - розширення числової компоненти вектора.

Основні обмеження системи й напрямки розвитку пов’язані з трьома аспектами. По-перше, відсутність статистики гравців у вхідних даних. Включення MMR кожного учасника, його winrate на конкретному герої та загального KDA за останні ігри потенційно може підвищити точність до 62-65%. Для цього необхідний детальний парсинг кожного матчу через ендпоінт

/matches/{id}, що збільшує кількість API-запитів у 100 разів і потребує значно більше часу на збір.

По-друге, залежність від актуальності даних при зміні патчу. При виході нового патчу Valve змінює баланс героїв: winrate і синергії суттєво змінюються, і накопичена статистика стає застарілою. Розв'язком є фільтрація матчів за полем start_time для виключення матчів старих патчів та автоматичне перенавчання моделей при накопиченні достатньої кількості матчів нового патчу.

По-третє, продуктивність при масштабуванні аудиторії. Обчислені статистичні метрики кешуються в оперативній пам'яті з інвалідацією за кількістю матчів, що усуває повторні обчислення й забезпечує швидкий відгук при типовому навантаженні. Для масштабування на велику кількість одночасних користувачів подальшим напрямом є винесення кешу в зовнішнє сховище (наприклад, Redis) або періодичний перерахунок попередньо агрегованих показників за розкладом.

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі проведено комплексний аналіз результатів роботи розробленої системи та її тестування.

Охарактеризовано зібраний датасет обсягом 150 000 рейтингових матчів Dota 2 патчу 7.41. Розподіл результатів 53,4% / 46,6% на користь Radiant підтверджує відому асиметрію карти й врахований при налаштуванні параметрів навчання всіх моделей.

За результатами порівняння п’яти підходів найкращу ефективність продемонстрував Random Forest з accuracy 55,98% та ROC-AUC 0,5801 - єдиний метод, що стабільно перевищує базову модель за обома метриками. Аналіз матриці помилок показав збалансований recall для обох класів (0,5553 для Dire і 0,5638 для Radiant), що підтверджує коректність коригування ваг класів і відсутність зміщення моделі в бік переважаючого класу. Аналіз важливості ознак виявив, що три метрики взаємодій між героями (CTR і дві SYN) забезпечують 43,60% сукупної важливості моделі при тому, що займають лише 0,73% розмірності вектора. Це підтверджує правильність методології формування ознак і вказує на напрямок подальшого вдосконалення через розширення числової компоненти.

Дослідження динаміки точності від обсягу даних показало, що прирости сповільнюються зі зростанням навчальної вибірки - для поточного набору ознак модель наближається до насичення. Аналіз усіх п'яти підходів виявив, що XGBoost має найбільший потенціал зростання (+3,09% від 10 до 120 тисяч матчів), а при малих обсягах усі ML-методи поступаються простому baseline, який залишається конкурентним приблизно до 80 тисяч матчів. Функціональне тестування підтвердило коректність роботи всіх компонентів, включно з перевіркою відсутності витоку даних. Визначено три напрямки подальшого розвитку: включення статистики гравців, підтримку актуальності при зміні патчу та масштабування кешування статистики.

ВИСНОВКИ

У дипломному проєкті розроблено клієнт-серверну систему обробки та прогнозування результатів командних матчів Dota 2 на основі статистичних метрик. Поставлену мету досягнуто, а всі визначені у вступі завдання виконано в повному обсязі.

Розроблено модуль автоматизованого збору даних, який через публічний OpenDota API сформував датасет обсягом 150 000 рейтингових матчів рівня Ancient і вище. Реалізована шестиетапна фільтрація за критеріями якості та механізм відновлення пагінації забезпечили збір репрезентативної й однорідної вибірки. Спроектовано й реалізовано реляційну базу даних на SQLite з обґрунтованою структурою зберігання складів команд, що поєднує зручність SQL-запитів з ефективністю підготовки даних для машинного навчання.

Реалізовано модуль розрахунку статистичних метрик - winrate і pickrate героїв, синергій пар та контрпіків між командами - із баєсівським згладжуванням, що усуває статистичні артефакти при малій кількості спільних матчів. Реалізовано й порівняно п’ять підходів до прогнозування: базову модель, логістичну регресію, Random Forest, XGBoost і багатошаровий персептрон. Для кожного методу обґрунтовано математичну основу та вибір гіперпараметрів з урахуванням специфіки переважно бінарного простору ознак і дисбалансу класів.

Найкращу ефективність продемонстрував метод Random Forest з точністю 55,98% та ROC-AUC 0,5801 - єдиний метод, що стабільно перевищив базову модель за обома метриками. Збалансований recall для обох класів підтвердив відсутність зміщення моделі в бік переважаючого класу. Отримане значення точності наближається до нижньої теоретичній межі pick-based прогнозу в Dota 2 (57-60%) і узгоджується з результатами наявних досліджень, що підтверджує коректність реалізації.

Основним науковим результатом роботи є кількісне встановлення прогностичної здатності складу команд: самі лише піки забезпечують точність близько 56%, а решта варіативності результату визначається майстерністю гравців і перебігом гри, що не піддається прогнозу до початку матчу. Аналіз важливості ознак виявив, що метрики взаємодії героїв (контрпіки та синергії) несуть майже половину всієї інформації моделі, займаючи менше одного відсотка розмірності вектора, - це підтверджує, що відповідність складів важливіша за індивідуальну силу окремих героїв. Показово, що складніші моделі (XGBoost, MLP) не перевершили простіший Random Forest: це свідчить про обмеженість сигналу у вхідних даних, який майже повністю вичерпується агрегованими статистичними ознаками.

Практична цінність роботи полягає у створенні функціональної веб-системи, яка, на відміну від наявних аналітичних платформ, поєднує комплексний розрахунок метрик взаємодії героїв із прогнозуванням результату матчу та порівнянням ефективності різних алгоритмів. Зібраний датасет і навчені моделі можуть слугувати основою для подальших досліджень. Визначено перспективні напрямки розвитку: включення статистики гравців для підвищення точності, підтримку актуальності даних при зміні ігрових патчів та оптимізацію продуктивності для масштабування аудиторії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Dota 2 - статистика гравців [Електронний ресурс] // SteamDB. - Режим доступу: https://steamdb.info/app/570/charts/.
2. Dotabuff - Dota 2 Statistics [Електронний ресурс]. - Режим доступу: https://www.dotabuff.com/.
3. OpenDota [Електронний ресурс]. - Режим доступу: https://www.opendota.com/.
4. Stratz [Електронний ресурс]. - Режим доступу: https://stratz.com/.
5. Kalyanaraman K. To win or not to win? A prediction model to determine the outcome of a DotA2 match : Technical Report. - San Diego : University of California, 2014. - 5 p.
6. Conley K., Perry D. How Does He Saw Me? A Recommendation Engine for Picking Heroes in Dota 2 : Technical Report. - Stanford : Stanford University, 2013. - 7 p.
7. Breiman L. Random Forests // Machine Learning. - 2001. - Vol. 45, No. 1. - P. 5-32.
8. Friedman J. H. Greedy Function Approximation: A Gradient Boosting Machine // The Annals of Statistics. - 2001. - Vol. 29, No. 5. - P. 1189-1232.
9. Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. - New York : ACM, 2016. - P. 785-794.
10. Hornik K., Stinchcombe M., White H. Multilayer Feedforward Networks are Universal Approximators // Neural Networks. - 1989. - Vol. 2, No. 5. - P. 359-366.
11. Pedregosa F. et al. Scikit-learn: Machine Learning in Python // Journal of Machine Learning Research. - 2011. - Vol. 12. - P. 2825-2830.

12. FastAPI Documentation [Електронний ресурс]. - Режим доступу: https://fastapi.tiangolo.com/.
13. SQLite Documentation [Електронний ресурс]. - Режим доступу: https://www.sqlite.org/docs.html.
14. Harris C. R. et al. Array Programming with NumPy // Nature. - 2020. - Vol. 585. - P. 357-362.
15. Python 3 Documentation [Електронний ресурс]. - Режим доступу: https://docs.python.org/3/.
16. Dota 2 [Електронний ресурс] // Valve Corporation. - Режим доступу: https://www.dota2.com/.
17. Fawcett T. An Introduction to ROC Analysis // Pattern Recognition Letters. - 2006. - Vol. 27, No. 8. - P. 861-874.
18. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. - 2nd ed. - New York : Springer, 2009. - 745 p.
19. Breiman L. Bagging Predictors // Machine Learning. - 1996. - Vol. 24, No. 2. - P. 123-140.
20. Kingma D. P., Ba J. Adam: A Method for Stochastic Optimization // 3rd International Conference on Learning Representations (ICLR). - San Diego, 2015. - 15 p.
21. Glorot X., Bordes A., Bengio Y. Deep Sparse Rectifier Neural Networks // Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS). - 2011. - P. 315-323.
22. Pydantic Documentation [Електронний ресурс]. - Режим доступу: https://docs.pydantic.dev/.
23. Jinja Documentation [Електронний ресурс] // Pallets Projects. - Режим доступу: https://jinja.palletsprojects.com/.
24. Uvicorn Documentation [Електронний ресурс]. - Режим доступу: https://www.uvicorn.org/.

25. Requests: HTTP for Humans - Documentation [Електронний ресурс]. - Режим доступу: https://requests.readthedocs.io/.

26. Bishop C. M. Pattern Recognition and Machine Learning. - New York : Springer, 2006. - 738 p.

27. Gelman A., Carlin J. B., Stern H. S. et al. Bayesian Data Analysis. - 3rd ed. - Boca Raton : CRC Press, 2013. - 675 p.

28. Chart.js Documentation [Електронний ресурс]. - Режим доступу: https://www.chartjs.org/docs/.

ДОДАТОК А

Клієнт-серверна система обробки та прогнозування результатів командних матчів на основі статистичних метрик

Алгоритм програми

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2026 р

					ІАЛЦ.467200.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Васильєв В.А.			**Клієнт-серверна система обробки та прогнозування результатів командних матчів на основі статистичних метрик** Алгоритм програми	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Хмельницький А.А.					
Н. Контр.		Коренко Д.В.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-23	
Затвердив							

ДОДАТОК Б

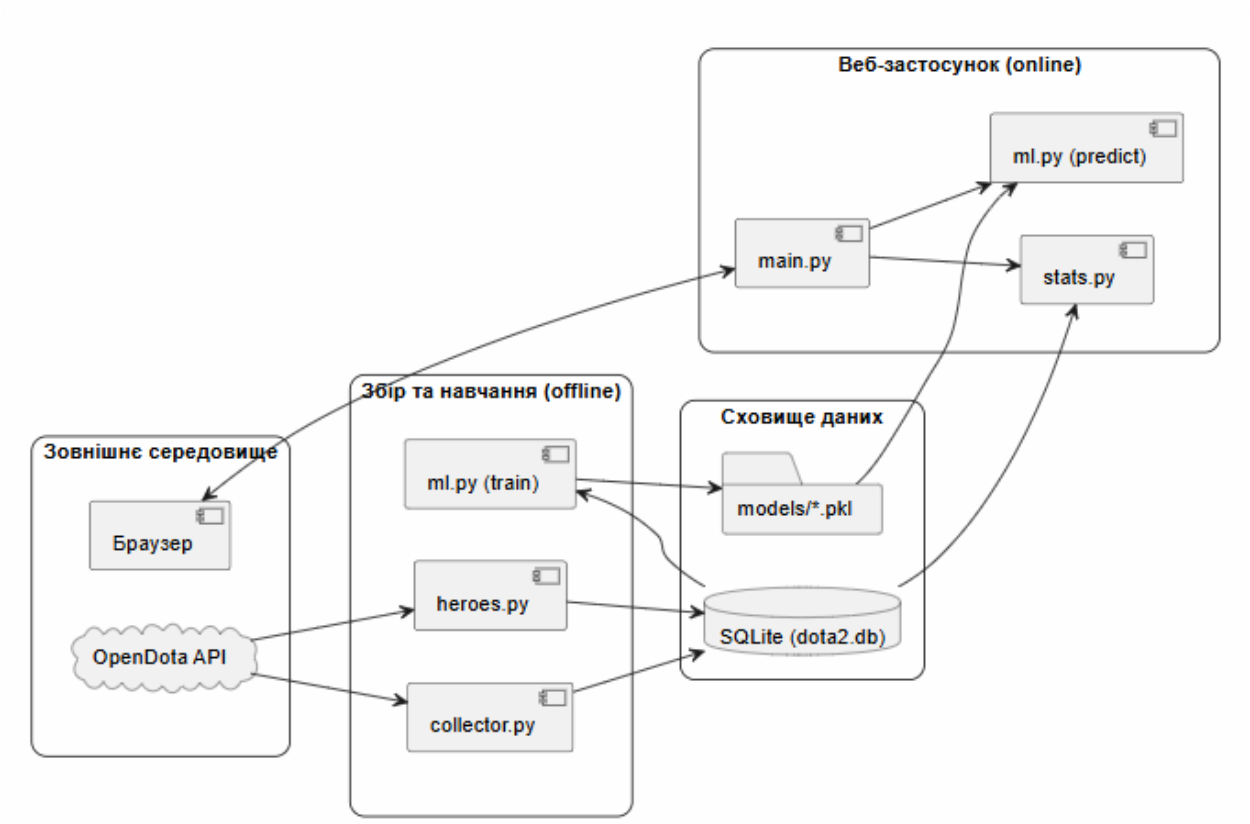
Клієнт-серверна система обробки та прогнозування результатів командних матчів на основі статистичних метрик

Діаграма компонентів

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2026 р



					ІАЛЦ.467200.005 Д2	
		№ докум.	Підпис	Дата		
Розробив		Васильєв В.А.			**Клієнт-серверна система обробки та прогнозування результатів командних матчів на основі статистичних метрик** Діаграма компонентів	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Хмельницький А.А.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-23
Н. Контр.		Коренко Д.В.				
Затвердив						

ДОДАТОК В

Клієнт-серверна система обробки та прогнозування результатів командних матчів на основі статистичних метрик

Діаграма варіантів використання

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2026 р

					ІАЛЦ.467200.006 Д3		
		№ докум.	Підпис	Дата			
Розробив		Васильєв В.А.			**Клієнт-серверна система обробки та прогнозування результатів командних матчів на основі статистичних метрик** Діаграма варіантів використання	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Хмельницький А.А.					
Н. Контр.		Коренко Д.В.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-23	
Затвердив							

ДОДАТОК Г

Клієнт-серверна система обробки та прогнозування результатів командних матчів на основі статистичних метрик

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 72

Київ 2026 р

.env.example:

```
# Шлях до файлу бази даних
DB_PATH=dota2.db

# Директорія зі збереженими моделями
MODELS_DIR=models

# Адреса OpenDota API
OPENDOTA_API=https://api.opendota.com/api

# Цільова кількість матчів для збору
TARGET_MATCHES=150000

# Затримка між запитами до API, секунди
REQUEST_DELAY=1.3

# Мінімальний середній ранг (60 = Ancient, 70 = Divine, 80 = Immortal)
MIN_RANK_TIER=60

# Тривалість матчу, хвилини
MIN_DURATION_MINUTES=20
MAX_DURATION_MINUTES=90

# Розмір простору ідентифікаторів героїв для one-hot
HERO_COUNT=200
```

config.py:

```
"""
Конфігурація проєкту. Значення читаються з файлу .env,
а за його відсутності використовуються значення за замовчуванням.
"""

import os
from dotenv import load_dotenv

load_dotenv()

# Шляхи
DB_PATH = os.getenv("DB_PATH", "dota2.db")
MODELS_DIR = os.getenv("MODELS_DIR", "models")

# OpenDota API
OPENDOTA_API = os.getenv("OPENDOTA_API", "https://api.opendota.com/api")

# Параметри збору даних
TARGET_MATCHES = int(os.getenv("TARGET_MATCHES", "150000"))
REQUEST_DELAY = float(os.getenv("REQUEST_DELAY", "1.3"))

# Критерії фільтрації матчів
MIN_RANK_TIER = int(os.getenv("MIN_RANK_TIER", "60"))            # 60 = Ancient
MIN_DURATION = int(os.getenv("MIN_DURATION_MINUTES", "20")) * 60
MAX_DURATION = int(os.getenv("MAX_DURATION_MINUTES", "90")) * 60

# Розмір простору героїв для one-hot кодування
HERO_COUNT = int(os.getenv("HERO_COUNT", "200"))
```

requirements.txt:

```
fastapi==0.111.0
uvicorn==0.30.1
jinja2==3.1.4
requests==2.32.3
python-dotenv==1.0.1
numpy==1.26.4
scikit-learn==1.5.0
xgboost==2.1.0
```

collector.py:

```
"""
Збір рейтингових матчів Dota 2 через OpenDota API (ендпоінт /publicMatches).
Запуск: python collector.py
"""

import sqlite3
import time
import logging

import requests

from config import (
    DB_PATH, OPENDOTA_API, TARGET_MATCHES, REQUEST_DELAY,
    MIN_RANK_TIER, MIN_DURATION, MAX_DURATION,
)

API_URL = f"{OPENDOTA_API}/publicMatches"

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s  %(message)s",
    datefmt="%H:%M:%S",
```

```
)
log = logging.getLogger(__name__)


def init_db(conn):
    """Створює таблицю matches, якщо її ще немає."""
    conn.execute("""
        CREATE TABLE IF NOT EXISTS matches (
            id            INTEGER PRIMARY KEY,
            radiant_win   INTEGER NOT NULL,
            duration      INTEGER NOT NULL,
            avg_rank_tier INTEGER,
            num_rank_tier INTEGER,
            game_mode     INTEGER,
            start_time    INTEGER,
            r1 INTEGER, r2 INTEGER, r3 INTEGER, r4 INTEGER, r5 INTEGER,
            d1 INTEGER, d2 INTEGER, d3 INTEGER, d4 INTEGER, d5 INTEGER
        )
    """)
    conn.commit()


def get_existing_ids(conn):
    """Повертає множину id матчів, уже збережених у базі."""
    rows = conn.execute("SELECT id FROM matches").fetchall()
    return {row[0] for row in rows}


def save_match(conn, m):
    """Зберігає один матч; дублікати ігноруються за первинним ключем."""
    r, d = m["r"], m["d"]
    conn.execute("""
        INSERT OR IGNORE INTO matches
            (id, radiant_win, duration, avg_rank_tier, num_rank_tier,
             game_mode, start_time,
```

```
            r1, r2, r3, r4, r5, d1, d2, d3, d4, d5)
        VALUES (?,?,?,?,?, ?,?, ?,?,?,?,?, ?,?,?,?,?)
    """, (
        m["match_id"],
        int(m["radiant_win"]),
        m["duration"],
        m.get("avg_rank_tier"),
        m.get("num_rank_tier"),
        m.get("game_mode"),
        m.get("start_time"),
        r[0], r[1], r[2], r[3], r[4],
        d[0], d[1], d[2], d[3], d[4],
    ))


def extract_picks(raw):
    """
    Дістає склади команд із сирого матчу. Повертає (radiant, dire) по
    5 героїв або None, якщо склад неповний, нульовий чи містить повтори.
    """
    rt = raw.get("radiant_team")
    dt = raw.get("dire_team")
    if not rt or not dt:
        return None
    try:
        if isinstance(rt, str):
            r = [int(x) for x in rt.split(",") if x.strip()]
            d = [int(x) for x in dt.split(",") if x.strip()]
        else:
            r, d = list(rt), list(dt)
        if len(r) == 5 and len(d) == 5 and all(x > 0 for x in r + d) \
                and len(set(r + d)) == 10:
            return r, d
    except (ValueError, TypeError):
        pass
```

```
    return None


def is_quality(raw):
    """Перевіряє матч за критеріями якості датасету."""
    # має бути визначений результат
    if raw.get("radiant_win") is None:
        return False

    # тільки Ranked All Pick (22), без турбо й аркадних режимів
    if raw.get("game_mode") != 22:
        return False

    # ранг Ancient і вище, не менше 5 гравців із визначеним рангом
    rank = raw.get("avg_rank_tier") or 0
    num = raw.get("num_rank_tier") or 0
    if rank < MIN_RANK_TIER or num < 5:
        return False

    # адекватна тривалість
    duration = raw.get("duration", 0)
    if not (MIN_DURATION <= duration <= MAX_DURATION):
        return False

    # повний коректний склад обох команд
    if extract_picks(raw) is None:
        return False

    return True


def fetch_matches(less_than_match_id=None):
    """Запитує партію матчів із API, пріоритезуючи вищий MMR."""
    params = {"mmr_descending": 1}
    if less_than_match_id:
```

```
        params["less_than_match_id"] = less_than_match_id

    try:
        resp = requests.get(API_URL, params=params, timeout=15)
        resp.raise_for_status()
        return resp.json()
    except requests.exceptions.HTTPError:
        if resp.status_code == 429:
            log.warning("Rate limit - чекаємо 60 секунд...")
            time.sleep(60)
        else:
            log.warning("HTTP помилка: %d", resp.status_code)
        return []
    except Exception as e:
        log.warning("Помилка запиту: %s", e)
        return []


def collect():
    """Основний цикл збору: тягне матчі партіями, поки не набереться
TARGET_MATCHES."""
    conn = sqlite3.connect(DB_PATH)
    init_db(conn)

    existing_ids = get_existing_ids(conn)
    collected = len(existing_ids)
    less_than_id = None
    total_fetched = 0
    total_skipped = 0
    empty_streak = 0

    log.info("Старт. Вже в БД: %d матчів. Ціль: %d", collected, TARGET_MATCHES)

    while collected < TARGET_MATCHES:
        batch = fetch_matches(less_than_id)
```

```
        if not batch:
            log.warning("Порожня відповідь, повтор через 3с...")
            time.sleep(3)
            continue

        total_fetched += len(batch)
        less_than_id = min(m["match_id"] for m in batch)

        saved_this_batch = 0
        for raw in batch:
            mid = raw.get("match_id")
            if mid in existing_ids:
                continue
            if not is_quality(raw):
                total_skipped += 1
                continue
            r, d = extract_picks(raw)
            save_match(conn, {
                "match_id": mid,
                "radiant_win": raw["radiant_win"],
                "duration": raw["duration"],
                "avg_rank_tier": raw.get("avg_rank_tier"),
                "num_rank_tier": raw.get("num_rank_tier"),
                "game_mode": raw.get("game_mode"),
                "start_time": raw.get("start_time"),
                "r": r, "d": d,
            })
            existing_ids.add(mid)
            saved_this_batch += 1

        conn.commit()
        collected += saved_this_batch

        # якщо кілька партій підряд нічого не дали - скидаємо пагінацію
        if saved_this_batch == 0:
```

```
            empty_streak += 1
            if empty_streak >= 5:
                log.warning("5 партій без збережень - скидаємо пагінацію")
                time.sleep(30)
                less_than_id = None
                empty_streak = 0
        else:
            empty_streak = 0

        log.info(
            "Збережено: %d  |  Всього: %d/%d  |  З API: %d  |  Відфільтровано: %d",
            saved_this_batch, collected, TARGET_MATCHES,
            total_fetched, total_skipped,
        )

        time.sleep(REQUEST_DELAY)

    conn.close()
    log.info("Готово! Зібрано %d матчів у %s", collected, DB_PATH)


if __name__ == "__main__":
    collect()
```

heroes.py:

```
"""
Одноразове завантаження довідника героїв з OpenDota у таблицю heroes.
Запуск: python heroes.py
"""

import sqlite3

import requests
```

```
from config import DB_PATH, OPENDOTA_API


def init_heroes_table(conn):
    """Створює таблицю heroes, якщо її ще немає."""
    conn.execute("""
        CREATE TABLE IF NOT EXISTS heroes (
            id             INTEGER PRIMARY KEY,
            name           TEXT NOT NULL,
            localized_name TEXT NOT NULL,
            primary_attr   TEXT,
            attack_type    TEXT,
            roles          TEXT,
            img            TEXT
        )
    """)
    conn.commit()


def main():
    conn = sqlite3.connect(DB_PATH)
    init_heroes_table(conn)

    print("Завантаження героїв з OpenDota...")
    heroes = requests.get(f"{OPENDOTA_API}/heroes", timeout=15).json()

    for h in heroes:
        short_name = h["name"].replace("npc_dota_hero_", "")
        img = (
            "https://cdn.cloudflare.steamstatic.com/apps/dota2/images/"
            f"dota_react/heroes/{short_name}.png"
        )
        roles = ",".join(h.get("roles", []))

        conn.execute("""
```

```
            INSERT OR REPLACE INTO heroes
                (id, name, localized_name, primary_attr, attack_type, roles, img)
            VALUES (?, ?, ?, ?, ?, ?, ?)
        """, (
            h["id"],
            h["name"],
            h["localized_name"],
            h.get("primary_attr", ""),
            h.get("attack_type", ""),
            roles,
            img,
        ))

    conn.commit()

    count = conn.execute("SELECT COUNT(*) FROM heroes").fetchone()[0]
    print(f"Збережено {count} героїв")

    # коротка перевірка результату
    rows = conn.execute(
        "SELECT id, localized_name, primary_attr, roles FROM heroes LIMIT 5"
    ).fetchall()
    for r in rows:
        print(f"  id={r[0]:>4}  {r[1]:<20}  {r[2]:<4}  {r[3]}")

    conn.close()


if __name__ == "__main__":
    main()
```

stats.py:

```
"""
Розрахунок аналітичних метрик по матчах для веб-інтерфейсу:
winrate і pickrate героїв, синергії пар, контрпіки, розподіл тривалості
та winrate Radiant за рангом.

Результати кешуються в пам'яті й перераховуються лише тоді, коли
змінюється кількість матчів у базі (перевірка через COUNT).
"""

import sqlite3
from collections import defaultdict

from config import DB_PATH


def _connect():
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    return conn


# Кеш обчислених результатів та версія даних (= кількість матчів)
_CACHE = {}
_CACHE_VERSION = None


def _data_version() -> int:
    """Поточна версія даних - кількість матчів у базі."""
    conn = _connect()
    v = conn.execute("SELECT COUNT(*) FROM matches").fetchone()[0]
    conn.close()
    return v
```

```
def _memo(key, compute):
    """Віддає кешований результат; перераховує лише при зміні версії даних."""
    global _CACHE_VERSION
    version = _data_version()
    if version != _CACHE_VERSION:
        _CACHE.clear()
        _CACHE_VERSION = version
    if key not in _CACHE:
        _CACHE[key] = compute()
    return _CACHE[key]


def _load_matches():
    """Завантажує всі матчі в пам'ять (кешується між запитами)."""

    def _query():
        conn = _connect()
        rows = conn.execute("""
            SELECT radiant_win, duration, avg_rank_tier,
                   r1,r2,r3,r4,r5, d1,d2,d3,d4,d5
            FROM matches
        """).fetchall()
        conn.close()
        return rows

    return _memo("rows", _query)


def get_match_count() -> int:
    """Загальна кількість матчів у базі."""
    return _data_version()


def get_avg_duration() -> int:
```

```
    """Середня тривалість матчу у хвилинах (округлена)."""

    def _compute():
        conn = _connect()
        avg = conn.execute("SELECT AVG(duration) FROM matches").fetchone()[0]
        conn.close()
        return round(avg / 60) if avg else 0

    return _memo("avg_duration", _compute)


def get_radiant_winrate() -> float:
    """Загальний winrate Radiant по всіх матчах (0.0-1.0)."""

    def _compute():
        conn = _connect()
        avg = conn.execute("SELECT AVG(radiant_win) FROM matches").fetchone()[0]
        conn.close()
        return round(avg, 4) if avg is not None else 0.0

    return _memo("radiant_winrate", _compute)


def _compute_hero_stats() -> dict:
    rows = _load_matches()
    total = len(rows)
    if total == 0:
        return {}

    picks = defaultdict(int)
    wins = defaultdict(int)

    for row in rows:
        radiant_win = row["radiant_win"]
        radiant = [row["r1"], row["r2"], row["r3"], row["r4"], row["r5"]]
```

```
        dire = [row["d1"], row["d2"], row["d3"], row["d4"], row["d5"]]

        for hid in radiant:
            picks[hid] += 1
            if radiant_win:
                wins[hid] += 1

        for hid in dire:
            picks[hid] += 1
            if not radiant_win:
                wins[hid] += 1

    result = {}
    for hid, p in picks.items():
        result[hid] = {
            "picks": p,
            "wins": wins[hid],
            "winrate": round(wins[hid] / p, 4) if p > 0 else 0.0,
            "pickrate": round(p / total, 4),
        }
    return result


def get_hero_stats() -> dict:
    """
    Для кожного героя рахує кількість піків, перемог, winrate і pickrate.
    Перемога зараховується незалежно від сторони (Radiant/Dire).
    """
    return _memo("hero_stats", _compute_hero_stats)


def get_top_heroes(by: str = "winrate", n: int = 20, min_picks: int = 20) -> list:
    """
    Топ-N героїв за winrate або pickrate. Параметр min_picks відсікає
    героїв із малою кількістю ігор (де winrate статистично незначущий).
```

```
    """
    stats = get_hero_stats()
    filtered = [
        {"hero_id": hid, **data}
        for hid, data in stats.items()
        if data["picks"] >= min_picks
    ]
    return sorted(filtered, key=lambda x: x[by], reverse=True)[:n]


def _compute_pair_accum() -> tuple:
    rows = _load_matches()
    pair_picks = defaultdict(int)
    pair_wins = defaultdict(int)

    for row in rows:
        radiant_win = row["radiant_win"]
        radiant = [row["r1"], row["r2"], row["r3"], row["r4"], row["r5"]]
        dire = [row["d1"], row["d2"], row["d3"], row["d4"], row["d5"]]

        for team, team_won in [(radiant, radiant_win), (dire, not radiant_win)]:
            for i in range(len(team)):
                for j in range(i + 1, len(team)):
                    pair = (min(team[i], team[j]), max(team[i], team[j]))
                    pair_picks[pair] += 1
                    if team_won:
                        pair_wins[pair] += 1

    return dict(pair_picks), dict(pair_wins)


def get_hero_pairs(n: int = 15, min_picks: int = 50) -> list:
    """
    Пари героїв однієї команди з найвищим спільним winrate.
    Пара ідентифікується відсортованим кортежем, тож (a, b) і (b, a) - одне.
```

```
    """
    pair_picks, pair_wins = _memo("pair_accum", _compute_pair_accum)

    result = []
    for (a, b), p in pair_picks.items():
        if p < min_picks:
            continue
        result.append({
            "hero_a": a,
            "hero_b": b,
            "picks": p,
            "winrate": round(pair_wins[(a, b)] / p, 4),
        })

    return sorted(result, key=lambda x: x["winrate"], reverse=True)[:n]


def _compute_counter_accum(hero_id: int) -> tuple:
    rows = _load_matches()
    counter_picks = defaultdict(int)
    counter_wins = defaultdict(int)

    for row in rows:
        radiant_win = row["radiant_win"]
        radiant = [row["r1"], row["r2"], row["r3"], row["r4"], row["r5"]]
        dire = [row["d1"], row["d2"], row["d3"], row["d4"], row["d5"]]

        if hero_id in radiant:
            won = not radiant_win  # перемогли суперники з Dire
            for cid in dire:
                counter_picks[cid] += 1
                if won:
                    counter_wins[cid] += 1
        elif hero_id in dire:
            won = radiant_win
```

```
            for cid in radiant:
                counter_picks[cid] += 1
                if won:
                    counter_wins[cid] += 1

    return dict(counter_picks), dict(counter_wins)


def get_counters(hero_id: int, n: int = 10, min_picks: int = 10) -> list:
    """
    Для заданого героя - список персонажів, що найчастіше його перемагають.
    winrate_vs - частка перемог героя-контера в матчах проти hero_id.
    """
    counter_picks, counter_wins = _memo(
        f"counter_accum:{hero_id}",
        lambda: _compute_counter_accum(hero_id),
    )

    result = []
    for cid, p in counter_picks.items():
        if p < min_picks:
            continue
        result.append({
            "counter_id": cid,
            "picks": p,
            "winrate_vs": round(counter_wins[cid] / p, 4),
        })

    return sorted(result, key=lambda x: x["winrate_vs"], reverse=True)[:n]


def _compute_duration_buckets() -> list:
    rows = _load_matches()

    buckets = {
```

```
        "20-25 хв": 0, "25-30 хв": 0, "30-35 хв": 0, "35-40 хв": 0,
        "40-45 хв": 0, "45-55 хв": 0, "55-90 хв": 0,
    }

    def label(sec):
        m = sec // 60
        if m < 25:  return "20-25 хв"
        if m < 30:  return "25-30 хв"
        if m < 35:  return "30-35 хв"
        if m < 40:  return "35-40 хв"
        if m < 45:  return "40-45 хв"
        if m < 55:  return "45-55 хв"
        return "55-90 хв"

    for row in rows:
        buckets[label(row["duration"])] += 1

    return [{"label": k, "count": v} for k, v in buckets.items()]


def get_duration_buckets() -> list:
    """Розподіл матчів за інтервалами тривалості (для стовпчастого графіка)."""
    return _memo("duration_buckets", _compute_duration_buckets)


def _compute_winrate_by_rank() -> list:
    rank_names = {6: "Ancient", 7: "Divine", 8: "Immortal"}
    rows = _load_matches()

    bucket_matches = defaultdict(int)
    bucket_wins = defaultdict(int)

    for row in rows:
        tier = row["avg_rank_tier"]
        if tier is None:
```

```
            continue
        bracket = tier // 10  # 72 -> 7 (Divine)
        if bracket not in rank_names:
            continue
        bucket_matches[bracket] += 1
        if row["radiant_win"]:
            bucket_wins[bracket] += 1

    result = []
    for bracket, name in rank_names.items():
        m = bucket_matches[bracket]
        if m == 0:
            continue
        result.append({
            "rank": name,
            "matches": m,
            "radiant_winrate": round(bucket_wins[bracket] / m, 4),
        })
    return result


def get_winrate_by_rank() -> list:
    """Winrate Radiant у розрізі рангового bracket (Ancient/Divine/Immortal)."""
    return _memo("winrate_by_rank", _compute_winrate_by_rank)


if __name__ == "__main__":
    print(f"Матчів у БД: {get_match_count()}")
    print(f"Середня тривалість: {get_avg_duration()} хв\n")

    print("Топ-5 за winrate (мін. 20 ігор):")
    for h in get_top_heroes(by="winrate", n=5):
        print(f"  hero {h['hero_id']:>4} | winrate {h['winrate'] * 100:.1f}% | picks
{h['picks']}")
```

```
    print("\nТоп-5 синергій:")
    for p in get_hero_pairs(n=5):
        print(f"  {p['hero_a']} + {p['hero_b']} | winrate {p['winrate'] * 100:.1f}% |
picks {p['picks']}")


    print("\nРозподіл тривалості:")
    for b in get_duration_buckets():
        print(f"  {b['label']}: {b['count']} матчів")
```

ml.py:

```
"""
Навчання і порівняння ML-моделей для прогнозування переможця матчу Dota 2
за складом команд. Реалізовано baseline, логістичну регресію, Random Forest,
XGBoost та MLP.
"""

import os
import pickle
import sqlite3
from collections import defaultdict

import numpy as np

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    accuracy_score, roc_auc_score,
    confusion_matrix, precision_recall_fscore_support,
)
from xgboost import XGBClassifier
```

```
from config import DB_PATH, MODELS_DIR, HERO_COUNT


def save_obj(obj, name: str) -> None:
    """Серіалізує об'єкт у models/<name>.pkl."""
    os.makedirs(MODELS_DIR, exist_ok=True)
    path = os.path.join(MODELS_DIR, f"{name}.pkl")
    with open(path, "wb") as f:
        pickle.dump(obj, f)
    print(f"  Збережено: {path}")


def load_obj(name: str):
    """Завантажує раніше збережений об'єкт із models/<name>.pkl."""
    with open(os.path.join(MODELS_DIR, f"{name}.pkl"), "rb") as f:
        return pickle.load(f)


def load_rows() -> list:
    """Зчитує всі матчі (результат + 10 героїв) як список кортежів."""
    conn = sqlite3.connect(DB_PATH)
    rows = conn.execute("""
        SELECT radiant_win, r1,r2,r3,r4,r5, d1,d2,d3,d4,d5
        FROM matches
    """).fetchall()
    conn.close()
    return rows


def compute_hero_winrates(rows: list) -> dict:
    """Winrate кожного героя (герої з менш ніж 5 іграми відкидаються)."""
    picks, wins = defaultdict(int), defaultdict(int)
    for row in rows:
        radiant_win = row[0]
        for hid in row[1:6]:
```

```
            picks[hid] += 1
            if radiant_win:
                wins[hid] += 1
        for hid in row[6:11]:
            picks[hid] += 1
            if not radiant_win:
                wins[hid] += 1
    return {hid: wins[hid] / picks[hid] for hid in picks if picks[hid] >= 5}


def compute_pickrates(rows: list) -> dict:
    """Pickrate кожного героя як частка матчів, де він присутній."""
    picks = defaultdict(int)
    total = len(rows)
    for row in rows:
        for hid in row[1:11]:
            picks[hid] += 1
    return {hid: picks[hid] / total for hid in picks}


def compute_interactions(rows: list) -> tuple:
    """
    Баєсівські оцінки синергій (пари однієї команди) і контрпіків
    (пари різних команд). Синергії згладжуються до глобального winrate,
    контрпіки - до нейтрального 0.5.
    """
    synergy_stats = defaultdict(lambda: [0, 0])
    counter_stats = defaultdict(lambda: [0, 0])

    for row in rows:
        radiant_win = int(row[0])
        radiant = sorted(row[1:6])
        dire = sorted(row[6:11])

        for i in range(5):
```

```
            for j in range(i + 1, 5):
                synergy_stats[(radiant[i], radiant[j])][1] += 1
                synergy_stats[(dire[i], dire[j])][1] += 1
                if radiant_win:
                    synergy_stats[(radiant[i], radiant[j])][0] += 1
                else:
                    synergy_stats[(dire[i], dire[j])][0] += 1

        for r_h in radiant:
            for d_h in dire:
                counter_stats[(r_h, d_h)][1] += 1
                counter_stats[(d_h, r_h)][1] += 1
                if radiant_win:
                    counter_stats[(r_h, d_h)][0] += 1
                else:
                    counter_stats[(d_h, r_h)][0] += 1

    global_wr = sum(1 for r in rows if r[0]) / len(rows)
    prior = global_wr * 10

    synergy = {k: (v[0] + prior) / (v[1] + 10) for k, v in synergy_stats.items()}
    counter = {k: (v[0] + 5) / (v[1] + 10) for k, v in counter_stats.items()}
    return synergy, counter


def _interaction_features(radiant, dire, synergy, counter) -> tuple:
    """Середня синергія кожної команди та середній контрпік Radiant проти Dire."""
    r_syn, d_syn, ctr = [], [], []
    for i in range(5):
        for j in range(i + 1, 5):
            r_syn.append(synergy.get(tuple(sorted([radiant[i], radiant[j]])), 0.5))
            d_syn.append(synergy.get(tuple(sorted([dire[i], dire[j]])), 0.5))
    for r_h in radiant:
        for d_h in dire:
            ctr.append(counter.get((r_h, d_h), 0.5))
```

```
    return float(np.mean(r_syn)), float(np.mean(d_syn)), float(np.mean(ctr))



def _build_num_vec(radiant, dire, winrates, pickrates, synergy, counter) -> list:
    """13 числових ознак: агрегати winrate/pickrate, їх різниці, синергії,
контрпік."""
    r_wrs = [winrates.get(h, 0.5) for h in radiant]
    d_wrs = [winrates.get(h, 0.5) for h in dire]
    r_pr = float(np.mean([pickrates.get(h, 0.0) for h in radiant]))
    d_pr = float(np.mean([pickrates.get(h, 0.0) for h in dire]))
    r_syn, d_syn, r_ctr = _interaction_features(radiant, dire, synergy, counter)

    return [
        float(np.mean(r_wrs)), float(np.mean(d_wrs)),
        float(np.max(r_wrs)), float(np.max(d_wrs)),
        float(np.min(r_wrs)), float(np.min(d_wrs)),
        r_pr, d_pr,
        float(np.mean(r_wrs)) - float(np.mean(d_wrs)),
        r_pr - d_pr,
        r_syn, d_syn, r_ctr,
    ]



def _build_cat_vec(radiant, dire) -> list:
    """One-hot вектор присутності героїв: HERO_COUNT позицій на кожну команду."""
    vec = [0] * (HERO_COUNT * 2)
    for hid in radiant:
        if 0 < hid < HERO_COUNT:
            vec[hid] = 1
    for hid in dire:
        if 0 < hid < HERO_COUNT:
            vec[HERO_COUNT + hid] = 1
    return vec
```

```
def rows_to_features(rows, winrates, pickrates, synergy, counter) -> tuple:
    """Перетворює рядки БД у матриці числових і категоріальних ознак та мітки."""
    X_num, X_cat, y = [], [], []
    for row in rows:
        radiant = row[1:6]
        dire = row[6:11]
        X_num.append(_build_num_vec(radiant, dire, winrates, pickrates, synergy,
counter))
        X_cat.append(_build_cat_vec(radiant, dire))
        y.append(int(row[0]))
    return (
        np.array(X_num, dtype=np.float32),
        np.array(X_cat, dtype=np.float32),
        np.array(y, dtype=np.int32),
    )


def _class_metrics(y_true, y_pred) -> dict:
    """Precision/Recall/F1 окремо для класів Dire (0) і Radiant (1)."""
    prec, rec, f1, _ = precision_recall_fscore_support(
        y_true, y_pred, labels=[0, 1], zero_division=0
    )
    return {
        "precision_dire": round(float(prec[0]), 4),
        "recall_dire": round(float(rec[0]), 4),
        "f1_dire": round(float(f1[0]), 4),
        "precision_radiant": round(float(prec[1]), 4),
        "recall_radiant": round(float(rec[1]), 4),
        "f1_radiant": round(float(f1[1]), 4),
    }


def baseline_predict(rows: list, winrates: dict) -> tuple:
    """Прогноз baseline: перемагає команда з вищим середнім winrate героїв."""
    y_true, y_pred, y_prob = [], [], []
```

```
    for row in rows:
        radiant_win = int(row[0])
        r_wr = float(np.mean([winrates.get(h, 0.5) for h in row[1:6]]))
        d_wr = float(np.mean([winrates.get(h, 0.5) for h in row[6:11]]))
        total = r_wr + d_wr
        prob = r_wr / total if total > 0 else 0.5
        y_true.append(radiant_win)
        y_pred.append(1 if prob >= 0.5 else 0)
        y_prob.append(prob)
    return y_true, y_pred, y_prob


def train() -> dict:
    """Повний пайплайн: розбивка, ознаки, навчання моделей і оцінювання."""
    print("Завантаження даних...")
    rows = load_rows()
    y_all = np.array([int(r[0]) for r in rows])

    # стратифікована розбивка зберігає пропорцію класів у train і test
    idx_train, idx_test = train_test_split(
        np.arange(len(rows)), test_size=0.2, random_state=42, stratify=y_all
    )
    train_rows = [rows[i] for i in idx_train]
    test_rows = [rows[i] for i in idx_test]

    # статистики рахуються тільки на train, щоб не було витоку даних
    winrates = compute_hero_winrates(train_rows)
    pickrates = compute_pickrates(train_rows)
    synergy, counter = compute_interactions(train_rows)

    save_obj(winrates, "baseline_winrates")
    save_obj(pickrates, "baseline_pickrates")
    save_obj(synergy, "baseline_synergy")
    save_obj(counter, "baseline_counter")
```

```
    X_num_tr, X_cat_tr, y_train = rows_to_features(train_rows, winrates, pickrates,
synergy, counter)
    X_num_te, X_cat_te, y_test = rows_to_features(test_rows, winrates, pickrates,
synergy, counter)


    # нормалізація лише числових ознак; one-hot уже в {0,1}
    scaler = StandardScaler()
    X_train_final = np.hstack((scaler.fit_transform(X_num_tr), X_cat_tr))
    X_test_final = np.hstack((scaler.transform(X_num_te), X_cat_te))
    save_obj(scaler, "scaler")


    print(f"  Матчів:       {len(rows)}")
    print(f"  Фічей (Num):  {X_num_tr.shape[1]}")
    print(f"  Фічей (Cat):  {X_cat_tr.shape[1]}")
    print(f"  Radiant wins: {y_all.sum()} ({y_all.mean() * 100:.1f}%)")
    print(f"  Train: {len(y_train)} | Test: {len(y_test)}\n")


    model_defs = {
        "lr": LogisticRegression(
            max_iter=1000, C=0.1, class_weight="balanced", random_state=42,
        ),
        "rf": RandomForestClassifier(
            n_estimators=300, max_depth=15, min_samples_leaf=5,
            class_weight="balanced", random_state=42, n_jobs=-1,
        ),
        "xgb": XGBClassifier(
            n_estimators=500, max_depth=6, learning_rate=0.01,
            subsample=0.7, colsample_bytree=0.7, reg_lambda=10.0,
            min_child_weight=10,
            scale_pos_weight=(len(y_train) - y_train.sum()) / y_train.sum(),
            random_state=42, eval_metric="logloss", verbosity=0,
        ),
        "mlp": MLPClassifier(
            hidden_layer_sizes=(128, 64), activation="relu", alpha=0.01,
            early_stopping=True, validation_fraction=0.1,
            n_iter_no_change=20, max_iter=500, random_state=42,
```

```
    ),
}

results = {}
trained = {}
preds = {}

for name, model in model_defs.items():
    print(f"Навчання: {name.upper()}...")
    model.fit(X_train_final, y_train)
    y_pred = model.predict(X_test_final)
    y_prob = model.predict_proba(X_test_final)[:, 1]
    results[name] = {
        "accuracy": round(accuracy_score(y_test, y_pred), 4),
        "roc_auc": round(roc_auc_score(y_test, y_prob), 4),
        **_class_metrics(y_test, y_pred),
    }
    trained[name] = model
    preds[name] = y_pred
    save_obj(model, name)

print("\nРахуємо Baseline...")
y_true_b, y_pred_b, y_prob_b = baseline_predict(test_rows, winrates)
results["baseline"] = {
    "accuracy": round(accuracy_score(y_true_b, y_pred_b), 4),
    "roc_auc": round(roc_auc_score(y_true_b, y_prob_b), 4),
    **_class_metrics(y_true_b, y_pred_b),
}

best_acc = max(v["accuracy"] for v in results.values())
print("\n" + "=" * 48)
print(f"{'Модель':<12} {'Accuracy':>10} {'ROC-AUC':>10}")
print("-" * 48)
for name in ["baseline", "lr", "rf", "xgb", "mlp"]:
    r = results[name]
```

```
        marker = " <" if r["accuracy"] == best_acc else ""
        print(f"{name.upper():<12} {r['accuracy'] * 100:>9.2f}%
{r['roc_auc']:>10.4f}{marker}")
    print("=" * 48)

    save_obj(results, "results")
    print("\nВсі моделі збережені в папці models/")

    # матриця помилок і per-class метрики найкращої моделі
    best_name = max(
        (n for n in results if n != "baseline"),
        key=lambda n: results[n]["accuracy"],
    )
    cm = confusion_matrix(y_test, preds[best_name], labels=[0, 1])
    print("\n" + "=" * 56)
    print(f"Матриця помилок - {best_name.upper()} (найкраща модель)")
    print("-" * 56)
    print(f"{'':<16}{'Прогноз Dire':>18}{'Прогноз Radiant':>20}")
    print(f"{'Факт Dire':<16}{cm[0][0]:>18}{cm[0][1]:>20}")
    print(f"{'Факт Radiant':<16}{cm[1][0]:>18}{cm[1][1]:>20}")
    print("=" * 56)

    rb = results[best_name]
    print(f"\n{'Клас':<12}{'Precision':>12}{'Recall':>10}{'F1':>10}")
    print("-" * 44)

print(f"{'Dire':<12}{rb['precision_dire']:>12.4f}{rb['recall_dire']:>10.4f}{rb['f1_di
re']:>10.4f}")

print(f"{'Radiant':<12}{rb['precision_radiant']:>12.4f}{rb['recall_radiant']:>10.4f}{
rb['f1_radiant']:>10.4f}")
    print("=" * 44)

    # важливість ознак Random Forest
    num_names = [
        "Radiant_WinRate_Mean", "Dire_WinRate_Mean",
        "Radiant_WinRate_Max", "Dire_WinRate_Max",
```

```
        "Radiant_WinRate_Min", "Dire_WinRate_Min",
        "Radiant_PickRate", "Dire_PickRate",
        "Diff_WinRate", "Diff_PickRate",
        "Radiant_Synergy", "Dire_Synergy",
        "Radiant_Counter_Dire",
    ]
    cat_names = (
            [f"Hero_R_{i}" for i in range(HERO_COUNT)] +
            [f"Hero_D_{i}" for i in range(HERO_COUNT)]
    )
    all_names = num_names + cat_names
    importances = trained["rf"].feature_importances_
    indices = np.argsort(importances)[::-1]

    print("\n" + "=" * 48)
    print("ТОП-15 найважливіших фічей (Random Forest)")
    print("-" * 48)
    for i in range(15):
        idx = indices[i]
        print(f"{i + 1:2d}. {all_names[idx]:<25} {importances[idx]:.4f}")
    print("=" * 48)

    return results


def predict(radiant_heroes: list, dire_heroes: list) -> dict:
    """Прогноз кожної моделі для заданих складів (по 5 героїв на команду)."""
    if len(radiant_heroes) != 5 or len(dire_heroes) != 5:
        raise ValueError("Потрібно рівно 5 героїв для кожної команди")

    winrates = load_obj("baseline_winrates")
    pickrates = load_obj("baseline_pickrates")
    synergy = load_obj("baseline_synergy")
    counter = load_obj("baseline_counter")
    scaler = load_obj("scaler")
```

```
    num_vec = _build_num_vec(radiant_heroes, dire_heroes, winrates, pickrates,
synergy, counter)
    cat_vec = _build_cat_vec(radiant_heroes, dire_heroes)

    X_num = np.array([num_vec], dtype=np.float32)
    X_cat = np.array([cat_vec], dtype=np.float32)
    X_input = np.hstack((scaler.transform(X_num), X_cat))

    results = {}
    for name in ["lr", "rf", "xgb", "mlp"]:
        prob = float(load_obj(name).predict_proba(X_input)[0][1])
        results[name] = {
            "radiant_win_prob": round(prob, 4),
            "prediction": "Radiant" if prob >= 0.5 else "Dire",
        }

    r_wr = float(np.mean([winrates.get(h, 0.5) for h in radiant_heroes]))
    d_wr = float(np.mean([winrates.get(h, 0.5) for h in dire_heroes]))
    total = r_wr + d_wr
    prob = r_wr / total if total > 0 else 0.5
    results["baseline"] = {
        "radiant_win_prob": round(prob, 4),
        "prediction": "Radiant" if prob >= 0.5 else "Dire",
    }

    return results


if __name__ == "__main__":
    train()

    print("\nТест предикту:")
    test = predict(
        radiant_heroes=[88, 6, 15, 83, 84],
```

```
        dire_heroes=[67, 35, 71, 129, 73],
    )
    print(f"{'Модель':<12} {'Prob Radiant':>14} {'Прогноз':>10}")
    print("-" * 40)
    for name in ["baseline", "lr", "rf", "xgb", "mlp"]:
        r = test[name]
        print(f"{name.upper():<12} {r['radiant_win_prob']:>13.4f}
{r['prediction']:>10}")
```

main.py:

```
"""
FastAPI веб-сервер аналітики Dota 2: головна сторінка зі статистикою,
сторінка героїв (winrate, синергії, контрпіки) та предиктор результату.
Запуск: uvicorn main:app --reload
"""

import sqlite3

import requests
from fastapi import FastAPI, Request, Form
from fastapi.responses import HTMLResponse
from jinja2 import Environment, FileSystemLoader
from starlette.templating import Jinja2Templates

import stats
import ml
from config import DB_PATH, OPENDOTA_API

app = FastAPI(title="Dota 2 Analytics")

jinja_env = Environment(
    loader=FileSystemLoader("templates"),
    autoescape=True,
)
```

```
templates = Jinja2Templates(env=jinja_env)



def get_heroes() -> list:
    """Список усіх героїв із БД для випадаючих списків і таблиць."""
    conn = sqlite3.connect(DB_PATH)
    rows = conn.execute(
        "SELECT id, localized_name, primary_attr, attack_type, roles, img "
        "FROM heroes ORDER BY localized_name"
    ).fetchall()
    conn.close()
    return [
        {
            "id": r[0],
            "name": r[1],
            "attr": r[2],
            "type": r[3],
            "roles": r[4].split(",") if r[4] else [],
            "img": r[5],
        }
        for r in rows
    ]



def get_hero_map() -> dict:
    """Відображення {hero_id: {name, img}} для швидкого пошуку за id."""
    conn = sqlite3.connect(DB_PATH)
    rows = conn.execute("SELECT id, localized_name, img FROM heroes").fetchall()
    conn.close()
    return {r[0]: {"name": r[1], "img": r[2]} for r in rows}



@app.get("/", response_class=HTMLResponse)
def index(request: Request):
    """Головна сторінка: загальна статистика, графіки та топ героїв."""
```

```
    hero_map = get_hero_map()
    results = ml.load_obj("results")

    # найкраща модель визначається тут, у Python
    if results:
        best_name = max(results, key=lambda k: results[k]["accuracy"])
        best_model = best_name.upper()
        best_acc = results[best_name]["accuracy"]
    else:
        best_model, best_acc = "-", 0

    top_winrate = stats.get_top_heroes(by="winrate", n=5, min_picks=100)
    top_pickrate = stats.get_top_heroes(by="pickrate", n=5, min_picks=100)
    for h in top_winrate + top_pickrate:
        h["name"] = hero_map.get(h["hero_id"], {}).get("name", f"Hero
{h['hero_id']}")
        h["img"] = hero_map.get(h["hero_id"], {}).get("img", "")

    return templates.TemplateResponse("index.html", {
        "request": request,
        "match_count": stats.get_match_count(),
        "top_winrate": top_winrate,
        "top_pickrate": top_pickrate,
        "duration": stats.get_duration_buckets(),
        "radiant_winrate": stats.get_radiant_winrate(),
        "ml_results": results,
        "avg_duration": stats.get_avg_duration(),
        "best_model": best_model,
        "best_acc": best_acc,
    })


@app.get("/heroes", response_class=HTMLResponse)
def heroes_page(request: Request, attr: str = "", role: str = "", counter_hero: int =
0):
    """Сторінка героїв: таблиця статистики, синергії та контрпіки."""
```

```
hero_map = get_hero_map()
hero_stats = stats.get_hero_stats()
heroes_db = get_heroes()

# таблиця героїв з фільтрами за атрибутом і роллю
table = []
for h in heroes_db:
    stat = hero_stats.get(h["id"])
    if not stat:
        continue
    if attr and h["attr"] != attr:
        continue
    if role and role not in h["roles"]:
        continue
    table.append({
        "id": h["id"],
        "name": h["name"],
        "img": h["img"],
        "attr": h["attr"],
        "type": h["type"],
        "roles": h["roles"],
        "picks": stat["picks"],
        "winrate": round(stat["winrate"] * 100, 1),
        "pickrate": round(stat["pickrate"] * 100, 2),
    })
table.sort(key=lambda x: x["winrate"], reverse=True)

# топ синергій
pairs = stats.get_hero_pairs(n=10, min_picks=50)
for p in pairs:
    p["name_a"] = hero_map.get(p["hero_a"], {}).get("name", str(p["hero_a"]))
    p["name_b"] = hero_map.get(p["hero_b"], {}).get("name", str(p["hero_b"]))
    p["img_a"] = hero_map.get(p["hero_a"], {}).get("img", "")
    p["img_b"] = hero_map.get(p["hero_b"], {}).get("img", "")
    p["winrate_pct"] = round(p["winrate"] * 100, 1)
```

```
    # контрпіки обраного героя
    counters = []
    counter_hero_name = ""
    if counter_hero:
        counter_hero_name = hero_map.get(counter_hero, {}).get("name",
str(counter_hero))
        for c in stats.get_counters(counter_hero, n=10, min_picks=50):
            counters.append({
                "name": hero_map.get(c["counter_id"], {}).get("name",
str(c["counter_id"])),
                "img": hero_map.get(c["counter_id"], {}).get("img", ""),
                "picks": c["picks"],
                "winrate_vs": round(c["winrate_vs"] * 100, 1),
            })

    return templates.TemplateResponse("heroes.html", {
        "request": request,
        "table": table,
        "pairs": pairs,
        "attr": attr,
        "role": role,
        "heroes": heroes_db,
        "counter_hero": counter_hero,
        "counter_hero_name": counter_hero_name,
        "counters": counters,
    })


@app.get("/predict", response_class=HTMLResponse)
def predict_page(request: Request):
    """Форма вибору складу команд."""
    return templates.TemplateResponse("predict.html", {
        "request": request,
        "heroes": get_heroes(),
        "result": None,
```

```
    })


@app.post("/predict", response_class=HTMLResponse)
def predict_result(
        request: Request,
        r1: int = Form(...), r2: int = Form(...), r3: int = Form(...),
        r4: int = Form(...), r5: int = Form(...),
        d1: int = Form(...), d2: int = Form(...), d3: int = Form(...),
        d4: int = Form(...), d5: int = Form(...),
):
    """Обробка форми предиктора: перевірка унікальності та виклик моделей."""
    heroes = get_heroes()
    hero_map = get_hero_map()

    radiant = [r1, r2, r3, r4, r5]
    dire = [d1, d2, d3, d4, d5]

    error = None
    if len(set(radiant + dire)) != 10:
        error = "Кожен герой може бути обраний тільки один раз"

    result = None
    if not error:
        predictions = ml.predict(radiant, dire)
        ml_results = ml.load_obj("results")
        result = {
            "radiant_heroes": [
                {"id": hid, "name": hero_map.get(hid, {}).get("name", str(hid)),
                 "img": hero_map.get(hid, {}).get("img", "")}
                for hid in radiant
            ],
            "dire_heroes": [
                {"id": hid, "name": hero_map.get(hid, {}).get("name", str(hid)),
                 "img": hero_map.get(hid, {}).get("img", "")}
```

```
                for hid in dire
            ],
            "predictions": [
                {
                    "model": name.upper(),
                    "prob": round(pred["radiant_win_prob"] * 100, 1),
                    "winner": pred["prediction"],
                    "accuracy": round(ml_results.get(name, {}).get("accuracy", 0) *
100, 1),
                    "roc_auc": round(ml_results.get(name, {}).get("roc_auc", 0), 4),
                }
                for name, pred in predictions.items()
            ],
        }

    return templates.TemplateResponse("predict.html", {
        "request": request,
        "heroes": heroes,
        "result": result,
        "error": error,
        "selected_radiant": radiant if not error else [],
        "selected_dire": dire if not error else [],
    })


@app.get("/predict/by-match", response_class=HTMLResponse)
def predict_by_match(request: Request, match_id: str = ""):
    """Завантаження складу команд за ідентифікатором завершеного матчу."""
    heroes = get_heroes()
    error = None
    selected_radiant = []
    selected_dire = []

    if match_id:
        if not match_id.strip().isdigit():
```

```
            error = "Match ID має містити тільки цифри"
        else:
            mid = int(match_id.strip())
            if mid < 1000:
                error = "Невалідний Match ID"
            else:
                try:
                    resp = requests.get(f"{OPENDOTA_API}/matches/{mid}", timeout=15)
                    if resp.status_code == 404:
                        error = "Матч не знайдено"
                    elif resp.status_code != 200:
                        error = f"Помилка API: {resp.status_code}"
                    else:
                        data = resp.json()
                        if "players" not in data:
                            error = "Дані матчу недоступні"
                        elif data.get("duration", 0) == 0:
                            error = "Матч ще не завершено"
                        else:
                            players = data["players"]
                            radiant = [p["hero_id"] for p in players if
p.get("isRadiant")]
                            dire = [p["hero_id"] for p in players if not
p.get("isRadiant")]
                            if len(radiant) != 5 or len(dire) != 5:
                                error = "Не вдалось отримати повні піки матчу"
                            else:
                                selected_radiant = radiant
                                selected_dire = dire
                except requests.exceptions.Timeout:
                    error = "Час очікування вичерпано - спробуй ще раз"
                except Exception:
                    error = "Помилка з'єднання з OpenDota API"

    return templates.TemplateResponse("predict.html", {
        "request": request,
```

```
        "heroes": heroes,
        "result": None,
        "error": error,
        "selected_radiant": selected_radiant,
        "selected_dire": selected_dire,
        "match_id": match_id,
    })
```

base.html:

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Dota 2 Analytics{% endblock %}</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link
href="https://fonts.googleapis.com/css2?family=JetBrains+Mono:wght@400;500;700&family
=Inter:wght@300;400;500&display=swap" rel="stylesheet">
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <style>
        *, *::before, *::after { box-sizing: border-box; margin: 0; padding: 0; }

        :root {
            --bg:      #0a0a0a;
            --surface: #111111;
            --border:  #222222;
            --text:    #e8e8e8;
            --muted:   #666666;
            --accent:  #c8a84b;
            --red:     #e05252;
            --green:   #52c078;
            --blue:    #5288e0;
        }
```

```
body {
    background: var(--bg);
    color: var(--text);
    font-family: 'Inter', sans-serif;
    font-size: 14px;
    line-height: 1.6;
    min-height: 100vh;
}

/* ── Навбар ── */
nav {
    border-bottom: 1px solid var(--border);
    padding: 0 40px;
    display: flex;
    align-items: center;
    gap: 40px;
    height: 56px;
    position: sticky;
    top: 0;
    background: var(--bg);
    z-index: 100;
}

.nav-logo {
    font-family: 'JetBrains Mono', monospace;
    font-size: 13px;
    font-weight: 700;
    color: var(--accent);
    text-decoration: none;
    letter-spacing: 0.1em;
    text-transform: uppercase;
}

.nav-links { display: flex; gap: 32px; margin-left: auto; }
```

```
.nav-links a {
    color: var(--muted);
    text-decoration: none;
    font-size: 13px;
    letter-spacing: 0.05em;
    transition: color 0.2s;
}

.nav-links a:hover,
.nav-links a.active { color: var(--text); }

/* ── Контейнер ── */
.container { max-width: 1200px; margin: 0 auto; padding: 48px 40px; }

/* ── Заголовки ── */
h1 {
    font-family: 'JetBrains Mono', monospace;
    font-size: 24px;
    font-weight: 700;
    letter-spacing: -0.02em;
    margin-bottom: 8px;
}

h2 {
    font-family: 'JetBrains Mono', monospace;
    font-size: 14px;
    font-weight: 500;
    color: var(--muted);
    text-transform: uppercase;
    letter-spacing: 0.1em;
    margin-bottom: 24px;
}

h3 {
    font-family: 'JetBrains Mono', monospace;
```

```
    font-size: 13px;
    font-weight: 500;
    margin-bottom: 16px;
}

/* ── Секція ── */
.section { margin-bottom: 64px; }

.section-header {
    display: flex;
    align-items: baseline;
    gap: 16px;
    margin-bottom: 24px;
    padding-bottom: 12px;
    border-bottom: 1px solid var(--border);
}

.section-header h1 { margin-bottom: 0; }

/* ── Картки ── */
.card {
    background: var(--surface);
    border: 1px solid var(--border);
    padding: 24px;
}

.grid-2 { display: grid; grid-template-columns: 1fr 1fr; gap: 16px; }
.grid-3 { display: grid; grid-template-columns: 1fr 1fr 1fr; gap: 16px; }
.grid-4 { display: grid; grid-template-columns: repeat(4, 1fr); gap: 16px; }

/* ── Статистична картка ── */
.stat-card {
    background: var(--surface);
    border: 1px solid var(--border);
    padding: 20px 24px;
```

```
}

.stat-label {
    font-size: 11px;
    color: var(--muted);
    text-transform: uppercase;
    letter-spacing: 0.1em;
    margin-bottom: 8px;
}

.stat-value {
    font-family: 'JetBrains Mono', monospace;
    font-size: 28px;
    font-weight: 700;
    color: var(--accent);
}

.stat-sub {
    font-size: 12px;
    color: var(--muted);
    margin-top: 4px;
}

/* ── Таблиця ── */
table { width: 100%; border-collapse: collapse; }

th {
    text-align: left;
    font-size: 11px;
    color: var(--muted);
    text-transform: uppercase;
    letter-spacing: 0.08em;
    padding: 8px 12px;
    border-bottom: 1px solid var(--border);
    font-weight: 500;
```

```
}

td {
    padding: 10px 12px;
    border-bottom: 1px solid var(--border);
    font-size: 13px;
}

tr:last-child td { border-bottom: none; }
tr:hover td { background: rgba(255,255,255,0.02); }

/* ── Герой рядок ── */
.hero-cell {
    display: flex;
    align-items: center;
    gap: 10px;
}

.hero-img {
    width: 32px;
    height: 18px;
    object-fit: cover;
    border: 1px solid var(--border);
}

/* ── Бейдж ── */
.badge {
    display: inline-block;
    font-size: 10px;
    padding: 2px 6px;
    border: 1px solid var(--border);
    color: var(--muted);
    font-family: 'JetBrains Mono', monospace;
}
```

```
.badge-str { border-color: #c05050; color: #c05050; }
.badge-agi { border-color: #50a850; color: #50a850; }
.badge-int { border-color: #5080c0; color: #5080c0; }
.badge-all { border-color: var(--accent); color: var(--accent); }

/* ── Прогрес бар ── */
.bar-wrap {
    display: flex;
    align-items: center;
    gap: 8px;
}

.bar {
    height: 3px;
    background: var(--border);
    flex: 1;
    max-width: 80px;
}

.bar-fill {
    height: 100%;
    background: var(--accent);
}

/* ── Кнопки ── */
.btn {
    display: inline-block;
    padding: 8px 20px;
    font-size: 13px;
    font-family: 'JetBrains Mono', monospace;
    cursor: pointer;
    border: 1px solid var(--border);
    background: transparent;
    color: var(--text);
    text-decoration: none;
```

```
    transition: all 0.15s;
    letter-spacing: 0.05em;
}

.btn:hover { border-color: var(--accent); color: var(--accent); }

.btn-primary {
    background: var(--accent);
    border-color: var(--accent);
    color: #000;
    font-weight: 700;
}

.btn-primary:hover {
    background: transparent;
    color: var(--accent);
}

/* ── Форма ── */
select {
    background: var(--surface);
    border: 1px solid var(--border);
    color: var(--text);
    padding: 8px 12px;
    font-size: 13px;
    font-family: 'Inter', sans-serif;
    width: 100%;
    cursor: pointer;
    appearance: none;
}

select:focus { outline: none; border-color: var(--accent); }

select option {
background: #111111;
```

```
        color: #e8e8e8;
        }

        label {
            display: block;
            font-size: 11px;
            color: var(--muted);
            text-transform: uppercase;
            letter-spacing: 0.08em;
            margin-bottom: 6px;
        }

        .form-group { margin-bottom: 16px; }

        /* ── Повідомлення про помилку ── */
        .error {
            border: 1px solid var(--red);
            color: var(--red);
            padding: 12px 16px;
            font-size: 13px;
            margin-bottom: 24px;
        }

        /* ── Монотекст ── */
        .mono {
            font-family: 'JetBrains Mono', monospace;
            font-size: 13px;
        }

        .text-green { color: var(--green); }
        .text-red   { color: var(--red); }
        .text-muted { color: var(--muted); }
        .text-accent { color: var(--accent); }
    </style>
    {% block head %}{% endblock %}
```

```
</head>
<body>
    <nav>
        <a href="/" class="nav-logo">D2 Analytics</a>
        <div class="nav-links">
            <a href="/" {% if request.url.path == "/" %}class="active"{% endif
%}>Головна</a>
            <a href="/heroes" {% if request.url.path == "/heroes" %}class="active"{%
endif %}>Герої</a>
            <a href="/predict" {% if request.url.path == "/predict"
%}class="active"{% endif %}>Предиктор</a>
        </div>
    </nav>

    <div class="container">
        {% block content %}{% endblock %}
    </div>
</body>
</html>
```

index.html:

```
{% extends "base.html" %}
{% block title %}Dota 2 Analytics - Головна{% endblock %}

{% block content %}

<!-- Заголовок -->
<div class="section">
    <div class="section-header">
        <h1>Аналітика матчів</h1>
        <span class="text-muted mono">Dota 2 · Патч 7.41</span>
    </div>

    <!-- Загальна статистика -->
    <div class="grid-4" style="margin-bottom: 48px;">
```

```
        <div class="stat-card">
            <div class="stat-label">Матчів у базі</div>
            <div class="stat-value">{{ "{:,}".format(match_count) }}</div>
            <div class="stat-sub">Ancient / Divine bracket</div>
        </div>
        <div class="stat-card">
            <div class="stat-label">Найкраща модель</div>
            <div class="stat-value mono" style="font-size:20px;">
                {{ best_model }}
            </div>
            <div class="stat-sub">Accuracy {{ "%.1f"|format(best_acc * 100) }}%</div>
        </div>
        <div class="stat-card">
            <div class="stat-label">Winrate Radiant</div>
            <div class="stat-value">{{ "%.1f"|format(radiant_winrate * 100) }}%</div>
            <div class="stat-sub">Перевага сторони карти</div>
        </div>
        <div class="stat-card">
            <div class="stat-label">Середня тривалість</div>
            <div class="stat-value">{{ avg_duration }} хв</div>
            <div class="stat-sub">All Pick · Ranked</div>
        </div>
    </div>

    <!-- Графіки -->
    <div class="grid-2" style="margin-bottom: 48px;">
        <div class="card">
            <h3>Розподіл тривалості матчів</h3>
            <canvas id="durationChart" height="200"></canvas>
        </div>
        <div class="card">
            <h3>Точність ML моделей</h3>
            <canvas id="modelsChart" height="200"></canvas>
        </div>
    </div>
```

```
    <!-- Топ герої -->
    <div class="grid-2">
        <div class="card">
            <h3>Топ за Winrate</h3>
            <table>
                <thead>
                    <tr>
                        <th>Герой</th>
                        <th>Winrate</th>
                        <th>Піки</th>
                    </tr>
                </thead>
                <tbody>
                    {% for h in top_winrate %}
                    <tr>
                        <td>
                            <div class="hero-cell">
                                <img class="hero-img" src="{{ h.img }}" alt="{{
h.name }}" onerror="this.style.display='none'">
                                {{ h.name }}
                            </div>
                        </td>
                        <td>
                            <div class="bar-wrap">
                                <span class="mono {% if h.winrate > 0.52 %}text-
green{% elif h.winrate < 0.48 %}text-red{% endif %}">
                                    {{ "%.1f"|format(h.winrate * 100) }}%
                                </span>
                                <div class="bar"><div class="bar-fill"
style="width:{{ [h.winrate * 100 * 1.5, 100]|min }}%"></div></div>
                            </div>
                        </td>
                        <td class="text-muted mono">{{ h.picks }}</td>
                    </tr>
                    {% endfor %}
```

```
                </tbody>
            </table>
        </div>

        <div class="card">
            <h3>Топ за Pickrate</h3>
            <table>
                <thead>
                    <tr>
                        <th>Герой</th>
                        <th>Pickrate</th>
                        <th>Піки</th>
                    </tr>
                </thead>
                <tbody>
                    {% for h in top_pickrate %}
                    <tr>
                        <td>
                            <div class="hero-cell">
                                <img class="hero-img" src="{{ h.img }}" alt="{{
h.name }}" onerror="this.style.display='none'">
                                {{ h.name }}
                            </div>
                        </td>
                        <td>
                            <div class="bar-wrap">
                                <span class="mono">{{ "%.1f"|format(h.pickrate * 100)
}}%</span>
                                <div class="bar"><div class="bar-fill"
style="width:{{ [h.pickrate * 100 * 3, 100]|min }}%"></div></div>
                            </div>
                        </td>
                        <td class="text-muted mono">{{ h.picks }}</td>
                    </tr>
                    {% endfor %}
                </tbody>
```

```
            </table>
        </div>
    </div>
</div>

<!-- Порівняння моделей -->
<div class="section">
    <div class="section-header">
        <h1>ML Моделі</h1>
    </div>
    <div class="card">
        <table>
            <thead>
                <tr>
                    <th>Модель</th>
                    <th>Accuracy</th>
                    <th>ROC-AUC</th>
                    <th>Опис</th>
                </tr>
            </thead>
            <tbody>
                {% set descriptions = {
                    "baseline": "Середній winrate команди",
                    "lr": "Логістична регресія",
                    "rf": "Random Forest (300 дерев)",
                    "xgb": "XGBoost (500 дерев)",
                    "mlp": "Нейронна мережа (128-64)"
                } %}
                {% for name in ["baseline", "lr", "rf", "xgb", "mlp"] %}
                {% if name in ml_results %}
                {% set r = ml_results[name] %}
                <tr>
                    <td>
                        <span class="mono">{{ name.upper() }}</span>
                        {% if r.accuracy == best_acc %}
```

```
                            <span class="badge text-accent" style="border-color:var(-
--accent);color:var(--accent);">best</span>
                        {% endif %}
                    </td>
                    <td class="mono {% if r.accuracy == best_acc %}text-accent{%
endif %}">
                        {{ "%.2f"|format(r.accuracy * 100) }}%
                    </td>
                    <td class="mono">{{ "%.4f"|format(r.roc_auc) }}</td>
                    <td class="text-muted">{{ descriptions.get(name, "") }}</td>
                </tr>
                {% endif %}
                {% endfor %}
            </tbody>
        </table>
    </div>
</div>

{% endblock %}

{% block head %}
<script>
document.addEventListener("DOMContentLoaded", () => {
    const accent = "#c8a84b";
    const border = "#222222";
    const muted  = "#666666";

    Chart.defaults.color = muted;
    Chart.defaults.borderColor = border;

    // Графік тривалості
    const dur = {{ duration | tojson }};
    new Chart(document.getElementById("durationChart"), {
        type: "bar",
        data: {
            labels: dur.map(d => d.label),
```

```
            datasets: [{
                data: dur.map(d => d.count),
                backgroundColor: accent + "33",
                borderColor: accent,
                borderWidth: 1,
            }]
        },
        options: {
            plugins: { legend: { display: false } },
            scales: {
                x: { grid: { color: border } },
                y: { grid: { color: border } }
            }
        }
    });

    // Графік моделей
    const models = {{ ml_results | tojson }};
    const names  = Object.keys(models).filter(k => k !== "baseline");
    new Chart(document.getElementById("modelsChart"), {
        type: "bar",
        data: {
            labels: names.map(n => n.toUpperCase()),
            datasets: [
                {
                    label: "Accuracy",
                    data: names.map(n => +(models[n].accuracy * 100).toFixed(2)),
                    backgroundColor: accent + "55",
                    borderColor: accent,
                    borderWidth: 1,
                },
                {
                    label: "ROC-AUC × 100",
                    data: names.map(n => +(models[n].roc_auc * 100).toFixed(2)),
                    backgroundColor: "#5288e055",
```

```
                    borderColor: "#5288e0",
                    borderWidth: 1,
                }
            ]
        },
        options: {
            plugins: { legend: { labels: { color: muted, font: { size: 11 } } } },
            scales: {
                x: { grid: { color: border } },
                y: { grid: { color: border }, min: 50, max: 62 }
            }
        }
    });
});
</script>
{% endblock %}
```

heroes.html:

```
{% extends "base.html" %}
{% block title %}Герої - Dota 2 Analytics{% endblock %}

{% block content %}

<div class="section">
    <div class="section-header">
        <h1>Статистика героїв</h1>
        <span class="text-muted mono">{{ table|length }} героїв</span>
    </div>

    <!-- Фільтри -->
    <form method="get" style="display:flex; gap:12px; margin-bottom:24px; align-
items:flex-end;">
        <div>
            <label>Атрибут</label>
```

```
            <select name="attr" style="width:140px;">
                <option value="">Всі</option>
                <option value="str" {% if attr == "str" %}selected{% endif
%}>Strength</option>
                <option value="agi" {% if attr == "agi" %}selected{% endif
%}>Agility</option>
                <option value="int" {% if attr == "int" %}selected{% endif
%}>Intelligence</option>
                <option value="all" {% if attr == "all" %}selected{% endif
%}>Universal</option>
            </select>
        </div>
        <div>
            <label>Роль</label>
            <select name="role" style="width:160px;">
                <option value="">Всі</option>
                {% for r in
["Carry","Support","Nuker","Disabler","Initiator","Durable","Escape","Pusher"] %}
                <option value="{{ r }}" {% if role == r %}selected{% endif %}>{{ r
}}</option>
                {% endfor %}
            </select>
        </div>
        <button type="submit" class="btn">Фільтрувати</button>
        {% if attr or role %}
        <a href="/heroes" class="btn">Скинути</a>
        {% endif %}
    </form>

    <!-- Таблиця героїв -->
    <div class="card" style="padding:0; margin-bottom:48px;">
        <table>
            <thead>
                <tr>
                    <th style="padding-left:24px;">Герой</th>
                    <th>Атрибут</th>
                    <th>Ролі</th>
```

```
                    <th>Winrate</th>
                    <th>Pickrate</th>
                    <th>Піків</th>
                </tr>
            </thead>
            <tbody>
                {% for h in table %}
                <tr>
                    <td style="padding-left:24px;">
                        <div class="hero-cell">
                            <img class="hero-img" src="{{ h.img }}" alt="{{ h.name
}}" onerror="this.style.display='none'">
                            <span>{{ h.name }}</span>
                        </div>
                    </td>
                    <td>
                        <span class="badge badge-{{ h.attr }}">{{ h.attr.upper()
}}</span>
                    </td>
                    <td>
                        <div style="display:flex; gap:4px; flex-wrap:wrap;">
                            {% for role in h.roles[:2] %}
                            <span class="badge">{{ role }}</span>
                            {% endfor %}
                        </div>
                    </td>
                    <td>
                        <div class="bar-wrap">
                            <span class="mono {% if h.winrate > 52 %}text-green{%
elif h.winrate < 48 %}text-red{% endif %}">
                                {{ h.winrate }}%
                            </span>
                            <div class="bar">
                                <div class="bar-fill" style="width:{{ [h.winrate *
1.5, 100]|min }}%;
                                    {% if h.winrate > 52 %}background:var(--green){%
elif h.winrate < 48 %}background:var(--red){% endif %}">
```

```
                            </div>
                        </div>
                    </div>
                </td>
                <td class="mono text-muted">{{ h.pickrate }}%</td>
                <td class="mono text-muted">{{ h.picks }}</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
</div>

<!-- Топ синергій -->
<div class="section-header">
    <h1>Топ синергій</h1>
    <span class="text-muted mono">мін. 50 спільних ігор</span>
</div>

<div class="card" style="padding:0; margin-bottom:48px;">
    <table>
        <thead>
            <tr>
                <th style="padding-left:24px;">Пара героїв</th>
                <th>Winrate разом</th>
                <th>Ігор</th>
            </tr>
        </thead>
        <tbody>
            {% for p in pairs %}
            <tr>
                <td style="padding-left:24px;">
                    <div style="display:flex; align-items:center; gap:8px;">
                        <img class="hero-img" src="{{ p.img_a }}" alt="{{
p.name_a }}" onerror="this.style.display='none'">
                        <span>{{ p.name_a }}</span>
```

```
                            <span class="text-muted">+</span>
                            <img class="hero-img" src="{{ p.img_b }}" alt="{{
p.name_b }}" onerror="this.style.display='none'">
                            <span>{{ p.name_b }}</span>
                        </div>
                    </td>
                    <td>
                        <div class="bar-wrap">
                            <span class="mono text-green">{{ p.winrate_pct }}%</span>
                            <div class="bar">
                                <div class="bar-fill" style="width:{{ [p.winrate_pct
* 1.5, 100]|min }}%; background:var(--green);"></div>
                            </div>
                        </div>
                    </td>
                    <td class="mono text-muted">{{ p.picks }}</td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>

    <!-- Контрпіки -->
    <div class="section-header" id="counters">
        <h1>Контрпіки</h1>
        <span class="text-muted mono">хто найчастіше перемагає обраного героя</span>
    </div>

    <form method="get" action="/heroes#counters" style="display:flex; gap:12px;
margin-bottom:24px; align-items:flex-end;">
        <div>
            <label>Герой</label>
            <select name="counter_hero" style="width:220px;">
                <option value="">- Оберіть героя -</option>
                {% for h in heroes %}
                <option value="{{ h.id }}" {% if counter_hero == h.id %}selected{%
endif %}>{{ h.name }}</option>
```

```
                {% endfor %}
            </select>
        </div>
        {% if attr %}<input type="hidden" name="attr" value="{{ attr }}">{% endif %}
        {% if role %}<input type="hidden" name="role" value="{{ role }}">{% endif %}
        <button type="submit" class="btn">Показати контрпіки</button>
    </form>

    {% if counters %}
    <div class="card" style="padding:0;">
        <table>
            <thead>
                <tr>
                    <th style="padding-left:24px;">Контрить {{ counter_hero_name
}}</th>
                    <th>Winrate проти</th>
                    <th>Ігор</th>
                </tr>
            </thead>
            <tbody>
                {% for c in counters %}
                <tr>
                    <td style="padding-left:24px;">
                        <div class="hero-cell">
                            <img class="hero-img" src="{{ c.img }}" alt="{{ c.name
}}" onerror="this.style.display='none'">
                            <span>{{ c.name }}</span>
                        </div>
                    </td>
                    <td>
                        <div class="bar-wrap">
                            <span class="mono text-green">{{ c.winrate_vs }}%</span>
                            <div class="bar">
                                <div class="bar-fill" style="width:{{ [c.winrate_vs *
1.5, 100]|min }}%; background:var(--green);"></div>
                            </div>
```

```
                        </div>
                    </td>
                    <td class="mono text-muted">{{ c.picks }}</td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
    {% elif counter_hero %}
    <div class="text-muted mono" style="padding:12px 0;">Недостатньо даних для цього героя (поріг 50 ігор).</div>
    {% endif %}
</div>

{% endblock %}
```

predict.html:

```
{% extends "base.html" %}
{% block title %}Предиктор - Dota 2 Analytics{% endblock %}

{% block head %}
<style>
    .teams-grid {
        display: grid;
        grid-template-columns: 1fr 1fr;
        gap: 24px;
        margin-bottom: 24px;
    }

    .team-label {
        font-family: 'JetBrains Mono', monospace;
        font-size: 11px;
        text-transform: uppercase;
        letter-spacing: 0.1em;
```

```
    margin-bottom: 12px;
}

.team-radiant { color: var(--green); }
.team-dire    { color: var(--red); }

.pick-slot {
    border: 1px solid var(--border);
    background: var(--surface);
    padding: 8px 12px;
    margin-bottom: 8px;
    display: flex;
    align-items: center;
    gap: 10px;
}

.pick-slot select {
    flex: 1;
    background: transparent;
    border: none;
    padding: 0;
    font-size: 13px;
}

.pick-slot select:focus { outline: none; }

.pick-num {
    font-family: 'JetBrains Mono', monospace;
    font-size: 11px;
    color: var(--muted);
    width: 16px;
    flex-shrink: 0;
}

/* Результати */
```

```
.result-section { margin-top: 48px; }

.result-teams {
    display: grid;
    grid-template-columns: 1fr auto 1fr;
    gap: 24px;
    align-items: center;
    margin-bottom: 32px;
}

.result-team-heroes {
    display: flex;
    flex-direction: column;
    gap: 6px;
}

.result-hero {
    display: flex;
    align-items: center;
    gap: 8px;
    font-size: 13px;
}

.result-hero img {
    width: 40px;
    height: 23px;
    object-fit: cover;
    border: 1px solid var(--border);
}

.vs-label {
    font-family: 'JetBrains Mono', monospace;
    font-size: 20px;
    color: var(--muted);
    text-align: center;
```

```
}

.model-row {
    display: grid;
    grid-template-columns: 100px 1fr 80px 100px 100px;
    align-items: center;
    gap: 16px;
    padding: 14px 0;
    border-bottom: 1px solid var(--border);
}

.model-row:last-child { border-bottom: none; }

.model-name {
    font-family: 'JetBrains Mono', monospace;
    font-size: 13px;
    font-weight: 500;
}

.prob-bar-wrap { position: relative; height: 4px; background: var(--border); }

.prob-bar-r {
    position: absolute;
    left: 50%;
    height: 100%;
    background: var(--green);
}

.prob-bar-d {
    position: absolute;
    right: 50%;
    height: 100%;
    background: var(--red);
}
```

```
    .prob-center {
        position: absolute;
        left: 50%;
        top: -3px;
        width: 1px;
        height: 10px;
        background: var(--muted);
    }

    .winner-badge {
        font-family: 'JetBrains Mono', monospace;
        font-size: 11px;
        padding: 3px 8px;
        border: 1px solid;
        text-align: center;
    }

    .winner-radiant { color: var(--green); border-color: var(--green); }
    .winner-dire    { color: var(--red);   border-color: var(--red); }
</style>
{% endblock %}

{% block content %}

<div class="section">
    <div class="section-header">
        <h1>Предиктор</h1>
        <span class="text-muted mono">Прогноз по піку</span>
    </div>

    {% if error %}
    <div class="error">{{ error }}</div>
    {% endif %}

    <!-- Пошук по match_id -->
```

```
    <form method="get" action="/predict/by-match" style="display:flex; gap:12px;
margin-bottom:32px; align-items:flex-end;">
        <div>
            <label>Match ID</label>
            <input
                type="number"
                name="match_id"
                placeholder="Введіть ID матчу"
                value="{{ match_id or '' }}"
                style="background:var(--surface); border:1px solid var(--border);
color:var(--text);
                       padding:8px 12px; font-size:13px; font-family:'Inter',sans-
serif; width:220px;"
            >
        </div>
        <button type="submit" class="btn">Завантажити піки</button>
    </form>

    <!-- Основна форма вибору героїв -->
    <form method="post" action="/predict">
        <div class="teams-grid">
            <!-- Radiant -->
            <div>
                <div class="team-label team-radiant">● Radiant</div>
                {% for i in range(1, 6) %}
                <div class="pick-slot">
                    <span class="pick-num">{{ i }}</span>
                    <select name="r{{ i }}" required>
                        <option value="">- Оберіть героя -</option>
                        {% for h in heroes %}
                        <option value="{{ h.id }}"
                            {% if selected_radiant and selected_radiant[i-1] == h.id
%}selected{% endif %}>
                            {{ h.name }}
                        </option>
                        {% endfor %}
                    </select>
```

```
                </div>
                {% endfor %}
            </div>

            <!-- Dire -->
            <div>
                <div class="team-label team-dire">● Dire</div>
                {% for i in range(1, 6) %}
                <div class="pick-slot">
                    <span class="pick-num">{{ i }}</span>
                    <select name="d{{ i }}" required>
                        <option value="">- Оберіть героя -</option>
                        {% for h in heroes %}
                        <option value="{{ h.id }}"
                            {% if selected_dire and selected_dire[i-1] == h.id
%}selected{% endif %}>
                            {{ h.name }}
                        </option>
                        {% endfor %}
                    </select>
                </div>
                {% endfor %}
            </div>
        </div>

        <button type="submit" class="btn btn-primary">Передбачити результат</button>
    </form>

    <!-- Результати -->
    {% if result %}
    <div class="result-section">
        <div class="section-header">
            <h1>Результат</h1>
        </div>
```

```
        <!-- Герої -->
        <div class="card" style="margin-bottom:24px;">
            <div class="result-teams">
                <div>
                    <div class="team-label team-radiant" style="margin-
bottom:12px;">Radiant</div>
                    <div class="result-team-heroes">
                        {% for h in result.radiant_heroes %}
                        <div class="result-hero">
                            <img src="{{ h.img }}" alt="{{ h.name }}"
onerror="this.style.display='none'">
                            {{ h.name }}
                        </div>
                        {% endfor %}
                    </div>
                </div>

                <div class="vs-label">VS</div>

                <div>
                    <div class="team-label team-dire" style="margin-
bottom:12px;">Dire</div>
                    <div class="result-team-heroes">
                        {% for h in result.dire_heroes %}
                        <div class="result-hero">
                            <img src="{{ h.img }}" alt="{{ h.name }}"
onerror="this.style.display='none'">
                            {{ h.name }}
                        </div>
                        {% endfor %}
                    </div>
                </div>
            </div>
        </div>

        <!-- Прогнози моделей -->
        <div class="card" style="padding:0 24px;">
```

```
            <div class="model-row" style="border-bottom:1px solid var(--border);">
                <span style="font-size:11px;color:var(--muted);text-
transform:uppercase;letter-spacing:.08em;">Модель</span>
                <div style="display:grid;grid-template-columns:1fr 1fr 1fr;font-
size:11px;color:var(--muted);text-transform:uppercase;letter-spacing:.08em;gap:8px;">
                    <span>Radiant</span>
                    <span style="text-align:center;"></span>
                    <span style="text-align:right;">Dire</span>
                </div>
                <span style="font-size:11px;color:var(--muted);text-
transform:uppercase;letter-spacing:.08em;">Переможець</span>
                <span style="font-size:11px;color:var(--muted);text-
transform:uppercase;letter-spacing:.08em;">Accuracy</span>
                <span style="font-size:11px;color:var(--muted);text-
transform:uppercase;letter-spacing:.08em;">ROC-AUC</span>
            </div>

            {% for p in result.predictions %}
            <div class="model-row">
                <span class="model-name">{{ p.model }}</span>

                <div>
                    <div style="display:flex;justify-content:space-between;margin-
bottom:4px;">
                        <span class="mono text-green" style="font-size:12px;">{{
p.prob }}%</span>
                        <span class="mono text-red"   style="font-size:12px;">{{
"%.1f"|format(100 - p.prob) }}%</span>
                    </div>
                    <div class="prob-bar-wrap">
                        <div class="prob-bar-r" style="width:{{ [p.prob - 50, 0]|max
}}%;"></div>
                        <div class="prob-bar-d" style="width:{{ [50 - p.prob, 0]|max
}}%;"></div>
                        <div class="prob-center"></div>
                    </div>
                </div>
```

```
                <div class="winner-badge {% if p.winner == 'Radiant' %}winner-
radiant{% else %}winner-dire{% endif %}">
                    {{ p.winner }}
                </div>

                <span class="mono text-muted">{{ p.accuracy }}%</span>
                <span class="mono text-muted">{{ p.roc_auc }}</span>
            </div>
            {% endfor %}
        </div>
    </div>
    {% endif %}
</div>

{% endblock %}
```