МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Чорноморський національний університет імені Петра Могили**

**Факультет комп'ютерних наук**

**Кафедра інженерії програмного забезпечення**

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2025 р.

# КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

на тему:

# «Інформаційна система підтримки вибору та оцінювання методів узгодження поведінки великих мовних моделей»

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмного забезпечення»

**Здобувач** _____ **Кіріл ТРУШЕВСЬКИЙ**

«__» _____ 2025 р.

**Керівник роботи** **Гліб ГОРБАНЬ**

канд. техн. наук,
доцент _____

«__» _____ 2025 р.

Миколаїв – 2025

**Завдання на виконання кваліфікаційної роботи**

Чорноморський національний університет імені Петра Могили

| | |
|---|---|
| Факультет | Комп'ютерних наук |
| Кафедра | Інженерії програмного забезпечення |
| Рівень вищої освіти | Другий (магістерський) |
| Освітній ступінь | Магістр |
| Спеціальність | 121 Інженерія програмного забезпечення |
| Освітня програма | Інженерія програмного забезпечення |

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«____» _____ 2025 р.

**ЗАВДАННЯ**
**на кваліфікаційну магістерську роботу здобувача вищої освіти**
**Трушевського Кіріла**

1. Тема кваліфікаційної роботи «Інформаційна система підтримки вибору та оцінювання методів узгодження поведінки великих мовних моделей» затверджена наказом ректора ЧНУ ім. Петра Могили №182 від «02» липня 2025 р.

2. Строк представлення кваліфікаційної роботи «___» _____ 2025 р.

3. Очікуваний результат роботи та початкові дані якщо такі потрібні.

4. Перелік питань, що підлягають розробці:

— аналіз предметної області автоматичного створення протоколів;`

— дослідження існуючих методів розпізнавання мовлення та генерації текстів;

— розробка моделі процесу обробки аудіо та формування структурованого протоколу;

— побудова архітектурної моделі програмного забезпечення;

— формування специфікації функціональних і нефункціональних вимог;

— реалізація прототипу системи;

— оцінка працездатність розробленого рішення.

5. Перелік графічних матеріалів: презентація

6. Консультанти:

| **Консультант** | **Кафедра (організація)** | **Частина роботи** |
|---|---|---|
| | | |
| | | |
| | | |

Дата видачі завдання «____» _____ 2025 р.

**КАЛЕНДАРНИЙ ПЛАН**
**виконання кваліфікаційної роботи**

Тема: **Інформаційна система підтримки вибору та оцінювання методів узгодження поведінки великих мовних моделей**

| № | Найменування роботи | Початок | Закінчення | Примітки |
|---|---|---|---|---|
| | Розробка та затвердження завдання на виконання КМР | 01.09.2025 | 08.09.2025 | Виконано |
| | Огляд літератури за темою роботи | 08.09.2025 | 19.09.2025 | Виконано |
| | Складання календарного плану КМР | 08.09.2025 | 10.09.2025 | Виконано |
| | Аналіз предметної області | 10.09.2025 | 26.09.2025 | Виконано |
| | Розробка проєктних рішень | 29.09.2025 | 17.10.2025 | Виконано |
| | Моделювання та конструювання ПЗ | 21.10.2025 | 03.11.2025 | Виконано |
| | Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача | 05.11.2025 | 24.11.2025 | Виконано |
| | Попередній захист | 24.11.2025 | 24.11.2025 | Виконано |
| | Відгук керівника КМР | | | Виконано |
| | Оформлення КМР та презентації | | | Виконано |
| | Рецензування | | | Виконано |
| | Завершення оформлення КМР та презентації | | | Виконано |
| | Захист кваліфікаційної роботи | | | Виконано |

**Здобувач** _____ **Кіріл ТРУШЕВСЬКИЙ**

«__» _____ 2025 р.

**Керівник роботи**

канд. техн. наук,

доцент _____ **Гліб ГОРБАНЬ**

«__» _____ 2025 р.

# АНОТАЦІЯ

до кваліфікаційної магістерської роботи

«Інформаційна система підтримки вибору та оцінювання методів узгодження поведінки великих мовних моделей»

Здобувач 608 гр.: Трушевський Кіріл

Керівник: канд. техн. наук, доцент Горбань Гліб

Магістерська робота присвячена дослідженню та розробленню методів оцінювання й вибору технологій узгодження поведінки великих мовних моделей (LLM) у контексті побудови інформаційної системи для керованого, відтворюваного та обґрунтованого процесу прийняття рішень. Актуальність дослідження зумовлена зростанням використання великих мовних моделей у промислових та дослідницьких застосуваннях, що супроводжується ризиками токсичності, упередженості, порушення форматів відповідей та нестабільності поведінки моделей. Такі виклики потребують створення уніфікованих інструментів, які забезпечують прозоре оцінювання та коректний вибір методів узгодження.

Об’єктом дослідження є процес узгодження поведінки великих мовних моделей.

Предметом дослідження є методи оцінювання, порівняння та багатокритеріального вибору технологій узгодження великих мовних моделей, а також моделі і програмні засоби, що забезпечують автоматизацію цього процесу.

Мета роботи – підвищення обґрунтованості та ефективності вибору методів узгодження поведінки великих мовних моделей шляхом розроблення інформаційної системи з багатокритеріальним аналізом рішень та стандартизованими процедурами оцінювання.

Відповідно до мети визначено такі завдання:

– виконати системний огляд підходів до узгодження та типів даних, що для них застосовуються;

– сформувати критерії та метрики оцінювання якості, безпеки, ефективності, надійності та вартості;

– побудувати концептуальну та інформаційну модель системи, визначити життєвий цикл експериментів і артефактів;

– розробити модель багатокритеріального вибору та методику оцінювання з чутливісним аналізом;

– підготувати специфікацію вимог до програмного забезпечення;

– спроєктувати архітектуру та інтерфейси системи, включно з моделями даних і користувацьким інтерфейсом;

– реалізувати прототип, провести тестування та експериментальне порівняння методів, сформувати звіти.

Кваліфікаційна магістерська робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

У вступі визначено актуальність теми, сформульовано мету, завдання, об’єкт і предмет дослідження.

У першому розділі проведено аналіз предметної області, сучасних методів узгодження поведінки великих мовних моделей та підходів до оцінювання їх ефективності й безпеки.

Другий розділ присвячено розробці формальної математичної моделі багатокритеріального вибору (MCDA), концептуальної та інформаційної моделей системи, а також специфікації функціональних і нефункціональних вимог.

У третьому розділі розроблено архітектуру програмного забезпечення, реалізовано прототип системи, описано модулі, API та процеси оцінювання, а також проведено тестування прототипу.

У висновках узагальнено результати роботи, визначено практичне значення та перспективи подальшого розвитку системи.

Кваліфікаційна робота викладена на 126 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 45 найменувань та 0 додатків. Праця містить 6 таблиць та 7 рисунків.

Ключові слова: великі мовні моделі, узгодження поведінки, оцінювання моделей, метрики якості, MCDA, інформаційна система, штучний інтелект.

# ABSTRACT

to the qualifying master's thesis

«Information system to support the selection and evaluation of methods for matching the behavior of large language models»

Student of 608 group: Trushevskyi Kiril

Supervisor: Candidate of Technical Sciences (Ph. D.), Associate Professor Horban Hlib

The master’s thesis is devoted to the study and development of methods for evaluating and selecting technologies for aligning the behavior of large language models (LLMs) in the context of building an information system that supports controlled, reproducible, and well–grounded decision–making. The relevance of the research is determined by the growing use of large language models in industrial and scientific applications, which is accompanied by risks such as toxicity, bias, response format inconsistencies, and instability of model behavior. These challenges necessitate the development of unified tools that ensure transparent evaluation and correct selection of оцінювання methods.

The object of the research is the process of aligning the behavior of large language models. The subject of the research includes methods for evaluating, comparing, and performing multi–criteria selection of LLM оцінювання technologies, as well as the models and software tools that enable automation of this process.

The aim of the work is to increase the justification and effectiveness of selecting methods for aligning the behavior of large language models by developing an information system with multicriteria decision analysis and standardized evaluation procedures.

In accordance with the aim, the following tasks are defined:

– conduct a systematic review of alignment approaches and the types of data used for them;

– define criteria and metrics for evaluating quality, safety, efficiency, reliability, and cost;

– build a conceptual and information model of the system and define the life cycle of experiments and artifacts;

– develop a multicriteria selection model and an evaluation methodology with sensitivity analysis;

– prepare a software requirements specification;

– design the system architecture and interfaces, including data models and the user interface;

– implement a prototype, perform testing, and carry out an experimental comparison of methods, and generate reports.

The master's qualification work consists of an introduction, four chapters, conclusions, a list of references, and appendices.

The introduction defines the relevance of the topic and formulates the aim, objectives, object, and subject of the research.

The first chapter presents an analysis of the domain, modern methods of LLM behavior оцінювання, and approaches for evaluating their efficiency and safety.

The second chapter is devoted to the development of a formal mathematical model for multi–criteria decision analysis (MCDA), the conceptual and information models of the system, and the specification of functional and non–functional requirements.

The third chapter presents the architecture of the software solution, the implementation of the system prototype, descriptions of modules, APIs, and evaluation processes, as well as testing of the prototype.

The conclusions summarize the results of the work, define its practical significance, and outline prospects for further development of the system.

The qualification work consists of 126 pages of printed text and includes an introduction, 4 chapters, general conclusions, a list of 45 references, and 0 appendices. The work contains 6 tables and 7 figures.

Keywords: large language models, model evaluation, quality metrics, MCDA, information system, artificial intelligence.

## ПЕРЕЛІК СКОРОЧЕНЬ

AI – Artificial Intelligence
AMI – Augmented Multiparty Interaction
API – Application Programming Interface
DNN – Deep Neural Network
GMM – Gaussian Mixture Model
GPT – Generative Pre–trained Transformer
GRU – Gated Recurrent Unit
HCI – Human–Computer Interaction
HMM – Hidden Markov Model
ICSI – International Computer Science Institute
LLM – Large Language Model
LSTM – Long Short–Term Memory
MFCC – Mel–Frequency Cepstral Coefficients
NER – Named Entity Recognition
NLP – Natural Language Processing
RNN – Recurrent Neural Network
SaaS – Software as a Service

# ЗМІСТ

# ВСТУП

Актуальність теми полягає насамперед в тому, що стрімке впровадження великих мовних моделей у продукти й сервіси вимагає ефективного та швидкого узгодження їхньої поведінки з вимогами безпеки, якості та політик. Науково–практичне значення роботи полягає у створенні інформаційної системи, яка забезпечує відтворюваний, прозорий та економний процес вибору і оцінювання методів узгодження, зменшує витрати на експерименти та підвищує якість прийняття рішень.

Необхідність нової розробки обумовлена тим, що наявні інструменти зазвичай покривають окремі етапи (навчання, бенчмарки, логування), але бракує інтегрованого рішення, яке поєднує каталог методів, уніфіковані протоколи оцінювання та багатокритеріальний вибір з урахуванням вартості, латентності, безпеки й обмежень даних, – детальніший аналіз таких прогалин наведено у розділі 1. Основні проєктні рішення включають клієнт–серверну архітектуру з виокремленими компонентами (ядро прийняття рішень, модуль оцінювань, реєстр експериментів, звітність), застосування багатокритеріального аналізу для ранжування альтернатив, відтворювані протоколи оцінювання та аудит артефактів.

Сфера застосування результатів охоплює дослідницькі групи та інженерні команди для підтримки вибору методів узгодження, освітні курси для стандартизації оцінювань, а також промислові пілоти, де потрібні контроль витрат і прозорість рішень. Апробація результатів планується насампред в формі представлення матеріалів на профільних семінарах/конференціях. інформацію про фактичну апробацію при її проведенні також буде наведено.

Структура роботи: у розділі 1 подано аналітичний огляд підходів, даних і метрик та сформульовано постановку задачі;у розділі 2 описано моделі системи, методику багатокритеріального вибору та специфікацію вимог до програмного забезпечення;у розділі 3 наведено архітектуру, моделі й проєктні рішення щодо даних та інтерфейсів;у розділі 4 подано реалізацію прототипу, результати

тестування та коротке керівництво користувача. Висновки узагальнюють досягнуті результати, перелік джерел і додатки завершують роботу.

# 1 АНАЛІЗ СУЧАСНОГО ВИСКОРИСТАННЯ МЕТОДІВ УЗГОДЖЕННЯ ПОВЕДІНКИ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

## 1.1. Об’єкт і предмет та стан проблеми

Попит на рішення на основі ШІ стрімко зростає у світі – від державних послуг до щоденних бізнес–процесів. Великі мовні моделі масово вбудовують у сервіси, але це несе ризики хибних відповідей, упередженості та небезпечного контенту, а також додає регуляторні вимоги [41, 8] .

Тому зростає потреба в пошуку зрозумілих і відтворюваних підходів до вибору та оцінювання методів узгодження поведінки моделей із чітко окресленими об’єктом, предметом, межами та припущеннями [21, 14].

Для чіткого окреслення предметної області розглянемо більш детально визначення об’єкта й предмета дослідження разом із межами та припущеннями.

Як було визначено у вступі, об’єкт дослідження – процес узгодження поведінки великих мовних моделей у прикладних системах, включно з прийняттям рішень щодо вибору методу узгодження, організацією оцінювання та інтерпретацією результатів у рамках заданих вимог і обмежень.

Предмет дослідження – методи, дані, метрики та програмні засоби, які забезпечують підтримку вибору і оцінювання підходів до узгодження поведінки моделей: класи методів (на основі преференцій, інструкційне налаштування, конституційні/політичні підходи тощо), критерії та процедури оцінювання, а також архітектурні та інформаційні рішення інформаційної системи підтримки рішень.

У кваліфікаційній магістерській роботі розглядаються лише великі мовні моделі для тексту (LLM) – тобто системи, що генерують відповіді послідовно, слово за словом. Мультимодальні моделі (які працюють із текстом разом із зображеннями, аудіо тощо) та вузькоспеціалізовані агенти сюди не входять, щоб звузити фокус і зробити результати однорідними.

Магістерська робота фокусується на прикладних способах узгодження поведінки. Це, по–перше, інструкційне навчання й тонке налаштування (коли

модель навчають виконувати інструкції користувача на прикладах), по–друге, підходи «на основі уподобань» (коли вибір кращих відповідей роблять люди або проксі–процедури, і модель вчиться на цих оцінках), і, по–третє, правила й фільтри під час виконання моделі («guardrails») – тобто перевірки та обмеження, що відсікають небажаний вміст уже на етапі відповіді. Нова базова архітектура моделі не створюється.

Оцінювання проводиться офлайн на заздалегідь підготовлених наборах завдань (датасетах). Під «датасетом» мається на увазі структурований набір запитань і очікуваних відповідей або правил перевірки, у тому числі завдання на безпеку (відсутність шкідливих порад), чесність/упередженість і небажаний вміст. Онлайнові A/B–тести (порівняння двох варіантів на реальних користувачах у працюючому середовищі не проводяться.

Джерела даних – публічні або доступні для навчального використання датасети, а також контрольовані внутрішні набори. Масовий новий збір людських розміток (коли багато людей вручну оцінюють відповіді) не планується.

Метрики (тобто числові показники як «точність», «частка токсичних відповідей», «час відповіді», «орієнтовна вартість») беруться зі стандартних бенчмарків – відкритих тестових наборів і узгоджених правил підрахунку. Автоматичне оцінювання «моделлю–суддею» (коли інша модель виставляє оцінку відповіді) буде застосовуватись лише як допоміжний інструмент і завжди зазначатись його обмеження та можливі похибки [36, 26, 44].

Програмна система робиться під лабораторне чи пілотне використання: є модуль підтримки вибору за кількома критеріями (з використаням MCDA – багатокритеріального аналізу рішень), модуль оцінювань (який запускає тести й збирає метрики), реєстр експериментів (облік запусків і результатів) та звітність. Теми масштабування до дуже великих навантажень торкаємося лише на рівні загальних підходів; промислові цільові показники надійності й швидкодії тут не встановлюються.

У межах дослідження не створюється новий алгоритм узгодження, не робиться формальна перевірка безпеки моделей і не проводиться юридична експертизу політик. Також не будуємо повний промисловий контур MLOps (від ML + DevOps – тобто конвеєр «дані –> навчання –> розгортання –> моніторинг –> оновлення»): не реалізується автоматичне масштабування, поступове розгортання на невеликій частині трафіку з можливістю швидкого відкату (canary/blue–green) та спеціальні низькорівневі оптимізації під «залізо».

Виходимо з того, що є доступні базові обчислювальні ресурси (GPU/CPU) або API моделей, а ліміти вартості та квот зафіксовані в налаштуваннях запусків. Експерименти мають бути налаштовані так, щоб їх можна було повторити: фіксуємо версії датасетів і конфігурацій, контролюємо випадковість, зберігаємо журнали та проміжні файли. Вибір «найкращого» методу робимо за кількома критеріями з наперед узгодженими вагами, при цьому окремо перевіряємо, як результат змінюється за інших ваг [4]. Розуміючи, що підсумки залежать від обраних бенчмарків і політик доступу до моделей, і можливі зсуви оцінок при використанні автоматизованих «суддів»–моделей – компенсуємо додатковими ручними перевірками вибірки.

Роботу розраховано на навчальні та дослідницькі сценарії, а також пілотні промислові впровадження, де потрібно прозоро порівняти підходи до узгодження і обрати метод з урахуванням якості, безпеки, вартості та обмежень середовища.

Перейшовши від окреслення меж та припущень, логічно пояснити, яку саме проблему розв’язуємо і чому це важливо.

Сьогодні існує багато способів «узгоджувати» поведінку великих мовних моделей: навчання на інструкціях, коригування за людськими уподобаннями, а також правила й фільтри під час відповіді моделі. На практиці вибір між цими підходами часто робиться «на око», під конкретний проєкт і під наявні ресурси, без єдиного порядку порівняння. Через це результати різних команд важко зіставити: використовуються різні набори завдань (датасетів), різні метрики

якості та безпеки, різні умови запуску. Підсумки виходять «непорівнюваними», а прийняті рішення – слабко обґрунтованими [19].

Проблему підсилюють практичні обмеження: обчислювальна вартість, швидкість відповіді, вимоги до безпеки й відповідності політикам. Єдиної «універсальної» метрики немає: підхід, що показує високу якість, може бути повільним або дорогим;той, що добре блокує небажаний вміст, інколи зменшує корисність відповідей. Без прозорої процедури порівняння команди витрачають більше часу та коштів, а вибір часто залежить від суб’єктивних уподобань, а не від даних.

Отже, виникає потреба у зрозумілій і відтворюваній системі підтримки вибору: вона має зібрати в одному місці опис підходів, дозволити запускати стандартизовані оцінювання на спільних датасетах, автоматично збирати метрики (якість, безпека, швидкодія, орієнтовні витрати) та допомагати обирати метод з урахуванням кількох критеріїв одночасно. Така система зменшує «ручні» помилки, пришвидшує експерименти, робить звіти порівнюваними й придатними для аудиту, а отже – підвищує обґрунтованість інженерних рішень [37].

Першими користувачами є дослідники й ML–інженери. Їм потрібно швидко запускати порівняння підходів до узгодження, працювати з єдиними наборами завдань, бачити зрозумілі метрики якості, безпеки та швидкодії, а головне – мати відтворювані експерименти: щоб будь–який запуск можна було повторити з тими самими налаштуваннями і отримати близькі результати.

Продакт–менеджери та власники продукту очікують прозорого вибору «що беремо у робочу версію»: їм потрібні короткі звіти з порівняннями, де видно компроміс між якістю, часом відповіді та витратами. Вони хочуть приймати рішення на підставі даних, а не інтуїції, і мати історію таких рішень для аудиту.

Фахівці з безпеки, етики та відповідності зосереджені на відповідності політикам: їм потрібні сценарії перевірок на шкідливий чи упереджений контент, чіткі протоколи тестування, журнали подій і зрозумілі звіти про ризики.

Важливо, щоб ці перевірки були повторюваними і фіксували, за яких умов отримано результат.

Команди інфраструктури/DevOps потребують простих способів інтегрувати систему у наявне середовище: зрозумілі конфігурації, можливість запускати обчислення в черзі, базовий моніторинг станів, імпорт/експорт результатів та API для автоматизації.

Анотаторам і перевіряльникам якості (якщо вони залучені) потрібні легкі інтерфейси для перегляду відповідей, ручної оцінки та швидкого формування вибіркових перевірок, щоб доповнювати автоматичні метрики людською оцінкою там, де це критично .

Нарешті, тестувальники й представники бізнес–підрозділів очікують, що система дозволятиме проганяти типові прикладні сценарії, порівнювати альтернативи «бік–о–бік» і отримувати зрозумілі підсумкові звіти (таблиці, графіки, короткі висновки), які можна додати до внутрішньої документації або презентацій [23, 40, 26]

Головні ризики та обмеження пов’язані з даними й оцінюванням. Дані можуть бути неповними або нерепрезентативними, що породжує упередженість (bias) – систематичні перекоси у відповідях моделі щодо певних тем чи груп. Є ризики безпеки: модель іноді генерує небажаний чи шкідливий контент, водночас жорсткі фільтри можуть занадто «перекривати» корисні відповіді. На результати також впливає випадковість роботи моделей (один і той самий запит інколи дає різні відповіді), тому важливо фіксувати умови запуску. Автоматичні перевірки якості (коли інша модель оцінює відповіді) зручні, але можуть помилятися, тож потрібні вибіркові ручні перевірки. Додаються практичні обмеження – обчислювальні ресурси, час виконання, умови використання API постачальників моделей і ліцензії на дані. Нарешті, існує ризик «пристосування під тест»: метод добре виглядає на конкретному бенчмарку, але гірше – у реальних завданнях [17, 38].

Обмеження нашого домену такі: працюємо з текстовими моделями загального призначення, оцінювання проводимо офлайн на підготовлених

наборах завдань (без довгих онлайн A/B–експериментів), використовуємо публічні або дозволені для навчальних цілей набори даних і не збираємо масові нові розмітки. Розглядаємо прикладні способи узгодження (тонке налаштування на інструкціях, підходи на основі оцінок відповідей, правила та фільтри під час відповіді моделі) і не розробляємо нову базову архітектуру моделі. Результати інтерпретуємо з урахуванням ресурсних і правових обмежень.

Відповідно можна сформулювати наступні дослідницькі запитання.

– визначити, який підхід до узгодження забезпечує найкращий баланс між якістю відповіді та безпекою за заданих умов;

– оцінити, як змінюються підсумки за використання різних наборів завдань і різних метрик оцінювання;

– окреслити прийнятні на практиці компроміси між якістю, швидкістю та орієнтовними витратами;

– перевірити, що проста й прозора процедура багатокритеріального вибору стабільно рекомендує метод у різних сценаріях використання;

– визначити мінімальні вимоги до даних та інфраструктури, необхідні для відтворення результатів в іншій організації.

### 1.2. Підходи до узгодження поведінки та дані

Коротко окреслимо основні класи методів узгодження.

Узгодження поведінки моделей зазвичай роблять кількома базовими способами. Перший – інструкційне навчання: модель донавчають на парах «запит –> бажана відповідь», щоб краще виконувала інструкції користувача. Другий – методи на основі уподобань: для одного запиту беруть дві відповіді, люди обирають кращу, і модель підганяють під ці вибори (через «модель винагороди» або прямо за позначками «краще/гірше»);це дає відповіді, ближчі до очікувань, але потребує людських оцінок і чітких інструкцій для оцінювачів [20].

Щоб зменшити вартість ручних оцінок, інколи частину роботи виконує інша модель–«суддя»: вона порівнює відповіді та виставляє оцінки;це швидше, але потребує вибіркових людських перевірок, бо «суддя» може помилятися. Окремий клас – правилові або «конституційні» підходи: заздалегідь задають набір принципів і заборон, навчають модель самокритиці та виправленню відповіді відповідно до цих правил. Є й керування під час самої відповіді: системні підказки (роль, тон), шаблони, перевірки на небажаний вміст, вимога повертати структурований формат – усе це швидко вмикається і вимикається, але інколи може «перекрити» корисні відповіді [25, 20].

Для безпеки використовують додаткові техніки: навчання відмовлятися у ризикованих сценаріях, тренування на «підступних» запитах, систематичні перевірки на шкідливий чи упереджений контент. На практиці підходи часто поєднують: спершу інструкційне навчання, далі дані з людськими або «модельними» уподобаннями, плюс правила і фільтри під час відповіді – так досягають балансу між якістю, безпекою, швидкодією й трудовитратами [43].

Дані для узгодження поведінки бувають кількох типів. По–перше, це пари «запит –> бажана відповідь» для інструкційного навчання: короткі інструкції користувача і приклади правильних відповідей. По–друге, дані уподобань: для одного запиту є дві (або більше) відповіді, і людина позначає, яка краща – такі позначки потрібні, коли хочемо «підтягнути» стиль і корисність відповіді до очікувань користувачів. По–третє, «безпечні» й «ризикові» приклади для перевірок: запити, на які модель має ввічливо відмовлятися або відповідати обережно (наприклад, потенційно шкідливі інструкції). Джерела – публічні набори для навчальних/дослідницьких цілей, внутрішні дані організації (якщо є дозволи), а інколи – обмежено синтетичні приклади, створені іншою моделлю чи за шаблонами, але з обов’язковою ручною перевіркою якості. Під «датасетом» тут маємо на увазі впорядкований набір таких прикладів із чіткими полями (запит, відповідь або вибір «краще/гірше», мітки безпеки, примітки тощо) [20, 32].

Якість даних критично важлива. Потрібні репрезентативність (щоб приклади справді відображали майбутні сценарії), покриття різних тем і складності, відсутність «засмічення» (дублі, помилки, образливий або випадковий текст), чисті розділення на тренування/перевірку/тест без перетинів, узгоджені інструкції для анотувальників і перевірка їхньої згоди між собою. Бажано фіксувати, яку саме версію набору використано (наприклад, умовне «v1.2») і коротко описувати зміни – це допомагає чесно порівнювати результати між командами та в часі. Автоматичні перевірки (скрипти для пошуку дублікатів, помилкових міток, «витоків» тесту в тренування) мають доповнювати вибірковий ручний перегляд складних випадків [39].

Етичні вимоги стосуються і джерел, і процесу. Дані мають мати зрозумілу ліцензію та правові підстави для використання;приватна інформація (імена, телефони, адреси, медичні відомості) – вилучатися або анонімізуватися, при цьому люди, які вручну перевіряють і оцінюють відповіді моделі (оцінювачі/розмічувачі даних), мають працювати за чіткими інструкціями, отримувати належну компенсацію та користуватися інструментами захисту, адже частина матеріалів може бути чутливою . Важливо контролювати й зменшувати упередженість: не допускати, щоб у наборах переважали тексти, які дискримінують групи людей;а також окремо тестувати модель на справедливість і відсутність токсичності. Якщо використовуються «моделі–судді» (коли інша модель автоматично оцінює відповіді), результати потрібно періодично звіряти з людськими оцінками й явно зазначати обмеження такого підходу [27].

Щоб бачити картину цілком, зведемо розглянуті підходи в одну коротку таблицю для швидкого порівняння (табл. 1.1).

Таблиця 1.1 – Порівняння підходів оцінки узгодження поведінки великих мовних моделей

| **Підхід** | **Що потрібно з даних** | **Сильні сторони** | **Обмеження / ризики** | **Де доречно** | **Орієнтовна складність** |
|---|---|---|---|---|---|
| Інструкційне навчання (тонке налаштування на інструкціях) | Пари «запит –> бажана відповідь» | Простий старт, стабільність, помірні витрати | Дуже залежить від якості прикладів;обмежений контроль безпеки | Базова адаптація під завдання | Низька–середня |
| Уподобання людини (RLHF/пряме навчання на виборах «краще/гірше») | Для одного запиту – кілька відповідей і вибір людиною кращої | Ближче до очікувань користувачів, покращує корисність | Дорога розмітка, потрібні чіткі інструкції оцінювачам, можливі упередження | Коли важливі якість і тон відповіді | Середня–висока |
| Уподобання на основі моделі («модель–суддя» – інша модель оцінює відповіді) | Пари відповідей;конфігурація «судді» | Швидке й дешевше збирання оцінок | Може помилятися, потрібно періодично звіряти з людиною | Попередній відбір, скринінг варіантів | Середня |
| Правилові / «конституційні» підходи (набори принципів і заборон) | Набір правил/політик; приклади виправлень | Менше небажаного контенту без масової розмітки | Неповне покриття правил;ризик «перекриття» корисних відповідей | Домени з чіткими політиками | Середня |
| Узгодження під час відповіді (системні підказки, шаблони, фільтри) | Переважно конфігурації, без навчальних даних | Швидко вмикається, гнучке керування | Можна обійти хитрими запитами;нестабільність у складних кейсах | Швидкі пілоти, інтеграції | Низька |
| Спеціальні техніки безпеки (відмова у ризикових кейсах, «підступні» тести, red–teaming) | Сценарії ризиків і перевірок | Знижують токсичність і шкідливі поради | Потребують експертизи й постійного оновлення сценаріїв | Високоризикові застосування | Середня–висока |
| Комбіновані стратегії (мікс підходів) | Комбінація попередніх типів даних | Кращий баланс якості/безпеки/швидкодії | Складніше проєктування та підтримка | Пілоти й впровадження у продуктах | Висока |

На основі даних таблиці можна зробити наступні висновки. Універсально найкращого способу немає: кожен підхід має свої плюси й мінуси, тож вибір

залежить від цілей і обмежень. Найшвидше стартувати з простого донавчання на інструкціях і керування під час відповіді (системні підказки, фільтри) – це дає помітний ефект за невеликі витрати. Якщо потрібна вища якість і «людяність» відповідей, варто додавати дані з виборами людей «яка відповідь краща», але це дорожче й довше. Правила безпеки й «конституційні» підходи добре зменшують небажаний контент, зате інколи урізають корисні відповіді – їх треба налаштовувати обережно. Підсумок сильно залежить від даних для оцінювання: що кращі й чесніші набори завдань, то надійніші висновки. На практиці найкраще працює комбінація: базове інструкційне донавчання, поверх – правила безпеки, а за потреби – шар із даними людських уподобань;усе це регулярно міряти на узгоджених тестах і дивитися на баланс «якість–безпека–швидкість–витрати».

### 1.3. Критерії й метрики оцінювання та постановка задачі

Критерії – це те, що оцінюється, а метрики – як саме це вимірюється. У нашому випадку головні критерії такі: якість відповіді (наскільки вона точна, корисна й по суті), безпека (відсутність шкідливого чи небажаного контенту й коректні відмови на ризикові запити), чесність/відсутність упереджень (щоб модель не дискримінувала групи людей), стабільність і надійність (чи дає подібні результати за схожих умов, чи не «ламається»), ефективність (час відповіді та орієнтовна вартість використання), а також керованість (наскільки добре модель дотримується інструкцій і потрібного формату, наприклад коли відповідь має бути в JSON). Для кожного критерію визначаємо прості метрики: точність/релевантність на стандартних тестових наборах (бенчмарках), частку токсичних або небезпечних відповідей, різницю якості між групами (показник справедливості), середній час відповіді, кількість використаних «токенів» як наближення до вартості, частку збоїв і таймаутів, відсоток відповідей, що суворо відповідають заданому формату [22].

Вимірювання проводимо на заздалегідь підготовлених тестових наборах завдань (впорядковані списки запитів із правилами перевірки або еталонними

відповідями). Метрики можна обчиснювати трьома способами: автоматичними правилами (наприклад, детектори токсичності), людською оцінкою (коли експерти переглядають і виставляють бали за зрозумілою шкалою) та за допомогою моделі–«судді» (її висновки потрібно періодично звіряти з людськими оцінками, бо вона може помилятися). Щоб порівняння було чесним, усі варіанти тестуємо в однакових умовах: ті самі набори запитів, однакові налаштування й ліміти, достатній обсяг прикладів і базова статистична перевірка (наприклад, довірчі інтервали для середніх значень) [5, 22].

Результати зручно подавати у вигляді короткої «панелі»: для кожного підходу показуємо ключові метрики якості, безпеки, ефективності та керованості, а також короткий коментар, що пояснює помічені компроміси (наприклад, «вища безпека – трохи повільніше»). На цьому етапі не розставлено пріоритети між критеріями – просто чесно вимірюємо й фіксуємо показники; питання «що важливіше саме для нашого сценарію» вирішуватимемо окремо, коли формулюватимемо постановку задачі й правила вибору.

Щоб результати було чесно порівнювати і на їх основі приймати рішення, потрібно зафіксувати рамки, у яких працюємо, і мінімальні вимоги до якості. Далі простою мовою пояснюємо, що саме вважаємо обмеженнями та якими мають бути пороги для проходження тестів.

Обмеження – це умови середовища: скільки часу модель може витрачати на відповідь;які обчислювальні ресурси доступні;які є вимоги до безпеки і політик (що не можна казати/радити);які дані дозволено використовувати за ліцензіями і правилами приватності. Також сюди входять вимоги до формату відповідей (наприклад, коли потрібен строгий JSON) і до стійкості роботи (щоб система не «падала») [7, 30, 43].

Пороги – це конкретні числа або правила «пройшов/не пройшов». Поділяються вони на три рівні. По–перше, обов’язкові пороги: якщо метод їх не виконує, він відсіюється одразу. Типові приклади – дуже низька частка шкідливих відповідей на тесті безпеки, коректні відмови на заборонені запити, відсутність приватних даних у відповідях, відповідність формату там, де це

критично, і прийнятний час відповіді. По–друге, цільові пороги: бажані рівні якості й швидкодії, до яких прагнемо (наприклад, вища точність на еталонних завданнях). По–третє, спостережні пороги: показники, за якими стежимо (наприклад, різниця якості між групами користувачів), але які самі по собі не «валять» метод, якщо обов’язкові умови виконані.

Числові значення порогів беруться з вимог замовника та правил домену (наприклад, внутрішніх політик безпеки або законодавчих обмежень). Якщо таких вимог немає, орієнтуємося на опубліковані рекомендації і ринкову практику та погоджуємо конкретні значення з учасниками проєкту. Щоб не робити висновки «з двадцяти прикладів», забезпечуємо достатній розмір тесту (достатньо багато запитів), а результати подаємо з оцінкою похибки – так легше побачити, чи різниця між методами справді значуща. Важливо також чітко назвати, на якому саме тестовому наборі все мірялося (назва і версія), щоб інші змогли повторити перевірку [11, 16].

Процедура застосування порогів проста: спершу перевіряємо обов’язкові умови безпеки, формату й часу відповіді;методи, які їх не виконують, до подальшого порівняння не допускаються. Далі серед тих, що пройшли, дивимося на цільові показники (якість, швидкість, орієнтовні витрати) і обираємо кращі з урахуванням контексту застосування. Якщо метод трохи не дотягує до цільового рівня, але має явні переваги в іншому критично важливому показнику, це фіксується в рішенні окремим коментарем і може бути прийнято як обґрунтований компроміс [31].

Після того як визначено, що саме міряємо і які маємо пороги, визначимо, як ухвалюємо рішення про вибір підходу.

Формулювання задачі: маємо кілька кандидатних підходів до узгодження. На однакових тестових наборах та в однакових умовах для кожного підходу вимірюємо ключові показники (якість відповіді, безпека, стабільність, швидкість, орієнтовні витрати) і застосовуємо правила відбору. Мета – обрати підхід (або невелику комбінацію), який дає найкращий баланс за цими показниками і не порушує жорстких порогів безпеки, формату та часу відповіді.

Вхідні дані задачі:

– перелік кандидатів;

– опис тестових наборів (назва, версія);

– список метрик;

– обов’язкові та цільові пороги;

– короткий опис контексту використання (що важливіше саме тут – якість, безпека чи швидкодія).

Виходом тут має бути ранжований список з чіткою рекомендацією і коротким поясненням «чому саме цей варіант».

Правило вибору (простий і прозорий підхід) буде наступним – рухаємось у два кроки: спочатку використовуємо «санітарний фільтр»: відсіюємо всі варіанти, які не пройшли обов’язкові пороги (безпека, формат, граничний час відповіді). Далі серед тих, що залишилися, порівнюємо їх за кількома критеріями одночасно. Якщо замовник може назвати пріоритети, перетворюємо метрики на шкалу 0–1 і обчислюємо простий підсумковий бал з вагами (ваги – це числове відображення пріоритетів, які задаються стейкхолдерами і погоджують до запуску тестів). Якщо пріоритети поки не визначені, використовуємо логіку Парето простою мовою: дивимось, хто «не гірший ні в чому і кращий хоча б у чомусь», – з таких «недомінованих» обираємо той, що найкраще відповідає контексту застосування (наприклад, трохи повільніший, але помітно безпечніший – для чутливої доменної області). Для надійності робимо коротку перевірку чутливості: якщо трохи змінити пріоритети, рекомендація не повинна радикально змінюватися.

Критерії успіху рішення, тобто вибору підходу будуть:

– пройшов усі обов’язкові пороги (безпека, формат, час відповіді);

– демонструє покращення над базою (наприклад, над простим інструкційним донавчанням) на узгоджених метриках якості та/або безпеки;

– вкладається у прийнятні часові та вартісні обмеження;

– результат відтворюється – повторний запуск на тій самій версії тестів дає близькі числа;

– супроводжується зрозумілим звітом: які тести, які метрики, які компроміси і чому зроблено саме такий вибір.

Щоб вибір був прозорим і повторюваним, зафіксуймо простий покроковий план перевірок.

Спочатку готуємо все необхідне: визначаємо версії тестових наборів (назва і версія), формуємо базовий варіант для порівняння і список кандидатів, фіксуємо умови запуску (параметри генерації відповіді, ліміти довжини, час очікування, середовище – локальна машина чи API). Для відтворюваності зберігаємо конфігурації та «зерно випадковості» (щоб повторні запуски давали близькі результати).

– Далі запускаємо оцінювання для кожного кандидата на тих самих наборах: якість на еталонних завданнях;

– безпека – частка небажаних відповідей і коректні відмови на заборонені запити;

– чесність/відсутність упередженості – спеціальні перевірки;

– керованість – дотримання формату (наприклад, вимога повернути валідний JSON) і інструкцій;

– надійність – частка збоїв і таймаутів;(д) ефективність – середній час відповіді та орієнтовні витрати (за потреби оцінюємо їх за обсягом обробленого тексту або тарифами постачальника) [35, 29].

Спершу застосовуємо обов’язкові пороги (безпека, формат, граничний час): хто їх не пройшов – вибуває.

Для контролю якості додаємо ручну перевірку на невеликій вибірці (наприклад, 100–200 прикладів): експерти оцінюють корисність і безпеку за простим чек–листом;зіставляємо ці оцінки з автоматичними. Якщо розбіжності перевищують заданий поріг, уточнюємо тести або інтерпретацію результатів.

Основні метрики подаємо з довірчими інтервалами й повторюємо частину запусків, щоб перевірити відтворюваність [12].

Після цього формуємо підсумкове порівняння: одна таблиця з ключовими метриками і короткий коментар про компроміси (де краща якість, де швидше, де безпечніше). Якщо пріоритети вже визначені замовником, розраховуємо підсумковий бал з урахуванням цих пріоритетів;якщо ні – користуємось простим принципом Парето (обираємо варіант, що не гірший ні в чому суттєвому і кращий хоча б у чомусь важливому). Додаємо перевірку чутливості: трохи змінюємо пріоритети або параметри і дивимось, чи рекомендація зберігається.

Нарешті, готуємо звіт і артефакти: версії наборів, конфігурації, журнали, графіки/таблиці, а також явний перелік обмежень. Окремо фіксуємо умови переходу в пілот (що вмикаємо, як моніторимо прості інциденти, коли робимо повторний тест після змін). Це дозволяє іншій команді відтворити наші результати і продовжити роботу без «прихованих налаштувань».

Для подальшого моделювання введемо прості математичні означення базових метрик. точність (Accuracy) обчислюємо як:

$$\text{Accuracy} = N_{corr} / N \tag{1.1}$$

де N – загальна кількість тестових запитів (завдань),

$N_{corr}$ – кількість відповідей, які вважаються правильними (за еталонною відповіддю або людською оцінкою),

$N_{unsafe}$ – кількість відповідей, що містять небажаний або небезпечний контент.

а частку небезпечних відповідей – обчислюємо як:

$$\text{UnsafeRate} = N_{unsafe} / N \tag{1.2}$$

Якщо якість відповіді оцінюють люди або окрема модель–«суддя» за бальною шкалою (наприклад, від 1 до 5), то для N оцінених відповідей із балами $score_i$ середній бал дорівнює:

$$\text{Score}_{avg} = (1 / N) * \Sigma\ score_i \tag{1.3}$$

Для окремих тестів на безпеку вводимо додаткові показники.

$$\text{ToxicityRate} = N_{toxic} / N \tag{1.4}$$

$$RefusalRate = N_{refusal\_ok} / N\_{risk} \quad (1.5)$$

де $N_{toxic}$ – кількість відповідей, ідентифікованих як токсичні,

$N_{risk}$ – кількість «ризикових» запитів (на заборонені теми),

$N_{refusal\_ok}$ – кількість коректних відмов на такі запити.

Для керованості нас цікавить дотримання формату відповіді.

$$FormatAdherence = N_{format\_ok} / N \quad (1.6)$$

де $N_{format\_ok}$ – кількість відповідей, що пройшли перевірку формату (наприклад, є валідним JSON там, де це вимагалося). Тоді:

Ефективність характеризуємо передусім часом відповіді та приблизними витратами. Якщо $time_{start\_i}$ та $time_{end\_i}$ – час початку і завершення обробки і–го запиту, то середній час відповіді:

$$Latency_{avg} = (1 / N) * \Sigma (time_{end\_i} – time_{start\_i}) \quad (1.7)$$

У випадку використання зовнішнього програмного інтерфейсу (API) орієнтовну вартість можна оцінити через кількість обробленого тексту. Нехай

$Tokens_{prompt\_i}$ та $Tokens_{completion\_i}$ – кількість «токенів» (частин тексту) у запиті та відповіді для і–го прикладу,

$c_{in}$ і $c_{out}$ – тарифи постачальника за токен на вході та виході.

Тоді середня кількість токенів на одну взаємодію:

$$Tokens_{avg} = (1 / N) * \Sigma (Tokens_{prompt\_i} + Tokens_{completion\_i}) \quad (1.8)$$

а орієнтовна сумарна вартість запуску оцінювання:

$$Cost = c_{in} * \Sigma Tokens_{prompt\_i} + c_{out} * \Sigma Tokens_{completion\_i} \quad (1.9)$$

У подальшому ці окремі метрики (якість, безпека, керованість, ефективність) будуть використані в моделі багатокритеріального вибору (розділ 2) та в експериментальній частині роботи для порівняння різних методів узгодження поведінки.

## 1.4 Огляд сучасних наукових робіт та програмних рішень у сфері автоматичного протоколювання

Проблематика узгодження поведінки великих мовних моделей (LLM) з людськими цінностями, нормами безпеки та доменними вимогами за останні роки перетворилася на один із центральних напрямів досліджень у галузі штучного інтелекту. Якщо початкові роботи зосереджувалися переважно на покращенні якості відповіді за рахунок класичних методів донавчання, то із появою підходів RLHF, RLAIF, DPO та «конституційного» навчання фокус змістився до побудови цілісних конвеєрів узгодження, які включають збір преференцій, побудову reward–моделей, оптимізацію політики та подальшу експлуатацію моделі в умовах реальних ризиків.

У цьому контексті актуальною стає розробка інформаційних систем, які не просто реалізують конкретний алгоритм, а допомагають обирати й оцінювати методи узгодження для конкретних прикладних задач.

Поворотним моментом для сучасного розуміння узгодження стало запропонований триетапний конвеєр RLHF: спочатку модель донавчається на демонстраціях експертів (SFT), потім навчається reward–модель на основі парних порівнянь відповідей, після чого застосовується навчання з підкріпленням (PPO) з використанням цієї reward–моделі [25]. Отримана лінійка моделей InstructGPT продемонструвала, що навіть менша за розміром модель, але узгоджена з людськими преференціями, може перевершувати «сиру» LLM значно більшого масштабу з погляду корисності, правдивості та зменшення токсичності.

Подальший розвиток ідеї масштабування людського нагляду привів до появи підходів Reinforcement Learning from AI Feedback (RLAIF) та «конституційного» навчання. Запропоновано використовувати набір принципів (умовну «конституцію») та допоміжну модель–«критика», яка генерує самооцінки й виправлені варіанти відповідей [31]. На основі цих даних навчається модель преференцій і виконується RL уже не з людським, а з AI–фідбеком. Показано, що такий підхід дає змогу отримати помітно більш «нешкідливого» помічника без великого обсягу ручної розмітки, водночас зменшуючи схильність моделі до надмірно ухильних відповідей.

Окремий напрям досліджень складають оглядові роботи та таксономії методів узгодження. Сучасні огляди RLHF та споріднених технік систематизують весь спектр підходів – від класичного RLHF до RLAIF, DPO, self–оцінювання та мультимодального узгодження [31]. У таких роботах порівнюються відкриті моделі (LLaMA 2/3, Mistral, Mixtral, Falcon, OpenAssistant, Alpaca, Zephyr тощо) за різними метриками корисності, безпечності, упередженості та ефективності, підкреслюється складність компромісу між «helpful» і «harmless», а також необхідність доменно–специфічних критеріїв оцінювання.

Значний розвиток отримали й альтернативні до RLHF методи, спрямовані на спрощення або здешевлення конвеєра узгодження. На відміну від RLHF, DPO дозволяє оптимізувати політику без явного навчання reward–моделі та без окремого RL–етапу, зводячи задачу узгодження до класифікації пар «краща/гірша відповідь» і спрощуючи реалізацію [31].

Паралельно розвиваються дослідження, присвячені якісній оцінці методів узгодження. Наприклад, у рамках проєкту HELM (Holistic Evaluation of Language Models) запропоновано багатовимірне оцінювання LLM за сімома групами метрик (accuracy, calibration, robustness, fairness, bias, toxicity, efficiency) у різних сценаріях використання. Це дозволяє аналізувати вплив конкретної стратегії узгодження не лише на якість відповідей, а й на токсичність, упередженість та інші небажані ефекти.

Водночас у літературі дедалі частіше підкреслюється, що процес узгодження сам по собі стає потенційною ціллю атак (наприклад, спотворення даних преференцій, adversarial feedback, prompt injection під час збору або оцінки відповідей). Це приводить до ідеї, що необхідно оцінювати не лише кінцеву модель, а й надійність самого конвеєра оцінювання, включно з джерелами фідбеку, конфігурацією навчальних фреймворків та використаними системами безпеки.

Таким чином, наукові дослідження забезпечують потужну теоретичну та алгоритмічну базу для узгодження LLM, однак дають обмежену підтримку з

точки зору практичного вибору методу для конкретної організації чи прикладної задачі. Це й створює нішу для спеціалізованої інформаційної системи, орієнтованої на підтримку прийняття рішень щодо методів узгодження.

Паралельно з теоретичними дослідженнями стрімко розвивається екосистема програмних фреймворків і платформ, що реалізують різні етапи конвеєра узгодження поведінки LLM. Їх умовно можна поділити на три підгрупи фреймворки для навчання та оцінювання, системи оцінювання та тестування, інструменти guardrails і безпеки.

Одним із ключових інструментів для практичної реалізації RLHF/DPO є бібліотека TRL (Transformer Reinforcement Learning) від Hugging Face [13]. Вона надає повний стек для SFT, reward modeling, PPO/GRPO, DPO та споріднених методів, щільно інтегрований з екосистемою transformers. Це дозволяє порівняно «швидко» будувати власні конвеєри узгодження на базі відкритих моделей, але вимагає глибокої ML–компетенції та ручної конфігурації гіперпараметрів.

Фреймворк DeepSpeed–Chat, розроблений Microsoft на базі DeepSpeed, орієнтований на масштабне RLHF–навчання «chatGPT–подібних» моделей [14]. Він реалізує повний трьохетапний конвеєр RLHF, описаний в InstructGPT, та оптимізації для тренування моделей обсягом до сотень мільярдів параметрів за прийнятних ресурсних витрат. Головний акцент – продуктивність і масштабованість, тоді як питання вибору конкретної стратегії узгодження лишаються на розсуд інженера.

Сучасним прикладом високопродуктивного оцінювання–фреймворку є OpenRLHF, який поєднує DeepSpeed, Ray і vLLM та підтримує PPO, REINFORCE++ та інші алгоритми [14]. Запропонований алгоритм REINFORCE++ усуває потребу в окремій критик–моделі, використовуючи нормалізовану винагороду як baseline і зменшуючи обчислювальні витрати порівняно з класичним PPO, що робить RLHF більш доступним для тренування великих моделей (до 70B параметрів і більше).

Важливо, що всі зазначені фреймворки реалізують конкретні алгоритми узгодження (RLHF, DPO, RLAIF–подібні схеми), орієнтовані на розробників і

дослідників, які самостійно визначають стратегію експериментів. Не пропонують вбудованого механізму порівняння методів узгодження між собою за набором узгоджених метрик (якість, безпека, вартість, залежність від людського фідбеку).

Для незалежного оцінювання поведінки LLM існує низка фреймворків, які можуть виступати джерелом даних для інформаційної системи.

lm–evaluation–harness від EleutherAI – універсальний фреймворк для few–shot оцінювання мовних моделей на великій кількості бенчмарків (переклад, логіка, QA тощо), з підтримкою різних форматів задач і конфігурацій моделей [15].

HELM (Holistic Evaluation of Language Models) – «живий» бенчмарк і Python–фреймворк, що дозволяє оцінювати моделі за багатьма метриками (точність, робастність, справедливість, токсичність, ефективність) у широкому спектрі сценаріїв [15].

OpenAI Evals – open–source фреймворк і хмарний сервіс для створення кастомних eval'ів, що дає змогу систематично оцінювати як окремі моделі, так і цілі LLM–системи відповідно до бізнес–вимог, з можливістю роботи через API та графічний інтерфейс [16].

DeepEval / Confident AI – open–source фреймворк (DeepEval) і хмарна платформа (Confident AI) для unit–тестування та регресійного тестування LLM–додатків [16]. Вони включають десятки дослідницьких метрик (G–Eval, RAGAS, метрики галюцинацій, релевантності, узгодженості тощо) і дозволяють будувати CI/CD–конвеєри для моніторингу якості й безпеки LLM у продакшені.

Хоча ці інструменти й забезпечують потужні можливості для бенчмаркінгу та моніторингу, вони оцінюють конкретні реалізації моделей, а не методи узгодження як такі, не надають користувачу інтегрованого інтерфейсу для порівняння різних стратегій оцінювання за множиною критеріїв і в різних доменах.

Третій клас рішень – це guardrails–фреймворки, які працюють на етапі виконання і контролюють поведінку LLM у реальному часі.

NVIDIA NeMo Guardrails – open–source toolkit, який вводить спеціальну мову Colang 1.0/2.0 для опису діалогових сценаріїв і політик безпеки [17]. Colang дозволяє задавати дозволені/заборонені теми, формати відповідей, логіку виклику інструментів і реакцію на потенційно небезпечні запити, а Python–рантайм виконує ці правила під час взаємодії з користувачем.

Guardrails AI (open–source бібліотека Guardrails + хмарний сервіс) забезпечує опис «рейок» у вигляді input/output–guard'ів, які перевіряють структуру, контент і ризики у відповідях моделі [18]. Підтримуються схеми валідації, повторні виклики моделі, а також каталог готових guardrails у Guardrails Hub.

Lakera Guard позиціонується як платформа для реального часу захисту LLM–додатків від prompt injection, jailbreak–атак, витоків даних та іншого зловмисного використання, з єдиним API, централізованим моніторингом та готовими політиками безпеки [19].

Ці рішення забезпечують операційний рівень узгодження (контроль того, що робить модель «тут і зараз»), але майже не враховують, як саме модель була узгоджена під час навчання. Не дають змоги систематично порівняти, наприклад, «RLHF + NeMo Guardrails» проти «DPO + Guardrails AI» за вартістю, ризиками та якістю для конкретного бізнес–сценарію.

Проведений аналіз показує, що сучасний стан програмних рішень у предметній області характеризується високою насиченістю низькорівневих фреймворків для реалізації різних методів узгодження (TRL, DeepSpeed–Chat, OpenRLHF тощо). Наявністю потужних платформ оцінювання (HELM, lm–evaluation–harness, OpenAI Evals, DeepEval/Confident AI), які дозволяють детально вимірювати поведінку моделей. Розвиненою екосистемою guardrails–інструментів (NeMo Guardrails, Guardrails AI, Lakera Guard), орієнтованих на безпеку та комплаєнс у продакшені.

Водночас існує суттєва прогалина на «meta–рівні»: бракує інтегрованої інформаційної системи підтримки вибору та оцінювання методів узгодження, яка агрегувала б знання про доступні методи оцінювання (RLHF, RLAIF, DPO,

Constitutional AI тощо), фреймворки їх реалізації та типові сценарії застосування та дозволяла б порівнювати методи між собою за сукупністю показників (якість, безпека, вартість, потреба в людському фідбеку, вимоги до інфраструктури, регуляторні обмеження)

Саме розроблювана в даній роботі інформаційна система підтримки вибору та оцінювання методів узгодження поведінки ВЛМ покликана частково заповнити цю прогалину, надаючи користувачам (розробникам, аналітикам, доменним експертам) єдиний інструмент для системного аналізу, порівняння й обґрунтованого вибору підходів до оцінювання у конкретних умовах застосування.

**Висновки до Розділу 1.**

У першому розділі уточнено місце задачі узгодження поведінки великих мовних моделей у сучасному контексті впровадження ШІ та сформульовано дослідницьку задачу, яку має розв’язувати інформаційна система. Показано, що масове використання LLM у прикладних сервісах поєднує значний потенціал із ризиками хибних і токсичних відповідей, упередженості, регуляторних обмежень та високої вартості експериментів. За таких умов інтуїтивний, «ручний» вибір методів узгодження є недостатньо обґрунтованим і погано відтворюваним, що обумовлює потребу у спеціалізованій системі підтримки рішень.

В подальшому чітко визначено об’єкт і предмет дослідження, межі домену та основні припущення. Об’єктом є процес узгодження поведінки текстових LLM у прикладних системах, предметом – методи, дані, метрики та програмні засоби, що забезпечують підтримку вибору й оцінювання підходів до узгодження. Обґрунтовано доцільність фокусування на текстових моделях загального призначення, офлайн–оцінюванні на підготовлених наборах завдань і прикладних способах узгодження (інструкційне навчання, методи на основі уподобань, правилові та «конституційні» підходи, guardrails), без виходу на

рівень розробки нових базових архітектур чи повних MLOps–конвеєрів. Це дозволило задати чіткі й реалістичні рамки дослідження.

Наступним проаналізовано коло стейкхолдерів та їхні потреби: дослідникам і ML–інженерам потрібні відтворювані експерименти та узгоджені метрики, продакт–менеджерам – прозорі порівняльні звіти з балансом «якість – безпека – швидкодія – вартість», фахівцям із безпеки й комплаєнсу – стабільні протоколи перевірок і журнали, DevOps–командам – зрозумілі механізми інтеграції, анотаторам і тестувальникам – зручні інтерфейси для ручної оцінки й порівняння результатів. Таким чином показано, що система має підтримувати не лише обчислення, а й колективний процес прийняття рішень у мультидисциплінарній команді.

Виконано систематизацію основних підходів до узгодження поведінки LLM і типів даних. Виокремлено інструкційне донавчання, методи на основі людських уподобань (RLHF, пряме навчання на виборах «краще/гірше»), підходи з моделлю–суддею, правилові та «конституційні» методи, керування на етапі генерації (системні підказки, шаблони, фільтри) та спеціальні техніки безпеки й їхні комбінації. Порівняння цих підходів за вимогами до даних, сильними сторонами, обмеженнями та складністю впровадження показало відсутність універсально найкращого рішення й продемонструвало переваги комбінованих стратегій, які поєднують інструкційне донавчання, правила безпеки та, за потреби, використання даних людських уподобань.

# 2. МОДЕЛІ ТА СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1. Концептуальна й інформаційна модель системи

Концептуальна модель описує, що саме зберігає та обробляє інформаційна система, і як ці сутності пов’язані між собою, без прив’язки до конкретної СУБД чи технологій реалізації. Для системи підтримки вибору та оцінювання методів узгодження поведінки великих мовних моделей у центрі моделі лежить поняття експерименту узгодження, навколо якого групуються дані про моделі, методи, набори завдань, метрики, користувачів та результати.

Базовим «контейнером» верхнього рівня є проєкт. Проєкт відповідає певному прикладному сценарію використання моделі (наприклад, «асистент підтримки користувачів», «освітній чат–бот» тощо) і містить опис цілей, обмежень і бізнес–контексту. У межах одного проєкту може бути кілька експериментів узгодження, які досліджують різні методи, конфігурації або версії моделей для цього ж сценарію.

Кожен експеримент завжди пов’язаний із конкретною версією мовної моделі. Модель розглядаємо як сутність «Модель» з атрибутами: назва, постачальник (відкрита модель, комерційний API тощо), версія, базовий розмір (кількість параметрів), режим доступу. Окремі експерименти можуть використовувати різні версії моделей, а одна й та сама версія може фігурувати в багатьох експериментах, що дозволяє порівнювати методи узгодження за фіксованої моделі [12].

Другий ключовий об’єкт – метод узгодження. Для нього зберігаються тип (інструкційне донавчання, методи на основі уподобань, правилові/«конституційні» підходи, комбіновані варіанти тощо), короткий опис, основні вимоги до даних (які потрібні набори завдань, чи потрібні оцінки людей), очікувані переваги й ризики. Один метод може застосовуватися в різних експериментах (наприклад, на різних моделях або для різних прикладних

сценаріїв), а один експеримент завжди має посилання на конкретний метод узгодження, який у ньому оцінюють [15].

Третя група сутностей пов’язана з даними. Узагальнено це сутність «Набір даних» (dataset), для якої фіксується назва, версія, джерело, тип (навчальний, валідаційний, тестовий, безпековий, бенчмарк), ліцензія, короткий опис змісту та структура завдань. Їх можна розділити принаймні три підтипи:

– набори для навчання/донавчання моделі або методу узгодження;

– набори для оцінювання якості (еталонні запитання з правильними або «еталонними» відповідями);

– безпекові й ризикові набори для перевірки на шкідливий, токсичний або небажаний контент.

Один набір даних може використовуватися в багатьох експериментах, а кожен експеримент може комбінувати кілька наборів (наприклад, один якісний та один безпековий) [6].

Щоб описати як саме буде проводитися оцінювання, вводимо сутність конфігурації оцінювання. У ній задаються: перелік метрик, які потрібно обчислювати (точність, частка небезпечних відповідей, час відповіді, дотримання формату та ін.), пороги/обмеження для окремих показників, параметри генерації відповідей (наприклад, максимальна довжина відповіді та параметр «температура», який керує ступенем випадковості генерації), ліміти часу, кількість прогонів для повторюваності. Кожен експеримент посилається на одну конфігурацію оцінювання, але одна й та сама конфігурація може бути спільною для кількох експериментів, якщо вони виконуються за однаковими правилами [45, 24].

Сам експеримент узгодження у концептуальній моделі поєднує модель, метод узгодження, набір (або набори) даних і конфігурацію оцінювання. Для експерименту зберігаються атрибути:

– ідентифікатор;

– назва;

– мета;
– пов’язаний проєкт;
– відповідальний користувач;
– дата створення;
– статус (запланований, виконаний, у процесі).

Один експеримент може мати кілька запусків (runs): це окремі пробні виконання з фіксацією дати й часу, технічного середовища, параметрів та артефактів (вихідних відповідей, логів). Зв’язок «експеримент–запуск» є співвідношенням «один до багатьох»: один експеримент може мати кілька запусків, але кожен запуск належить рівно одному експерименту.

Для опису результатів оцінювання виділяємо дві сутності: визначення метрики (MetricDefinition) та значення метрики (MetricValue). Перша містить назву метрики, її тип (частка, середнє значення, час, вартість), короткий опис способу обчислення та напрямок оптимізації («більше – краще» або «менше – краще»). Друга містить конкретне числове значення метрики для певного запуску експерименту, а також інформацію про тестовий набір, на якому це значення було отримано. Через ці сутності забезпечується можливість зберігати й порівнювати результати різних запусків, експериментів і методів на спільних або різних наборах завдань [28].

На рівні представлення підсумків важливо мати сутність звіт. Звіт акумулює ключові результати одного або кількох запусків/експериментів у зручному для користувача вигляді: таблиці з метриками, графіки порівняння, текстові висновки. Один звіт пов’язується з одним або кількома експериментами, а один експеримент може мати кілька звітів (наприклад, технічний та управлінський варіанти).

Окрему групу складають користувачі та ролі. Концептуально виділяємо сутність «Користувач системи» з можливими ролями:

– інженер/дослідник (створює експерименти, налаштовує конфігурації, аналізує сирі результати);

– аналітик або продакт–менеджер (аналізує зведені звіти, приймає рішення щодо вибору методу);

– фахівець із безпеки/комплаєнсу (перевіряє результати тестів безпеки, затверджує пороги);

– адміністратор системи (керує правами доступу та технічними налаштуваннями).

Один користувач може брати участь у кількох проєктах та експериментах у різних ролях, а кожен експеримент має хоча б одного відповідального користувача.

Узагальнюючи, концептуальна модель системи включає такі ключові сутності:

– проєкт;
– експеримент;
– запуск експерименту;
– модель;
– метод узгодження;
– набір даних;
– конфігурація оцінювання;
– метрика та її значення;
– звіт;
– користувач/роль.

Між ними задаються відносини типу «один до багатьох» (проєкт–експерименти, експеримент–запуски, запуск–значення метрик, користувач–експерименти тощо) та «багато до багатьох» через проміжні сутності (наприклад, експеримент може одночасно використовувати кілька наборів даних, а один набір даних – входити до різних експериментів). На основі цього словесного опису у п.2.1 буде побудована інформаційна модель та схема даних, а в розділі 3 – UML– та інші діаграми, що деталізують реалізацію.

На основі описаної концептуальної моделі далі уточнимо, як саме ці сутності будуть представлені в базі даних, які атрибути зберігаються і як організовані зв’язки між ними.

Інформаційна модель системи відображає концептуальні сутності у вигляді структур даних, придатних для реалізації в системі керування базами даних. У рамках даної роботи передбачається використання реляційної моделі даних, де основними елементами є таблиці, їх атрибути, первинні та зовнішні ключі.

Життєвий цикл експерименту починається зі створення проєкту. На цьому етапі користувач (наприклад, інженер або аналітик) фіксує мету, прикладний сценарій, основні обмеження та очікувані критерії оцінювання. Далі створюється експеримент у межах вибраного проєкту: обирається версія мовної моделі з довідника моделей, метод узгодження, який планується досліджувати, та одна або кілька версій наборів даних із каталогу. Одночасно до експерименту прив’язується конфігурація оцінювання, у якій задаються активні метрики, параметри запуску та базові пороги якості й безпеки.

Після підготовки експеримент переходить у стадію запуску. Користувач створює один або кілька запусків (runs), для кожного з яких фіксуються дата й час, технічне середовище, версія коду та налаштування. Під час виконання запуску система опрацьовує вибрані набори даних через зазначену модель і метод узгодження, зберігає проміжні артефакти (вихідні відповіді, журнали) та обчислює значення метрик згідно з визначеннями з довідника метрик. Після завершення запуску результати прив’язуються до відповідного запису в таблиці Runs, а експеримент може отримати оновлений статус (наприклад, «виконаний»).

Далі результати агрегуються у вигляді звітів. На основі значень метрик для різних запусків система формує підсумкові таблиці, графіки та короткі текстові висновки, які зберігаються як окремі звіти й пов’язуються з експериментом. На цьому етапі аналітик або продакт–менеджер може порівнювати різні методи, сценарії та конфігурації. Після прийняття рішення експеримент може бути

переведений у стан «затверджено» або «архівовано», а в разі потреби – повторно запущений із новими параметрами, але з чіткою фіксацією, що це новий запуск у межах того самого експерименту.

Політика даних у системі базується на кількох принципах. По–перше, усі суттєві об’єкти (набори даних, моделі, метрики, конфігурації) мають версії. Для кожного набору даних фіксується номер версії та короткий опис змін, щоб можна було відтворити експеримент на тій самій вибірці. Експеримент і його запуски також не змінюються «заднім числом»: замість редагування історичних записів створюються нові запуски з оновленими параметрами, а старі залишаються доступними для перегляду. Це забезпечує відтворюваність результатів та прозорий аудит.

По–друге, чітко розділяються сирі дані й похідні артефакти. Сирі тексти запитів та відповідей, що можуть містити чутливу інформацію, зберігаються в окремих сховищах із обмеженим доступом, а в основній базі даних за потреби можуть зберігатися лише посилання на них або агреговані показники. Якщо в наборах даних потенційно є персональні чи конфіденційні відомості, перед завантаженням до системи вони мають бути анонімізовані або очищені відповідно до етичних і правових вимог.

По–третє, доступ до даних і операцій над ними регулюється ролями. Користувачі з роллю інженера можуть створювати експерименти й запуски, але не змінюють налаштувань безпеки. Фахівці з безпеки та комплаєнсу отримують доступ до звітів із безпекових тестів і можуть встановлювати або коригувати пороги. Адміністратор керує правами доступу, базовими налаштуваннями системи та резервним копіюванням. Передбачається ведення журналу дій (логів) принаймні для ключових операцій: створення та видалення експериментів, запусків, зміну конфігурацій оцінювання.

Нарешті, політика зберігання даних передбачає використання резервного копіювання та можливість архівації або видалення застарілих експериментів і їхніх артефактів згідно з внутрішніми правилами організації. При цьому метрики та короткі підсумкові звіти можуть зберігатися довше, ніж сирі дані, оскільки не

містять чутливої інформації, але важливі для побудови історії рішень. Така організація життєвого циклу експерименту та політики даних забезпечує баланс між відтворюваністю, прозорістю, вимогами до безпеки й раціональним використанням ресурсів.

### 2.2. Модель багатокритеріального вибору та методика оцінювання

Щоб перейти від окремих метрик до цілісного рішення, формалізуємо обраний підхід багатокритеріального вибору.

У нашій задачі потрібно порівнювати кілька методів узгодження поведінки за кількома критеріями одночасно: якість, безпека, керованість, ефективність тощо (розглядали в першому розділі ). Така постановка є класичною для багатокритеріального аналізу рішень – MCDA (Multiple Criteria Decision Analysis) [42]. З огляду на вимоги до прозорості й простоти реалізації, у роботі використано метод зваженої суми, який належить до адитивних моделей MCDA і добре підходить для ситуацій, коли всі критерії можна представити у вигляді числових показників.

Нехай маємо M кандидатних методів узгодження, пронумерованих j=1,…,M. Для кожного методу обчислюємо набір метрик, визначених у п.1.3 (точність, частка небезпечних відповідей, частка коректних відмов, дотримання формату, середній час відповіді, орієнтовна вартість тощо). Запишемо цей набір у вигляді вектора:

$$m_j=(m_{j1},m_{j2},\ldots,m_{jK}), \quad (2.1)$$

де K – кількість метрик (критеріїв), а $m_{j,k}$ – значення k–ї метрики для j–го методу.

Різні метрики мають різну природу й одиниці виміру (частки, секунди, вартість), тому перед агрегуванням їх потрібно привести до спільної шкали. Для цього застосовується лінійна нормалізація «мін–макс» до відрізка [0;1]. Спочатку для кожної метрики k розрізняємо два типи критеріїв:

– критерії типу «чим більше – тим краще» (якість, частка коректних відмов, дотримання формату, середній бал людської оцінки тощо);

– критерії типу «чим менше – тим краще» (частка небезпечних відповідей, частка токсичних відповідей, час відповіді, вартість).

Для кожної метрики k знаходимо мінімальне та максимальне значення серед усіх методів:

$$m_k^{min} = min_j\, m_{j,k}, \quad m_k^{max} = max_j\, m_{j,k}. \tag{2.2}$$

Тоді нормалізоване значення $r_{j,k}$ для методу j обчислюємо так:

Нормалізація для критеріїв типу «чим більше – тим краще»:

$$r_{j,k} = (m_{j,k} - m_k^{min}) / (m_k^{max} - m_k^{min}). \tag{2.3}$$

Нормалізація для критеріїв типу «чим менше – тим краще»:

$$R_{j,k} = (m_k^{max} - m_{j,k}) / (m_k^{max} - m_k^{min}) \tag{2.4}$$

У результаті для кожного критерію всі значення приводяться до діапазону від 0 до 1, де 0 відповідає найгіршому результату серед кандидатів, а 1 – найкращому. Якщо для деякої метрики mkmax =mkmin (усі методи показали однаковий результат), нормалізоване значення для всіх j можна покласти рівним 1, оскільки така метрика не впливатиме на порівняння.

Далі для кожного критерію k задаєм ваги wk≥0, які відображають його відносну важливість для конкретного сценарію застосування. Ваги нормуються так, щоб сума ваг дорівнювала 1

$$\sum_{k=1}^{K} w_k = 1. \tag{2.5}$$

Значення ваг узгоджуються зі стейкхолдерами (наприклад, можна надати більшої ваги безпеці у чутливих доменах та більшої ваги швидкодії у менш ризикових сценаріях).

Після нормалізації метрик та задання ваг для кожного методу узгодження обчислюємо підсумковий бал

$$S_j = \sum_{k=1}^{K} w_k \cdot r_{j,k}, \tag{2.6}$$

де $S_j \in [0;1]$ – агрегована оцінка j–го методу з урахуванням усіх критеріїв. Чим більшим є $S_j$ , тим кращим вважається метод за заданих пріоритетів.

Далі визначимо, як саме система будує рейтинг методів і перевіряє, наскільки стійким є цей рейтинг до змін налаштувань.

На першому кроці застосовується фільтр обов'язкових порогів. Для кожного методу перевіряється, чи виконує він базові вимоги до безпеки, часу відповіді, дотримання формату та інших критично важливих показників (пороги задаються на етапі постановки задачі). Якщо хоча б один обов'язковий показник виходить за допустимі межі, метод вважається неприйнятним і до подальшого порівняння не включається. Таким чином формується множина допустимих альтернатив, які задовольняють мінімальні вимоги.

На другому кроці серед допустимих методів виконується ранжування за підсумковим балом. Оскільки для кожного методу обчислено агреговане значення $S_j$ (воно враховує всі нормалізовані метрики та ваги критеріїв), система впорядковує методи за спаданням цього бала. Метод із найбільшим значенням $S_j$ отримує перше місце в рейтингу, далі йдуть методи з нижчими значеннями. Якщо різниця між декількома методами дуже мала (наприклад, менша за наперед узгоджений поріг), у звіті можна вказувати, що вони практично еквівалентні, і пропонувати їх як альтернативи з різними компромісами.

У випадках, коли пріоритети між критеріями ще не визначені однозначно, система може додатково показувати недоміновані альтернативи. Метод вважається недомінованим, якщо немає іншого методу, який був би кращим за всіма метриками одночасно і при цьому кращим хоча б за однією з них. Такий підхід наближений до аналізу за Парето і допомагає побачити набір «кращих» рішень без жорсткого нав'язування однієї комбінації ваг.

Щоб оцінити, наскільки стійким є отриманий рейтинг, проводиться чутливісний аналіз. Перший його аспект – зміна ваг критеріїв. Визначається кілька можливих сценаріїв (наприклад, сценарій із підвищеним пріоритетом безпеки, сценарій зі зміщенням акценту на швидкодію тощо), для кожного з них

перераховується підсумковий бал $S_j$ і будується новий рейтинг. Якщо один і той самий метод стабільно залишається у верхній частині списку (або на першому місці) для більшості розумних варіантів ваг, це свідчить про стійкість рекомендації.

Другий аспект чутливісного аналізу стосується невизначеності самих метрик. Оскільки значення метрик обчислюються на скінченних вибірках, для них можуть бути визначені довірчі інтервали або межі похибки. У межах чутливісного аналізу можна розглядати «крайні» сценарії, коли метрики наближаються до верхніх або нижніх меж своїх інтервалів, і перевіряти, чи змінюється порядок лідерів. Якщо навіть у цих сценаріях перші місця в рейтингу істотно не змінюються, то модель вибору можна вважати достатньо надійною.

У реалізованій системі результати ранжування та чутливісного аналізу передбачається подавати у вигляді:

– таблиці з рейтингом методів, їхніми підсумковими балами та ключовими метриками;

– короткого текстового коментаря про основні компроміси (наприклад, «метод А має вищу безпеку, але нижчу швидкодію, ніж метод В»);

– за можливості, простих графіків (діаграми порівняння, точки на площині «якість–безпека», «якість–вартість»).

Це дає користувачеві не лише числову рекомендацію, а й зрозумілу картину того, як було отримане рішення і як воно змінюється при різних налаштуваннях.

Після визначення моделі багатокритеріального вибору та способу ранжування потрібно зафіксувати, як саме виконуються оцінювання в системі, що необхідно зберігати для відтворюваності та в якому вигляді результати подаються користувачам.

Протокол експериментів описує послідовність кроків, за якою в системі проводиться оцінювання методів узгодження. На підготовчому етапі користувач обирає проєкт, формує перелік кандидатних методів, для яких потрібно здійснити порівняння, та фіксує сценарій оцінювання: версії наборів даних,

перелік метрик, пороги якості та безпеки, ваги критеріїв для багатокритеріальної моделі. Усі ці параметри заносяться до конфігурації оцінювання, яка прив’язується до конкретного експерименту.

Далі для кожного кандидата створюються запуски експерименту. У межах запуску система використовує обрану модель, метод узгодження та вказані набори даних, генерує відповіді на всі тестові запити із заданими параметрами (максимальна довжина відповіді, температура, обмеження часу тощо) та зберігає вихідні артефакти в окремому сховищі. Після завершення прогону автоматично обчислюються значення метрик, описаних у п. 1.3, і записуються до таблиці значень метрик для цього запуску. На основі цих значень система застосовує пороги, описані в розділі 1.3, і позначає, чи пройшов кандидат базові вимоги, чи ні.

Для забезпечення відтворюваності кожен запуск супроводжується фіксацією ключових параметрів середовища. Зберігаються: ідентифікатор і версія моделі (включно з ідентифікатором зовнішнього API, якщо використовується хмарний сервіс), версії наборів даних, ідентифікатор конфігурації оцінювання, дата й час запуску, інформація про програмне середовище (версія коду або контейнера, основні бібліотеки), а також, за потреби, значення початкового «зерна» випадковості. Це дозволяє повторити запуск у тих самих умовах або, принаймні, оцінити, які саме відмінності могли вплинути на результати.

Важливим елементом протоколу є повторні запуски. Для частини кандидатів (наприклад, лідерів за попередніми результатами) може бути передбачено кілька незалежних запусків на тому самому наборі даних, щоб оцінити варіативність метрик. Система зберігає всі такі запуски як окремі записи і дозволяє обчислювати усереднені значення та прості показники розкиду (наприклад, мінімум, максимум, стандартне відхилення). Це дає змогу не спиратися на одиничний прогін і покращує надійність порівняння.

Після обчислення метрик для допустимих методів система формує підсумкові оцінки за багатокритеріальною моделлю: нормалізує показники,

застосовує ваги та обчислює підсумковий бал для кожного кандидата. Результати подаються у двох основних формах. По–перше, це детальні таблиці, де для кожного методу наведені всі ключові метрики, інформація про проходження порогів та підсумковий бал. По–друге, це стислий підсумок у вигляді ранжованого списку з поясненням: які методи є лідерами, у чому їхні переваги та які компроміси спостерігаються (наприклад, «метод А має вищу безпеку, але трохи більший час відповіді порівняно з методом В»).

Для зручності аналізу система може додатково формувати прості графічні уявлення: діаграми порівняння метрик між методами, діаграми «якість–безпека», «якість–вартість» тощо. Кожен такий звіт пов’язується з конкретним експериментом і набором запусків, на основі яких він був сформований, а також має явні вказівки на використані ваги критеріїв і пороги. Це дозволяє користувачу відтворити кроки, за якими було отримано підсумкову рекомендацію.

Таким чином, протокол експериментів задає єдину послідовність дій від підготовки конфігурації до формування звіту, а механізми зберігання версій, параметрів запуску і результатів забезпечують відтворюваність і прозоре подання підсумків порівняння методів узгодження поведінки великих мовних моделей.

В подальшому розглянемо та опишемо, яке саме програмне забезпечення має реалізувати ці ідеї, для кого воно призначене та в яких межах працює.

## 2.3. Специфікація вимог до програмного забезпечення

### 2.3.1 Призначення та межі проекту

Призначення системи – є інформаційною системою підтримки прийняття рішень щодо вибору та оцінювання методів узгодження поведінки великих мовних моделей. Основне призначення системи полягає в тому, щоб надати користувачам єдиний інструмент, у якому можна ставити та вести експерименти з різними методами узгодження, оцінювати їх на заздалегідь підготовлених

наборах завдань, обчислювати метрики якості, безпеки, керованості, ефективності та вартості, а також на основі цих даних порівнювати альтернативи за багатьма критеріями і формувати обґрунтовану рекомендацію щодо вибору методу. Система покликана зробити процес оцінювання прозорим і відтворюваним: кожен експеримент описується конфігурацією, яка фіксує використовувані моделі, методи узгодження, набори даних, метрики та параметри запуску, а результати зберігаються таким чином, щоб будь-який запуск можна було повторити й перевірити.

Погодження, що ухвалені в програмній документації – прийнято низку принципових припущень, які задають рамки функціонування системи. Розглядаються лише текстові великі мовні моделі загального призначення, тобто моделі, що генерують відповіді послідовно, слово за словом; мультимодальні моделі та спеціалізовані агенти свідомо виключені, щоб звузити фокус і забезпечити однорідність результатів. Оцінювання здійснюється офлайн, на спеціально підготовлених датасетах із завданнями, які описують запити, очікувані відповіді або правила перевірки; онлайн A/B-експерименти з реальними користувачами в продакшені не є частиною даної системи. Джерелами даних вважаються публічні або дозволені для навчального використання набори, а також контрольовані внутрішні колекції; масовий новий збір людських розміток не планується, натомість застосовуються наявні ресурси. Метрики, такі як точність, частка токсичних відповідей, час відповіді чи орієнтовна вартість, беруться зі стандартних бенчмарків та узгоджених правил підрахунку. Автоматичне оцінювання за допомогою іншої моделі-«судді» допускається лише як допоміжний інструмент і супроводжується явним зазначенням його обмежень.

Межі проєкту програмного забезпечення – проєкт свідомо не ставить за мету створення нової базової архітектури мовної моделі чи розроблення оригінального алгоритму узгодження поведінки; система працює поверх уже наявних LLM-моделей і методів узгодження, які підключаються як зовнішні виконавці. Впровадження повного промислового контуру MLOps, включно з

автоматичним масштабуванням, складними сценаріями поетапного розгортання та низькорівневою оптимізацією під конкретне апаратне забезпечення, не входить до обсягу робіт. Не проводиться також формальна верифікація безпеки моделей та юридична експертиза політик; система забезпечує технічну підтримку оцінювання, але не підміняє собою юридичні чи регуляторні процеси. Важливим обмеженням є орієнтація на лабораторне, навчальне та пілотне промислове використання: система має допомагати дослідникам і інженерам підготувати обґрунтований вибір методів узгодження перед масштабним впровадженням у продакшені, але не претендує на роль готового масового SaaS-рішення для кінцевих користувачів.

### 2.3.2 Загальний опис

Сфера застосування системи – охоплює кілька типових контекстів. По-перше, це дослідницькі групи та лабораторії, де необхідно порівнювати різні методи узгодження поведінки моделей, працюючи з єдиними наборами завдань і прозорими протоколами оцінювання. По-друге, інженерні команди, які готують інтеграцію ВЛМ у прикладні сервіси, можуть використовувати систему для попереднього аналізу компромісів між якістю, безпекою, латентністю та витратами, перш ніж приймати рішення про впровадження. По-третє, освітні курси й навчальні лабораторії можуть застосовувати систему як навчальний стенд, що демонструє повний цикл роботи з методами узгодження: від постановки експерименту до інтерпретації результатів. Нарешті, система придатна для пілотних промислових впроваджень у тих випадках, коли організація хоче прозоро й у відтворюваний спосіб обрати метод узгодження з урахуванням конкретних обмежень середовища та політик.

Характеристики користувачів – системою користуються кілька груп учасників, які взаємодіють із нею по-різному, але спираються на спільну модель даних та єдині протоколи оцінювання. Дослідники та ML-інженери відповідають за постановку й параметризацію експериментів: вони обирають моделі й методи узгодження, підключають набори завдань, налаштовують метрики і запуск

оцінювань, а також аналізують детальні результати. Продакт-менеджери та власники продукту працюють переважно зі зведеними звітами і використовують систему як джерело прозорих, кількісно обґрунтованих рекомендацій щодо того, який метод доцільно впровадити у продукт. Фахівці з безпеки, етики та комплаєнсу розглядають систему як інструмент для перевірки відповідності моделей внутрішнім політикам і регуляторним вимогам, аналізуючи сценарії, пов’язані з небажаним чи упередженим контентом. Команди інфраструктури та DevOps інтегрують систему в існуюче середовище, налаштовують середовища розгортання, моніторинг і резервування, а також забезпечують доступ до зовнішніх LLM-провайдерів. Анотатори й перевіряльники якості, якщо вони залучені, використовують інтерфейс для ручної оцінки вибірок відповідей, доповнюючи автоматичні метрики людськими оцінками. Тестувальники та представники бізнес-підрозділів запускають типові сценарії й аналізують зрозумілі звіти, у яких результати подані у формі таблиць, графіків та текстових висновків, придатних для включення у внутрішню документацію

Загальна структура і склад системи – з погляду системної інженерії розроблюване програмне забезпечення являє собою клієнт-серверний комплекс із кількома взаємопов’язаними підсистемами. У центрі знаходиться доменне «ядро» прийняття рішень на основі багатокритеріального аналізу, в якому реалізовано конвеєр перетворення сирих значень метрик на впорядкований список альтернатив з урахуванням порогових обмежень, ваг критеріїв і чутливісного аналізу. Окремий модуль відповідає за обчислення метрик на основі сирих результатів експериментів, ще один за оркестрацію запусків, взаємодію з базою конфігурацій і зовнішніми виконавцями, якими виступають API мовних моделей або локальні обчислювальні воркери. Реєстри моделей, методів і датасетів забезпечують централізоване зберігання та версіонування описів усіх артефактів, а підсистема звітності формує різнорівневі звіти для різних категорій користувачів. Над цими доменними компонентами працює веб-інтерфейс, реалізований як односторінковий застосунок, що відображає основні функціональні області: роботу з наборами даних, конфігураціями, запусками та

звітами. У підґрунті системи лежить реляційна база даних, яка зберігає всі конфігурації, журнали, результати оцінювання та службові довідники, а також інфраструктурні компоненти для контейнеризації, моніторингу й CI/CD.

Загальні обмеження – система спроєктована для роботи в умовах низки обмежень, що випливають із характеру предметної області. Оцінювання завжди проводиться на заздалегідь визначених наборах завдань, тому результати неминуче залежать від репрезентативності цих датасетів; при цьому можливі систематичні перекоси, пов’язані з неповнотою даних або нерівномірним покриттям тем. Автоматичні методи оцінки, зокрема використання моделей-«суддів», можуть допускати помилки, тому для критичних сценаріїв передбачається ручна вибіркова перевірка. Ресурсні обмеження (доступні обчислювальні ресурси, квоти й тарифи постачальників моделей) впливають на глибину та масштаб експериментів, тому налаштування запусків мають фіксувати ліміти вартості й часу виконання. У рамках проєкту не реалізується повна підтримка довготривалих онлайн-експериментів, не розглядаються мультимодальні моделі й не виконується глибока інтеграція з усіма можливими виробничими системами, тому результати слід інтерпретувати як такі, що отримані у контрольованому, але все ж обмеженому середовищі.

### 2.3.3 Функції системи

Основні функції системи реалізують повний цикл роботи з методами узгодження від постановки експерименту до формування підсумкових рекомендацій. Функція управління проєктами та експериментами забезпечує створення й супровід записів проєктів, у межах яких групуються експерименти, а також ведення запусків із фіксацією конфігурацій, статусів і артефактів. Саме тут задаються зв’язки між моделями, методами узгодження, наборами завдань і обраними метриками, а кожен запуск сприймається як відтворювана одиниця, яку можна повторити, модифікувати чи порівняти з іншими.

Функція керування моделями, методами й наборами даних відповідає за підтримку узгоджених довідників, у яких описано, які саме моделі й методи

доступні для оцінювання, яким чином до них здійснюється доступ, які версії застосовуються, а також які датасети можна використовувати в експериментах. Кожен елемент має набір атрибутів, що дозволяє системі з одного боку перевіряти коректність конфігурацій, а з іншого будувати звіти, де результати чітко пов’язуються з конкретними версіями моделей, методів та наборів завдань.

Функція запуску сценаріїв оцінювання реалізується у вигляді оркестраторів, які перетворюють описану конфігурацію в серію конкретних викликів LLM-моделей чи інших сервісів. Модуль ExperimentRunner організовує взаємодію між базою даних конфігурацій і зовнішніми виконавцями за шаблоном «виробник–споживач»: завдання на оцінювання потрапляють у чергу, обробляються фоновими воркерами без блокування основного веб-інтерфейсу, а результати й журнали виконання зберігаються для подальшого аналізу. Це дозволяє масштабувати експерименти, обробляючи великі набори завдань, зберігаючи при цьому чутливість інтерфейсу користувача і керованість процесу.

Функція обчислення метрик і багатокритеріального аналізу зосереджена в спеціалізованих модулях, що перетворюють сирі відповіді моделей та еталонні дані на числові показники, а потім — на впорядкований список альтернатив. Спочатку здійснюється попередня фільтрація: кандидати, які не відповідають жорстким порогам безпеки або формату (наприклад, надто висока частка токсичних відповідей), відсікаються ще до етапу агрегації. Далі значення метрик нормалізуються, до них застосовуються вагові коефіцієнти, а обраний метод багатокритеріального аналізу, наприклад зважена сума, формує інтегральну оцінку. Завдяки модульній реалізації стратегії підрахунку балів можна замінювати або розширювати, додаючи інші MCDA-підходи на кшталт TOPSIS без переробки решти системи, що підтримує еволюційний розвиток рішення.

Функція формування звітів забезпечує перетворення результатів обчислень на зрозумілі для різних ролей представлення. Для дослідників надаються детальні таблиці із значеннями метрик, журналами запусків і параметрами конфігурацій; для менеджерів — узагальнені порівняльні огляди, графічні візуалізації та короткі текстові висновки щодо рекомендацій; для

фахівців із безпеки — спеціальні звіти, що зосереджені на показниках токсичності, упередженості та відповідності політикам. Звіти можуть експортуватися у форматах, придатних для включення в документацію, презентації чи внутрішні звіти, що спрощує використання результатів за межами самої системи.

### 2.3.4 Вимоги до інформаційного забезпечення

Інформаційне забезпечення системи включає як вхідні дані, так і нормативно-довідкову інформацію та організацію зберігання. Джерелами вхідної інформації є, по-перше, датасети для оцінювання, які можуть бути публічними або спеціально підготовленими внутрішніми наборами; вони містять запити, очікувані відповіді, правила перевірки й завдання для тестування безпеки. По-друге, конфігурації експериментів, що визначають, які моделі, методи узгодження, метрики та параметри запускаються в кожному конкретному випадку. По-третє, результати запусків, до яких належать відповіді моделей, журнали, службові показники на кшталт часу виконання та використаних токенів. Метрики, що обчислюються системою, спираються на узгоджені визначення, запозичені зі стандартних бенчмарків, а автоматичне оцінювання моделями-«суддями» супроводжується збереженням додаткових атрибутів, які дають змогу аналізувати можливі похибки.

Нормативно-довідковий шар включає внутрішні політики безпеки та етики, зовнішні рекомендації щодо роботи з ВЛМ, класифікації типів моделей і методів узгодження, перелік метрик та їх формальних визначень, а також опис ролей користувачів і рівнів доступу. Вся ця інформація використовується при побудові конфігурацій, інтерпретації результатів і формуванні звітів. Збереження інформації організовано в реляційній базі даних; для всіх ключових сутностей підтримується версіонування, що дозволяє простежити історію змін і гарантує відтворюваність експериментів. Конфігурації, структури даних, результати оцінювань та артефакти тестування доповнюються журналами подій,

що формують аудиторський слід, необхідний для відстеження того, які саме умови були використані при отриманні конкретних висновків.

### 2.3.5 Вимоги до технічного забезпечення

Технічне забезпечення системи передбачає серверне середовище, здатне виконувати веб-застосунок, фонові воркери й базу даних, а також клієнтські пристрої користувачів. На сервері розгортається контейнеризований стек, що включає серверну частину на .NET 8, базу даних PostgreSQL та допоміжні служби, упаковані у версіоновані Docker-образи; інфраструктура організована у три ізольовані середовища — Development, Staging і Production, кожне з яких має власні параметри конфігурації, логування та моніторингу. Це дозволяє відпрацьовувати зміни поступово й забезпечувати стабільність продакшн-середовища. Клієнтська частина працює в сучасному веб-браузері і не потребує спеціалізованого апаратного забезпечення, що робить систему доступною для широкого кола користувачів. За потреби система може використовувати апаратні прискорювачі або зовнішні LLM-API, однак мінімальною умовою є наявність обчислювальних ресурсів, достатніх для виконання веб-застосунку та обробки експериментів у заданих межах.

### 2.3.6 Вимоги до програмного забезпечення

Архітектура програмної системи побудована у вигляді модульного моноліту з використанням підходу «порти й адаптери». Серверна частина реалізована на платформі .NET 8, фронтенд — як SPA на базі React і TypeScript, для обчислення спеціалізованих метрик передбачено опційний Python-модуль. Така декомпозиція дозволяє ізолювати доменну логіку багатокритеріального аналізу, оркестрації експериментів та обчислення метрик від деталей інфраструктури, забезпечуючи можливість тестування й подальшої еволюції системи без серйозної перебудови її основи.

У ролі системного програмного забезпечення використовуються операційна система сімейства Linux або Windows Server для серверної частини,

середовище виконання .NET 8 для бекенду та PostgreSQL як основна реляційна система керування базами даних. Контейнеризація компонентів виконується за допомогою Docker, що полегшує розгортання й оновлення системи в різних середовищах, а також підтримує відтворюваність конфігурацій.

Для мережної взаємодії застосовується веб-сервер ASP.NET Core, який обробляє HTTP(S)-запити від клієнтів, а також набір конфігурацій для інтеграції із зовнішніми API моделей. Усі зовнішні підключення здійснюються через зашифрований транспорт, а налаштування адрес, токенів доступу та параметрів тайм-аутів зберігаються в конфігураційних файлах і змінних середовища, не потрапляючи безпосередньо в кодову базу.

Доступ до даних реалізовано за допомогою ORM-фреймворку Entity Framework Core у поєднанні з Dapper для оптимізованих SQL-запитів. Такий підхід дозволяє поєднати зручність роботи з доменними моделями та контроль над продуктивністю в критичних місцях. Міграції схеми бази даних виконуються автоматизовано в рамках CI/CD-конвеєрів; для тестових середовищ використовується запуск тимчасових контейнерів з PostgreSQL, щоб перевіряти коректність запитів і обмежень цілісності без ризику для робочих даних.

Основною мовою серверної частини є C# у середовищі .NET 8; клієнтська частина реалізована мовами TypeScript і JavaScript з використанням фреймворку React. Для організації архітектурних патернів застосовуються бібліотеки MediatR, FluentValidation, Polly та інші засоби, що забезпечують розділення відповідальностей, валідацію даних і підвищення стійкості до збоїв. Контейнеризація й розгортання базуються на Docker і Docker Compose, а конвеєри CI/CD автоматизують усі етапи перевірки й публікації артефактів.

### 2.3.7 Вимоги до зовнішніх інтерфейсів

Інтерфейс користувача реалізовано як односторінковий веб-застосунок, у якому окремі екрани відповідають ключовим доменним областям: управлінню наборами даних, конфігураціями оцінювання, запуском експериментів, переглядом їхнього прогресу та детальних звітів. Навігація організована таким

чином, щоб ML-інженер міг швидко перейти від створення конфігурації до запуску й аналізу результатів, а менеджер — без зайвих деталей побачити агреговані висновки. Інтерфейс враховує потреби різних ролей, але залишається єдиним веб-додатком, доступ до функцій у якому регулюється засобами автентифікації та авторизації.

Спеціалізовані апаратні інтерфейси не передбачаються. Система взаємодіє з апаратним забезпеченням опосередковано — через операційну систему та інфраструктурні сервіси; для її роботи достатньо стандартних серверів і персональних комп'ютерів користувачів із доступом до мережі.

Система надає програмний інтерфейс у вигляді REST-API, описаного за допомогою OpenAPI-специфікації. Це дозволяє генерувати типізованих клієнтів для інтеграції з іншими застосунками та автоматизованими сценаріями, наприклад запускати оцінювання з CI/CD-конвеєрів або експортувати результати у зовнішні аналітичні системи. Крім того, реалізовано адаптери до зовнішніх LLM-провайдерів, які інкапсулюють деталі автентифікації, формату запитів і обробки помилок, залишаючи для доменної логіки уніфікований інтерфейс.

Взаємодія між клієнтом і сервером, а також між сервером і зовнішніми сервісами здійснюється поверх протоколу HTTP з використанням шифрування TLS. Конфігурації мережних підключень зберігаються в окремих файлах і змінних середовища, що дає змогу відокремити параметри розгортання від коду й налаштовувати різні середовища без перекомпіляції.

### 2.3.8 Властивості програмного забезпечення

Доступність – система розгортається в кількох середовищах з використанням контейнеризації та автоматизованих конвеєрів розгортання, що дозволяє зменшити простої та швидко відновлювати працездатність у разі збоїв. Використання перевірок здоров'я сервісів і дашбордів моніторингу дає змогу оперативно виявляти проблеми й підтримувати доступність для користувачів у навчальних і пілотних сценаріях.

Супроводжуваність – модульний моноліт із чітким розділенням на доменні служби, адаптери й інфраструктурний код, а також використання шаблонів на кшталт «порти й адаптери» та Mediator, забезпечують хорошу структурованість і передбачувану еволюцію системи. Monorepo-підхід спрощує координацію змін між бекендом, фронтендом і інфраструктурою, а повноцінний набір автоматизованих тестів (юніт-, інтеграційних, наскрізних) знижує ризик регресій при доопрацюваннях.

Переносимість – завдяки контейнеризації всі компоненти системи упаковано в Docker-образи, які можна запускати в різних середовищах — від локальних стендів до хмарної інфраструктури. Налаштування конкретного оточення виносяться у файли конфігурації та змінні середовища, що дозволяє переносити систему між серверами без зміни коду.

Продуктивність – архітектура з окремими фоновими воркерами та шаблоном «виробник–споживач» для виконання експериментів дає змогу уникати блокування основного веб-потоку і забезпечувати прийнятний час відгуку інтерфейсу навіть за великої кількості завдань. Водночас збереження метрик часу виконання, кількості токенів і витрат дозволяє користувачам оцінювати продуктивність системи та приймати рішення з урахуванням реальних ресурсних обмежень.

Надійність – забезпечується поєднанням архітектурних рішень і інфраструктурних практик: використанням реальної СУБД PostgreSQL у тестах, автоматизованими міграціями й можливістю відкату, розділенням середовищ Dev/Stage/Prod, а також впровадженням стратегій blue/green-розгортання й smoke-тестів перед оновленням продакшн-середовища. Така організація мінімізує ризик «дрейфу конфігурацій» і дозволяє впевнено розгортати нові версії системи без порушення її цілісності.

Безпека – передбачає автентифікацію та авторизацію користувачів, безпечне поводження із секретами (ключами доступу, рядками підключення), шифрування мережного трафіку і контроль доступу до конфігурацій та результатів експериментів. Секретні значення не зберігаються в репозиторії, а

передаються через змінні середовища, що відповідає рекомендаціям «дванадцятифакторного застосунку». Журнали подій і результати оцінювань формують аудиторський слід, необхідний для аналізу інцидентів і підтвердження коректності рішень, прийнятих на основі роботи системи.

### 2.3.9 Інші вимоги

До інших вимог належать дотримання ліцензійних обмежень на використання моделей і даних, наявність користувацької та технічної документації, що описує порядок налаштування, запуску оцінювань і інтерпретації результатів, а також можливість подальшого розширення системи новими методами узгодження, типами метрик і інтеграціями з іншими сервісами. Окремо підкреслюється важливість етичного поводження з даними, включно з можливістю анонімізації чутливих записів і прозорим описом обмежень отриманих висновків. Сукупність цих вимог доповнює функціональний аспект специфікації, забезпечуючи відповідний рівень якості програмного забезпечення та його готовність до відповідального використання в дослідницьких, навчальних і пілотних промислових сценаріях.

## Висновки до Розділу 2.

У другому розділі сформовано цілісний опис майбутньої інформаційної системи: від концептуальної та інформаційної моделі до формальної моделі багатокритеріального вибору й специфікації вимог до програмного забезпечення. Це дозволило перейти від загальної постановки задачі до чітко структурованого проєктного рішення, придатного до практичної реалізації.

По–перше, у розділі уточнено концептуальну модель системи, де ядром виступає експеримент узгодження, що поєднує модель, метод узгодження, набори даних і конфігурацію оцінювання та породжує запуски (runs). Виділено основні сутності (проєкти, експерименти, запуски, моделі, методи, датасети, конфігурації, метрики, звіти, користувачі/ролі) та показано, як така структура

забезпечує відтворюваність експериментів і розподіл відповідальності між різними ролями.

По–друге, побудовано інформаційну модель у вигляді реляційної схеми (Projects, Experiments, Runs, Models, ОцінюванняMethods (), Datasets, EvalConfigs, MetricDefinitions, MetricValues, Reports, Users, Roles, проміжні таблиці). Нормалізована структура та зовнішні ключі дозволяють уникати дублювання, гнучко додавати нові типи метрик, методів і моделей, а також будувати потрібні аналітичні запити й звіти без перебудови всієї бази даних.

По–третє, описано життєвий цикл експерименту та політику роботи з даними: від створення проєкту й експерименту до запусків, формування звітів і архівації. Запроваджено принципи версіонування суттєвих об’єктів, відокремлення сирих даних від агрегованих метрик, рольового контролю доступу та ведення журналу дій, а також архівації застарілих експериментів при збереженні підсумкових показників. Це забезпечує відтворюваність, прозорість та дотримання вимог до безпеки й приватності.

По–четверте, формалізовано модель багатокритеріального вибору на основі методу зваженої суми. Метрики якості, безпеки, часу відповіді, вартості тощо нормалізуються до [0;1] з урахуванням типу критерію («більше/менше – краще»), далі агрегуються з використанням ваг, узгоджених зі стейкхолдерами. Визначено двокрокову схему вибору: спочатку фільтр обов’язкових порогів (санітарний мінімум), потім ранжування допустимих методів за підсумковим балом, із можливістю показу недомінованих альтернатив у стилі Парето.

По–п’яте, окреслено протокол оцінювання та чутливісний аналіз. Єдина послідовність кроків включає фіксацію конфігурації (метрики, пороги, ваги, датасети), запуск прогонів, обчислення метрик, застосування порогів, побудову рейтингу та формування звітів. Передбачено перевірку стійкості рекомендацій до зміни ваг критеріїв і до статистичної варіативності метрик (повторні запуски, довірчі інтервали), що дозволяє відрізняти «випадкових» лідерів від стабільно кращих методів.

По–шосте, визначено функціональні вимоги та сценарії використання з огляду на ролі користувачів. Система підтримує керування проєктами та експериментами, довідниками моделей/методів/датасетів/метрик, запуск і моніторинг прогонів, автоматичне обчислення й перегляд метрик, застосування багатокритеріальної моделі, формування та збереження звітів, а також керування користувачами й ролями. Інженери відповідають за технічну частину експериментів, аналітики – за інтерпретацію компромісів, фахівці з безпеки – за контроль порогів, адміністратори – за доступ і працездатність системи.

По–сьоме, сформульовано вимоги до інформаційного, технічного та програмного забезпечення й нефункціональні властивості. Система орієнтована на лабораторне/пілотне розгортання (один сервер або робоча станція із СУБД та вебдодатком), клієнтський доступ через браузер і взаємодію з мовними моделями через API (HTTP/HTTPS, JSON). Задано вимоги до доступності, надійності (транзакційність, резервне копіювання), продуктивності інтерфейсу, супроводжуваності (модульна архітектура, документовані інтерфейси), переносимості, а також аудиторського сліду, ліцензійної коректності й етичної обробки даних.

У підсумку розділ 2 реалізує три ключові завдання роботи:

– розроблено концептуальну та інформаційну модель системи;

– сформовано формальну модель багатокритеріального вибору й методику оцінювання;

– визначено специфікацію вимог до програмного забезпечення з урахуванням ролей користувачів, сценаріїв використання та нефункціональних обмежень. Це створює повноцінний проєктний каркас для подальшої архітектурної деталізації та програмної реалізації системи.

# 3. АРХІТЕКТУРА І ПРОЄКТУВАННЯ

## 3.1. Архітектурне рішення і компоненти

Для формального опису поведінки системи підтримки вибору та оцінювання методів узгодження поведінки великих мовних моделей можна використати кілька типів UML–діаграм. Вони доповнюють архітектурний опис і фіксують взаємодію користувачів із системою, обмін даними між компонентами та структуру доменної моделі.

На діаграмі варіантів використання (Use Case) показано основних акторів: адміністратора, дослідника/аналітика, оглядача результатів, фонового виконавця (воркера) та зовнішнього постачальника мовних моделей (API). Для них виділено ключові сценарії: керування наборами даних, створення конфігурацій оцінювання, запуск експериментів, моніторинг прогресу й журналів, перегляд метрик, запуск багатокритеріального ранжування, формування та експорт звітів, керування користувачами й ролями. Зв’язки include/extend відображають, зокрема, що запуск експерименту включає перевірку порогових умов і збереження артефактів.

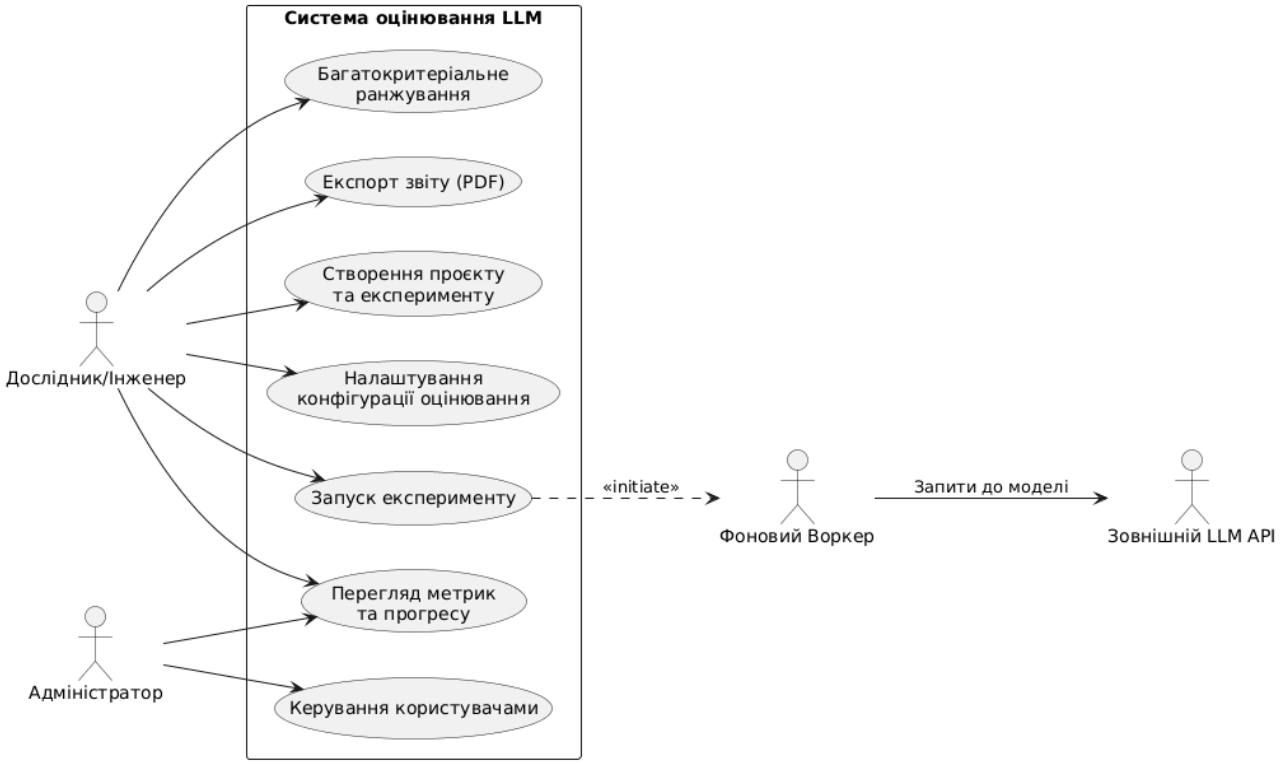

Рисунок 3.1. Діаграма варіантів використання (Use Case) системи

У рамках системи передбачається кілька типів користувачів (ролей) які по суті відповідають розглядуваним раніше сутностям User:

Інженер / дослідник – створює проєкти та експерименти, обирає моделі, методи узгодження та набори даних, налаштовує конфігурації оцінювання, запускає експерименти й аналізує детальні результати.

Аналітик або продакт–менеджер – переглядає зведені звіти, порівнює методи за ключовими показниками, інтерпретує компроміси (якість, безпека, швидкодія, вартість) та готує управлінські рекомендації.

Фахівець із безпеки / комплаєнсу – перевіряє результати тестів безпеки, контролює виконання обов'язкових порогів, за потреби уточнює вимоги до наборів завдань і політик.

Адміністратор системи – налаштовує середовище, керує користувачами та їхніми ролями, відповідає за резервне копіювання та оновлення системи.

Таке розмежування ролей дозволяє розподілити відповідальність: інженери займаються технічною частиною експериментів, аналітики й стейкхолдери – інтерпретацією результатів і прийняттям рішень, спеціалісти з безпеки – контролем ризиків, а адміністратори – підтримкою працездатності системи.

Для конкретизації призначення системи далі опишемо, які саме дії вона має підтримувати з точки зору користувача.

Основні функції системи можна поділити на кілька груп, що відповідають етапам роботи з експериментами та прийняття рішень.

По–перше, система забезпечує керування проєктами та експериментами. Користувач може створювати, редагувати та переглядати проєкти, в межах яких формуються експерименти з узгодження поведінки. Для кожного експерименту система дозволяє задавати мету, опис, обирати модель, метод узгодження, набори даних і конфігурацію оцінювання, а також змінювати статус (запланований, у процесі, завершений, архівний).

По–друге, реалізується керування довідниками. До них належать: перелік доступних мовних моделей, методів узгодження, наборів даних і метрик, тощо. Система повинна дозволяти додавати нові записи до цих довідників, оновлювати їхні описи й версії, а також позначати записи як неактуальні без фізичного видалення, щоб не порушувати цілісність історичних експериментів.

По–третє, система виконує функції запуску та моніторингу експериментів. Користувач може ініціювати запуск експерименту або окремого прогону (run), вказавши необхідні параметри середовища. Під час виконання система має збирати службову інформацію (час запуску, технічне оточення, стан виконання) та надавати користувачеві базовий моніторинг стану (очікує, виконується, завершено, помилка).

По–четверте, система відповідає за обчислення, зберігання та перегляд метрик. Після виконання прогону вона автоматично обчислює значення метрик, визначених у конфігурації оцінювання (точність, частка небезпечних відповідей, час відповіді, дотримання формату тощо), зберігає їх у базі даних і надає зручний інтерфейс для перегляду цих значень у розрізі експериментів, запусків і наборів даних.

По–п’яте, система реалізує підтримку багатокритеріального вибору. На основі збережених метрик вона повинна вміти застосувати заздалегідь задані пороги, виконати нормалізацію показників, розрахувати підсумкові бали для кожного методу узгодження, сформувати рейтинг альтернатив та, за потреби, відобразити результати для кількох наборів ваг критеріїв (сценаріїв). Результати цього обчислення мають бути доступні як у вигляді числової таблиці, так і у вигляді коротких текстових висновків.

По–шосте, система забезпечує формування та збереження звітів. Користувач може створити звіт за результатами одного або кількох експериментів;до такого звіту входять таблиці метрик, зведені оцінки, ранжування методів та словесний опис основних висновків. Система має зберігати створені звіти, дозволяти їх повторний перегляд та, за можливості, експорт (наприклад, у форматі PDF або іншому зручному форматі).

Окремою функціональною групою є керування користувачами та ролями. Адміністратор системи повинен мати можливість створювати облікові записи, призначати ролі (інженер, аналітик, фахівець із безпеки, адміністратор) та обмежувати доступ до окремих функцій і проєктів відповідно до цих ролей. Для ключових операцій (створення, зміна та видалення експериментів, запусків, конфігурацій) доцільно передбачити ведення журналу дій.

Нарешті, система має підтримувати інтеграцію з джерелами моделей. Це може бути виклик зовнішнього API або підключення до локально розгорнутих моделей через стандартизований програмний інтерфейс. Мінімальною вимогою є можливість зберегти параметри підключення та виконати запити згідно з описаним протоколом експериментів, не прив’язуючи систему до конкретного постачальника. У сукупності зазначені функції забезпечують повний цикл роботи користувача – від постановки експериментів до отримання обґрунтованого рішення щодо вибору методу узгодження поведінки.

### 3.2. Моделі та алгоритми

Діаграма послідовності (Sequence)– для сценарію «Запуск експерименту та збір метрик» показує порядок викликів між вебінтерфейсом, бекендом/API, сервісом автентифікації, службою запуску експериментів, конектором до мовної моделі, модулем обчислення метрик, сховищем артефактів і базою даних. Послідовність охоплює створення запису про запуск, постановку завдань у чергу, пакетну обробку запитів, виклики до моделі, збереження «сирих» відповідей і журналів, обчислення первинних метрик та фіксацію підсумкового статусу.

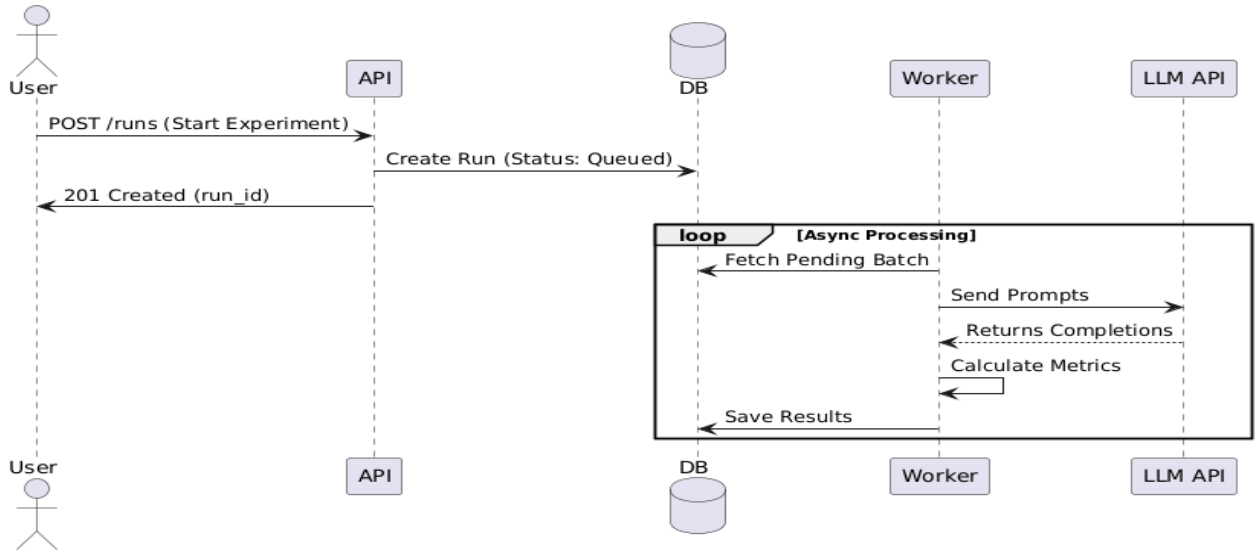


Рисунок 3.2. Діаграма послідовності виконання експерименту (Sequence) системи

Діаграма діяльності (Activity) формалізує сценарій оцінювання одного кандидата з урахуванням порогів. Вона описує підготовку підмножини даних, генерацію відповідей, перевірку формату, обчислення метрик якості й безпеки, порівняння з обов’язковими порогами, маркування кандидата як прийнятного чи відхиленого та агрегування результатів за всіма завданнями. Контрольні точки на діаграмі позначають місця, від яких можливе відновлення виконання після збоїв.


Рисунок 3.3. Діаграма діяльності (Activity) системи

Статичну структуру доменної моделі задає діаграма класів (Class/ER). На ній виділено основні сутності: набір даних/бенчмарк, конфігурацію оцінювання (метрики, пороги, параметри генерації), експеримент, окремий запуск, результати метрик, кандидата–підхід (метод узгодження), матрицю рішень, вектор ваг, результат ранжування, звіт, а також користувачів, ролі та права доступу. Показано, що один експеримент містить кілька запусків;кожен запуск пов’язаний із конкретною конфігурацією та одним чи кількома наборами даних;результати метрик агрегуються в матрицю рішень, на основі якої формується рейтинг альтернатив

На діаграмі компонентів (Component) позначено основні програмні модулі й інтерфейси: бекенд на .NET з наборами API для роботи з даними, експериментами, метриками, багатокритеріальною оцінкою та звітами;сервіс автентифікації й авторизації;фоновий виконавець запусків;конектори до зовнішніх мовних моделей;модулі обчислення метрик і MCDA;сервіс формування звітів;реляційну базу даних та файлове/об’єктне сховище артефактів. На інтерфейсах зафіксовано формати обміну (JSON) і використання версіонованих контрактів. Оскільки вона визначає фундаментальну архітектуру системи, її зображення та детальний опис наведено в попередньому підрозділі (див. рис. 3.1).

Діаграма розгортання (Deployment) відображає розміщення компонентів на вузлах у середовищах Dev, Stage і Prod: контейнери з бекендом, окремі екземпляри БД та воркера, реверс–проксі з підтримкою TLS, допоміжні сервіси моніторингу й логування. На каналах зв’язку позначено протоколи (HTTPS, gRPC, підключення до БД) і мережеві обмеження, важливі з погляду безпеки.

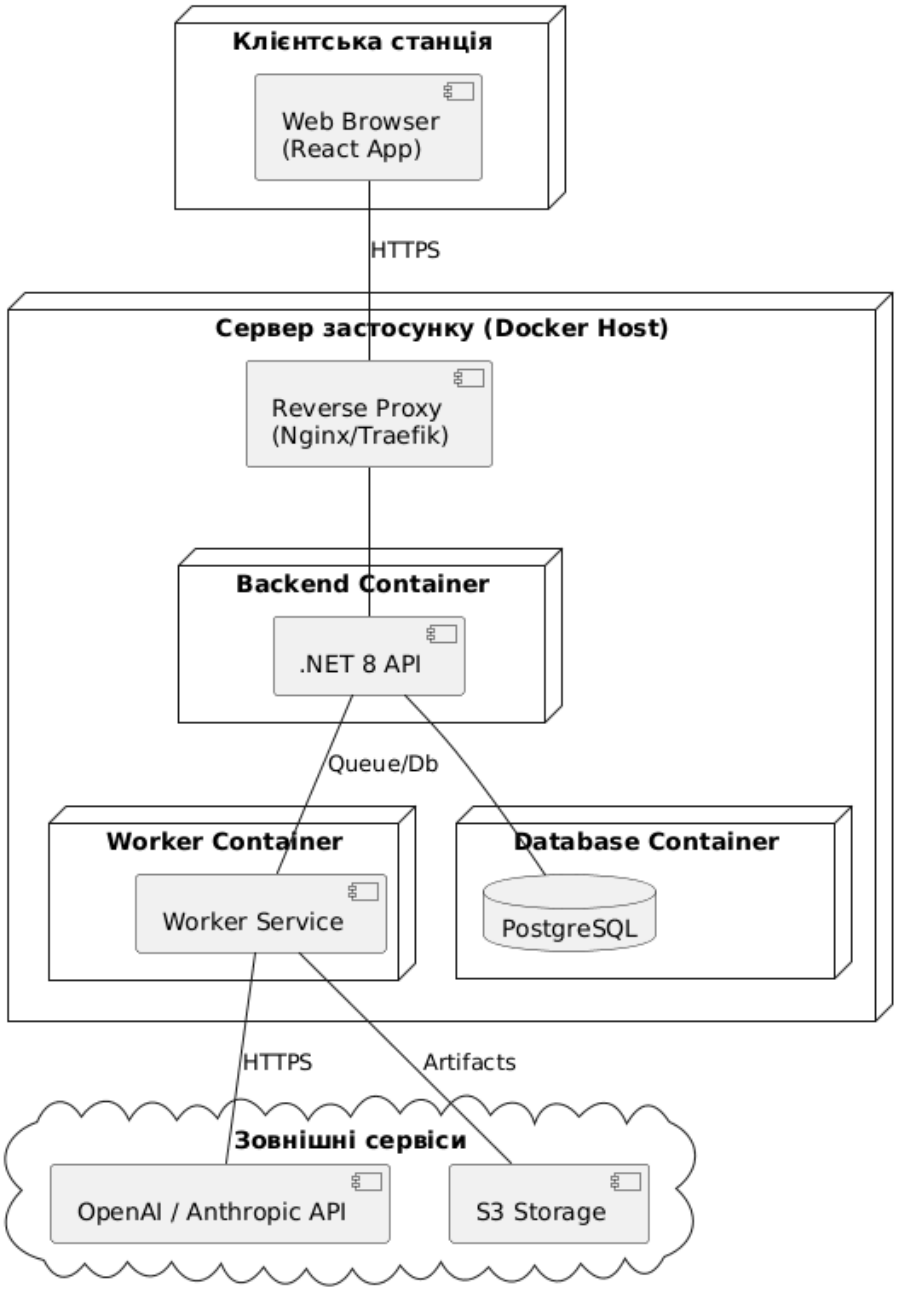


Рисунок 3.4. Діаграма розгортання (Deployment) системи

Алгоритми ядра визначають, як система організовує прогін оцінювання, готує дані для порівняння кандидатів (методів узгодження) і формує підсумкову рекомендацію.

Перед запуском задаються відтворювані вхідні умови: конфігурація оцінювання (перелік метрик, пороги, параметри генерації, тайм–аути), перелік кандидатів, набори завдань із фіксованими версіями, а також обмеження на

паралелізм і ліміти звернень до зовнішніх API. Ці налаштування зберігаються разом із результатами й дають змогу повторити експеримент у тих самих умовах.

Прогін починається зі створення запису з унікальним ідентифікатором, початковим статусом, прив'язкою до версій наборів завдань і конфігурації, а також фіксацією «зерна випадковості». Набори завдань діляться на батчі, для яких формується черга робіт. Фонові виконавці обробляють їх паралельно, дотримуючись заданих лімітів запитів до моделей. Для кожної порції генеруються відповіді, «сирі» результати потрапляють до сховища артефактів, зберігаються часові мітки, виконуються первинні перевірки формату та безпеки, обчислюються базові метрики (якість, час відповіді тощо). Часткові підсумки записуються в базу, оновлюється прогрес і створюються контрольні точки для відновлення після збоїв. Після обробки всіх батчів метрики агрегуються по наборах і кандидатах, а статус прогону змінюється на «завершено» або «помилка» з посиланнями на журнали.

Далі застосовується обов'язковий фільтр порогів. До подальшого порівняння допускаються лише ті кандидати, які виконують мінімальні вимоги щодо безпеки, формату та граничного часу відповіді. Порушення будь–якого порога призводить до відсіву кандидата з фіксацією причин у результатах і звіті. Для тих, хто пройшов фільтр, формується матриця рішень: по кожному кандидату обчислюються узгоджені показники (якість, безпека, дотримання формату, надійність, ефективність), а за наявності кількох тестових наборів спершу виконується агрегування в межах наборів, потім – між ними.

Оскільки метрики різнорідні, їх приводять до спільної шкали. Базовим методом є мін–макс нормалізація до [0;1] із відмінністю для метрик типу «більше краще» та «менше краще». За потреби враховуються екстремальні значення (обрізання «хвостів» або z–нормалізація). Обраний метод і параметри нормалізації зберігаються разом із результатами.

Ранжування альтернатив виконується насамперед за зваженою сумою: нормалізовані значення множаться на вектор ваг критеріїв (узгоджених із замовником, сума ваг дорівнює одиниці), після чого підсумовуються. Так

отримуються підсумкові бали й рейтинг кандидатів. Для перевірки стійкості висновку може додатково застосовуватися альтернативний метод (наприклад, TOPSIS) і перевірятися збіг принаймні лідера.

Далі виконується чутливісний аналіз: ваги критеріїв варіюють у розумних межах навколо базових значень, повторно обчислюють бали та фіксують, чи зберігається перше місце за різних наборів ваг. Для кількісної оцінки може використовуватися ранговий коефіцієнт кореляції, який показує, наскільки стабільним залишається порядок кандидатів.

Алгоритми обробляють також збої й обмеження зовнішніх сервісів. Помилки, тайм–аути та перевищення квот обробляються політиками повторних спроб із наростаючою затримкою;проблемні приклади маркуються для ручного перегляду. Операції проєктуються як ідемпотентні: повторний запуск прогону з тим самим ідентифікатором не дублює результати, а за наявності контрольних точок дозволяє продовжити обробку, а не починати її з нуля.

Підсумкові результати зберігаються у формі, придатній для перевірки й повторення: фіксуються версії наборів завдань і конфігурації, спосіб нормалізації, правило ранжування, використані ваги, параметри чутливісного аналізу, а також посилання на журнали й артефакти. Звіт містить нормалізовані метрики, бали, місця кандидатів і короткий опис основних компромісів (де підхід якісніший, безпечніший чи швидший), а також окремо позначає рішення, що не пройшли пороги, із поясненням причин.

Таким чином, ядро алгоритмів охоплює повний цикл прийняття рішень: від підготовки відтворюваних умов і чесного вимірювання до відбору, ранжування, перевірки стійкості та прозорого звітування.

Підсистема обробки помилок і журналювання подій має три основні цілі: зберегти стабільність роботи (щоб окремий збій не зупиняв увесь прогін), зробити події прозорими (щоб можна було простежити ланцюжок дій і відтворити прийняте рішення) та забезпечити відповідальність за зміни (чітко розуміти, хто, коли і що саме змінив).

Система розрізняє кілька класів помилок:

1. валідаційні (некоректний JSON, відсутні пороги, помилкові параметри) – користувач отримує зрозуміле повідомлення та підказку, що виправити;

2. помилки бізнес–правил (непройдені пороги безпеки, формату, часу) – прогін коректно завершується зі статусом «відхилено» з фіксацією причин;

3. інтеграційні/мережеві (тайм–аути, вичерпані ліміти, недоступність сервісів) – застосовуються повторні спроби з паузами;проблемні приклади маркуються для повторної обробки;

4. системні/невідомі винятки – батч завершується безпечно, контекст помилки потрапляє в лог, відповідальні особи отримують сповіщення.

Для підвищення стійкості кожен батч виконується з тайм–аутами та політикою повторних спроб із експоненційною затримкою й джиттером. Для нестабільних джерел застосовується «запобіжник» (circuit breaker), що тимчасово блокує виклики. Записи в БД робляться ідемпотентно (ключ run_id + batch_id + item_id), після кожного батчу фіксується контрольна точка. Якщо відповідь моделі не відповідає формату, артефакт зберігається окремо, а приклад відмічається для ручної перевірки чи повторного запуску.

Усі сервіси ведуть структуровані логи (наприклад, JSON) з обов’язковими полями: timestamp (UTC), level, service, env, request_id, run_id, user_id/role (за наявності), action, status, duration_ms, error_code, коротке message, details, а також dataset_version і config_version. Це дозволяє корелювати події між бекендом, воркером, БД і конекторами. Секрети (API–ключі, токени, ПД) перед записом маскуються, логи ротуються й зберігаються за визначеною політикою, для «шумних» подій застосовується семплінг. На панелі моніторингу виводяться графіки помилок, затримок, частки успішних батчів, використання квот.

Кожному запиту призначається request_id, який проходить через усі сервіси. За потреби підтримується трасування, сумісне з OpenTelemetry (trace/span–ідентифікатори), що дозволяє побачити повний ланцюжок викликів і вузькі місця. Пороги сповіщень налаштовують так, щоб виявляти аномальний ріст тайм–аутів, відмов, помилок формату та вичерпання лімітів;у таких випадках адміністратор отримує алерт.

Аудит відповідає на запитання «хто, коли і що змінив». Для цього використовується таблиця audit_log у форматі append–only, де фіксуються: user_id і роль, час події, тип об’єкта (конфігурація, набір даних, запуск, користувач/роль, звіт), тип дії (створення, зміна, видалення, запуск, доступ), короткий опис, посилання на попереднє й нове значення, а також технічні атрибути (IP, user–agent). У вміст «сирих» відповідей моделі аудит не заглядає – лише посилання на артефакт. Доступ до audit_log обмежений (адмін, відповідальний за якість), записи можуть підписуватися хеш–сумами й зберігатися не менше встановленого строку із можливістю експорту для зовнішнього аудиту.

Комунікація з користувачем у разі помилок організована так, щоб повідомлення були зрозумілими, але без розкриття внутрішніх деталей. Інтерфейс показує, що сталося і які кроки варто зробити, а технічна інформація зберігається в логах. Для кожного прогону є сторінка статусу з прогресом, кількістю повторних спроб, посиланнями на журнали та короткими рекомендаціями (наприклад, зменшити паралельність чи збільшити тайм–аут).

Сервіси підтримують самодіагностику: реалізовані health/readiness–перевірки й сторінка самоперевірки інтеграцій (доступність БД, сховищ, коректність ключів API). План відновлення після збоїв описано як послідовність: виявлення події —> фіксація в логах і аудиті —> тимчасове призупинення нових прогонів —> за потреби відкат останніх змін —> відновлення з контрольної точки —> підготовка короткого звіту з причинами та запобіжними заходами, узгодженими з RTO/RPO з розділу 3.1.

Завдяки цим механізмам кожен інцидент можна локалізувати й пояснити, кожен висновок – відтворити, а всі зміни в конфігураціях і даних – відстежити, що є критично важливим для довіри до результатів оцінювання методів узгодження поведінки LLM.

### 3.3. Дані, програмні інтерфейси та інтерфейс користувача

Щоб забезпечити відтворюваність експериментів, прозорий підрахунок метрик і коректне формування рекомендацій, база даних організована навколо чотирьох логічних блоків: довідникових сутностей (моделі, набори даних, користувачі/ролі), конфігурацій оцінювань, запусків та результатів, а також аудиту й артефактів. Далі наведено змістовний опис основних таблиць, зв’язків і ключових обмежень (орієнтація на PostgreSQL).

Структура відображена раніше на ER–діаграмі а нижче наведено детальніше скорочений опис основних сутностей.

Розглянемо довідники та ідентифікацію. Таблиця users зберігає облікові записи (email, ПІБ, статус), roles – доступні ролі (admin, engineer, analyst, viewer тощо). Багато–до–багатьох зв’язок реалізується через user_roles, що фактично задає RBAC–модель.

Постачальники моделей описуються у providers, конкретні моделі – в models (посилання на provider_id, назва, версія/label, розмір контекстного вікна, примітки щодо обмежень).

Набори даних описані на двох рівнях: datasets (назва, домен, джерело, ліцензія) та dataset_versions (dataset_id, version, checksum, опис). У експериментах завжди використовуються саме версії, що забезпечує відтворюваність. Окремі завдання зберігає таблиця tasks (dataset_version_id, вхідний текст, еталон/правило перевірки, тег ризику/теми, службові мітки).

Другий блок описує конфігурації оцінювання та кандидати.

Таблиця evaluation_configs визначає, «як вимірюється» поведінка моделей: перелік метрик, пороги, параметри генерації, політику паралельності й повторних спроб, спосіб нормалізації, критерії та вектор ваг. Основна структура зберігається у полі JSONB, а ключові атрибути (коди метрик, тип нормалізації, прапор «використовує ваги») дублюються в окремих колонках для індексації й фільтрації.

Кандидатні підходи до узгодження поведінки описані у candidates (назва/клас методу, версія, короткий опис), а їх специфічні параметри – у candidate_params як ключ–значення (JSONB, посилання на candidate_id).

Третій блок – це запуски, результати, рішення та артефакти. Таблиця runs є заголовком прогону: містить run_id (UUID), ініціатора (user_id), часові мітки start/finish, статус (queued, running, completed, failed…), посилання на evaluation_config_id, обраний метод ранжування, config_hash і seed для відтворюваності.

Зв’язок між прогоном і версіями датасетів задає run_datasets (run_id, dataset_version_id, опційна вага набору). Обробка розбивається на порції, які описує batches (run_id, номер батчу, розмір, статус, лічильник спроб, часові мітки).

Результати на рівні окремого завдання зберігаються у item_results: run_id, dataset_version_id, task_id, candidate_id, посилання на відповідь (URI у файловому/об’єктному сховищі або короткий текст), latency_ms, лічильники токенів, ознака format_ok, прапорці безпеки safety_flags (JSONB), коди помилок.

Агреговані показники – у metric_results: run_id, candidate_id, dataset_version_id (або спеціальна «загальна» мітка), metric_code, value, за потреби – довірчі інтервали та спосіб оцінювання. Можливі метрики централізовано описані у metric_definitions (код, назва, напрям інтерпретації «більше/менше краще», текстовий опис).

Після ранжування формується таблиця decisions: run_id, selected_candidate_id (за наявності однозначного лідера), використаний метод MCDA, вектор ваг, підсумковий бал, місце в рейтингу, узагальнений результат чутливісного аналізу, коротке текстове обґрунтування.

Файли та проміжні артефакти описуються у artifacts: run_id, тип (raw_jsonl, таблиця метрик, звіт), шлях/URI, checksum, розмір і службові мітки. Це дозволяє відділити великі об’єкти від реляційних даних, зберігаючи посилання для відтворюваності.

Для прозорості дій використовується audit_log у режимі append–only: фіксуються користувач, час, тип об’єкта (конфігурація, датасет, запуск, користувач/роль, звіт), дія (створення, зміна, видалення, запуск, доступ), стислий опис змін «до/після» та технічні атрибути (IP, user–agent).

Наступний блок – зв’язки, індекси та обмеження. Основні зв’язки відображають життєвий цикл експериментів:

- users 1–M runs;
- runs 1–M batches і 1–M item_results;
- dataset_versions 1–M tasks, M–N із runs через run_datasets;
- candidates 1–M item_results та 1–M metric_results;
- metric_definitions 1–M metric_results.

Зовнішні ключі, як правило, мають стратегію ON DELETE RESTRICT, щоб не втрачати історичні результати. Для ідемпотентності результатів використовується унікальне обмеження на кшталт UNIQUE(run_id, task_id, candidate_id). Версії датасетів та конфігурацій мають обмеження UNIQUE(dataset_id, version) та унікальні імена. Для пришвидшення запитів використовуються індекси на стовпцях run_id, candidate_id, status, а також GIN–індекси для полів JSONB (наприклад, safety_flags). За великих обсягів передбачається партиціювання таблиці item_results за run_id або періодами часу.

Числові поля захищаються CHECK–обмеженнями (latency_ms ≥ 0, частки та ймовірності в [0;1]), статуси задаються через ENUM або CHECK. Усі імена таблиць і колонок – у стилі snake_case, первинні ключі реалізовані як UUID. Еволюція схеми відбувається через міграції (up/down);для JSONB–структур використовується schema_version та адаптери для читання старих записів.

Розглянемо також конфіденційність і доступ. «Сирі» відповіді моделей (які можуть містити чутливу інформацію) зберігаються поза реляційною БД – лише у сховищі артефактів, а в БД залишається мінімально необхідний технічний зріз (часи, лічильники, прапорці, ідентифікатори). Доступ до сирих артефактів обмежується спеціальними ролями, тоді як агреговані метрики й рішення

доступні ширшому колу. Секретні значення (ключі, токени, паролі) ніколи не потрапляють до таблиць з результатами й аудиту.

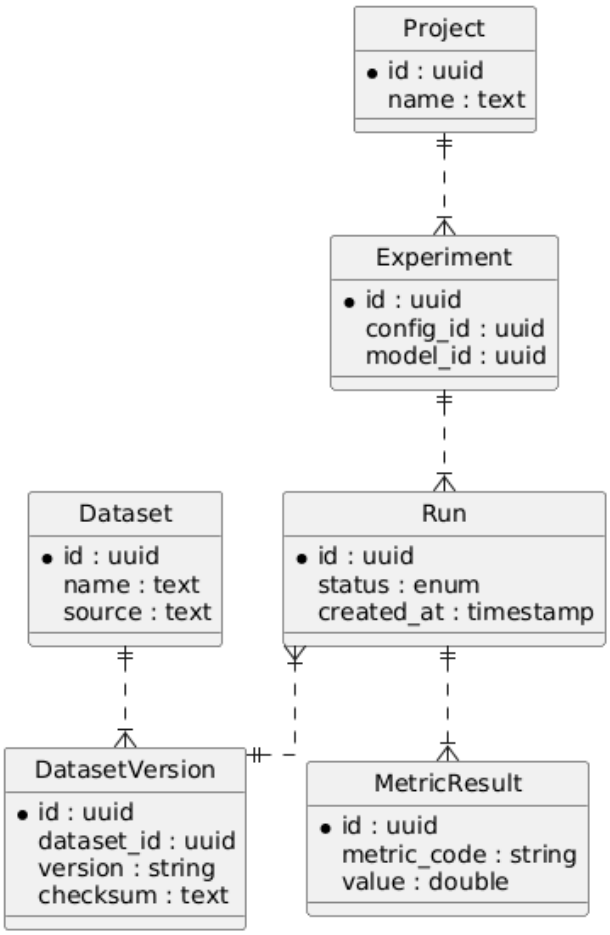


Рисунок 3.5. Фрагмент логічної моделі даних (або ER–діаграма). системи

У сукупності така схема забезпечує відтворюваність експериментів, зручний аналітичний доступ до метрик, прозорий аудит та контрольований доступ до чутливих даних, створюючи надійну основу для API та користувацького інтерфейсу, описаних у наступних підпунктах.

API системи побудовано як ресурсно–орієнтований REST–інтерфейс поверх HTTPS із використанням JSON. Усі часові мітки – у форматі ISO 8601 (UTC), коди статусів HTTP відповідають семантиці операцій (2xx, 4xx, 5xx). Запити є стейтлес, для трасування використовується X–Request–Id, що проходить крізь усі сервіси й логи. Автентифікація та авторизація відповідають вимогам підпункту 3.1: клієнти передають короткоживучі токени в заголовку Authorization: Bearer <token>, права задаються ролями й політиками доступу.

Ресурси та маршрути.

Для роботи з довідниками датасетів використовуються:

GET /v1/datasets – перелік наборів;

GET /v1/datasets/{id} – деталі одного набору;

GET /v1/dataset–versions?dataset_id=… – версії набору;

GET /v1/tasks?dataset_version_id=…&page=… – завдання з пагінацією.

Конфігурації оцінювання – GET/POST /v1/evaluation–configs, кандидатні підходи – GET/POST /v1/candidates.

Запуски оцінювання керуються ресурсом runs:

POST /v1/runs – створення прогону;

GET /v1/runs/{id} – статус і «паспорт» прогону;

POST /v1/runs/{id}:cancel – коректне скасування.

Агреговані метрики доступні через GET /v1/runs/{id}/metrics, детальні результати – GET /v1/runs/{id}/results?candidate_id=…, підсумкове рішення – GET /v1/runs/{id}/decision. Артефакти перераховуються через GET /v1/artifacts?run_id=…, записи аудиту – через GET /v1/audit?object_type=…&object_id=….

Основні ресурси та маршрути програмного інтерфейсу, через які клієнтська частина взаємодіє з сервером, наведено в табл. 3.2.

Таблиця 3.2. Опис API–ендпоїнтів

| **HTTP Метод** | **Ресурс (URI)** | **Призначення** |
|---|---|---|
| GET | /v1/datasets | Отримання списку доступних наборів даних та їхніх версій |
| GET | /v1/evaluation–configs | Перегляд та створення конфігурацій оцінювання |
| POST | /v1/candidates | Реєстрація нового методу узгодження (кандидата) |
| POST | /v1/runs | Ініціювання нового запуску експерименту (Run) |
| GET | /v1/runs/{id} | Отримання статусу прогону та паспорта експерименту |
| GET | /v1/runs/{id}/results | Завантаження детальних покрокових результатів (item–level) |
| GET | /v1/runs/{id}/metrics | Завантаження обчислених агрегованих метрик |
| GET | /v1/runs/{id}/decision | Отримання результату багатокритеріального ранжування |
| GET | /v1/artifacts | Доступ до файлів звітів та «сирих» логів |
| GET | /v1/audit | Перегляд журналу аудиту дій |

Уся структура обміну даними зафіксована в одному головному документі (OpenAPI). Він слугує еталоном для розробників і дозволяє автоматично перевіряти, чи відповідає написаний код узгодженим правилам.

Колекційні відповіді повертають список у полі items та інформацію про пагінацію (page, size, total або курсор next_cursor). Поширені параметри – limit, cursor, фільтри та сортування в query–рядку, наприклад:

GET /v1/runs?status=completed&sort=–started_at.

Помилки повертаються у сталому обгорнутому форматі:

```
{
 "error": {
  "code": "LIMIT_EXCEEDED",
  "message": "API rate limit exceeded for this token.",
  "details": {"limit": 120, "window_sec": 60},
  "request_id": "b7f6…"
 }
}
```

Тут code використовується для машинної обробки, message може локалізуватись, request_id дозволяє швидко знайти подію в логах.

Операції створення (зокрема POST /v1/runs) підтримують ідемпотентність через заголовок Idempotency–Key: повторний запит з тим самим ключем повертає той самий результат. При перевищенні квот запитів повертається 429 Too Many Requests та заголовки RateLimit–Limit, RateLimit–Remaining, RateLimit–Reset.

Зовнішні інтеграції можуть підписуватися на події:

– зміна стану прогону (run.status.changed);
– готовність метрик (run.metrics.ready);
– публікація рішення (run.decision.published);

Повідомлення містять type, version, occurred_at, run_id, стислий опис розбіжностей між попереднім та поточним станом об'єкта і URL для отримання повної інформації через основне API. Доставка є надійною: у разі збою виконується серія повторів із експоненційною затримкою до отримання відповіді 2xx. Для перевірки автентичності використовується підпис (наприклад, HMAC) у спеціальному заголовку.

Версіонування здійснюється на двох рівнях:

1. Основна версія в шляху (/v1/...). У межах однієї версії дозволені лише зворотно сумісні зміни: додавання необов'язкових полів із безпечними значеннями за замовчуванням, розширення enum–ів, нові ендпоїнти. Будь–які несумісні зміни (перейменування/видалення полів або маршрутів, зміна типів, зміна семантики) допускаються лише в новій версії (/v2/...);

2. Поле schema_version у тілі ресурсів (наприклад, для evaluation_config), що дозволяє еволюціонувати структури без негайного підвищення версії шляху й підтримувати читання старих записів через адаптери.

Політика знецінення передбачає оголошення змін у changelog, паралельну підтримку старої та нової схеми протягом перехідного періоду та використання заголовків Deprecation і Sunset перед повним вимкненням застарілих шляхів. На основі OpenAPI–опису будуються автоматизовані контрактні й consumer–driven тести, а за потреби генеруються легкі SDK (TypeScript, C#) з обгортками для пагінації, ідемпотентних викликів і розбору помилок.

Усі ендпоїнти доступні лише через HTTPS. Агреговані метрики й статистика можуть бути доступні ширшому колу ролей, тоді як доступ до «сирих» відповідей обмежується;посилання на артефакти робляться короткочасними й перевіряються за правами доступу. Чутливі поля маскуються як у логах, так і у вебхуках.

Про зміни в API інтегратори інформуються через CHANGELOG, сповіщення й окремі гайди для міграцій між основними версіями (наприклад, v1 -> v2).

У підсумку така організація контрактів API робить інтеграції передбачуваними й стабільними в часі: невеликі оновлення не ламають існуючі клієнтські застосунки, а суттєві зміни впроваджуються через нову основну версію з прозорими правилами міграції. Після опису зовнішніх контрактів логічно перейти до того, як користувачі працюють із системою через інтерфейс.

Інтерфейс користувача проєктують з урахуванням трьох основних ролей: аналітика (запускає оцінювання, переглядає метрики, порівнює підходи),

рецензента/контролера якості (перевіряє спірні приклади, надає людські оцінки) та адміністратора (керує користувачами, ролями, квотами й параметрами безпеки), на базі яких можуть формуватись дочірні інтерфейси з невеликими коригуванням базових. Для всіх ролей ключовою вимогою є відтворюваність: з будь–якого екрану має бути доступ до «першоджерел» – відповідної версії датасета, конфігурації оцінювання, артефакту або запису експерименту.

Глобальна структура інтерфейсу базується на лівій панелі навігації (sidebar) та верхній панелі керування. Ліва панель містить основні розділи системи: «Головна (огляд)», «Датасети», «Конфігурації оцінювання», «Кандидати (методи)», «Запуски (Runs)», «Результати», «Рішення», «Артефакти», «Аудит», а також «Користувачі/Ролі» (доступний лише адміністраторам). На верхній панелі розміщені пошук, перемикач мови (українська/англійська) та меню профілю користувача. Для позначення поточного контексту використовуються т.зв. «хлібні крихти» (breadcrumbs) – клікабельні елементи навігації інтерфейсу користувача, які показують ієрархічний шлях до поточної сторінки та дозволяє швидко повернутися на вищі рівні, а блок «швидких дій» дозволяє виконувати типові операції на будь–якому екрані, зокрема створити новий запуск, експортувати дані у формат CSV/JSON або створити копію наявної конфігурації.

Головна сторінка виконує роль оглядової панелі. На ній розміщуються компактні інформаційні блоки з ключовими показниками: список останніх запусків зі статусами, індикатор «найкращого» кандидата за підсумковим балом для останнього запуску, графік із динамікою якості та безпеки, а також панель сповіщень (наприклад, про наближення до лімітів API чи виявлені порушення порогів). Усі плитки є інтерактивними: перехід із них веде до детальних сторінок відповідних прогонів, рішень або налаштувань.

Розділ датасетів подає перелік наборів даних у вигляді таблиці з фільтрами за доменом, ліцензією, ризиковими темами, розміром та датою створення. Для кожної версії датасета доступна окрема картка, де відображаються контрольна сума (checksum), опис, політика використання й приклади завдань (за потреби –

з анонімізацією чутливої інформації). Інтерфейс передбачає типові дії: використати датасет у новому запуску, переглянути список завдань, експортувати індекс прикладів для стороннього аналізу.

Конфігурації оцінювання редагуються у спеціальній формі з вбудованими валідаторами. Користувач може обрати метрики та методи їх обчислення, задати обов'язкові пороги (безпека, формат, час відповіді), параметри генерації, рівні паралельності й кількість повторів, спосіб нормалізації та вектор ваг для багатокритеріального вибору. Передбачено перевірку сумісності конфігурації з обраними датасетами й кандидатами (чи всі метрики підтримуються) та можливість порівняння двох конфігурацій у вигляді «diff». Для забезпечення відтворюваності окрему версію конфігурації можна «заморозити» через дію «Заблокувати версію».

Розділ кандидатів містить список підходів до узгодження поведінки моделей із класифікацією за типом (інструкційне донавчання, навчання на вподобаннях, «конституційний» підхід, політики на етапі інференсу). Для кожного кандидата відображається «паспорт»: версія, параметри, основні залежності, а також історія отриманих результатів у різних експериментах. Додаткова функція «тестовий прогін» дозволяє виконати невелике пробне оцінювання на обмеженому наборі завдань для попереднього ознайомлення з поведінкою методу.

Сторінка запусків подає список прогонів зі статусами queued, running, completed і failed, із можливістю фільтрації за ініціатором, конфігурацією оцінювання та датасетом. Для кожного прогону доступна деталізована сторінка, що відображає часову шкалу подій, прогрес обробки батчів, кількість повторів, використані ліміти зовнішніх API та фрагменти логів із кореляційним request_id. Дозволені дії включають скасування поточного прогону, відновлення з контрольної точки у разі часткового збою, а також позначення прогону як «базового» для подальших порівнянь.

Розділ результатів підтримує два основні способи подання: «зведений» (dashbord ключових метрик по кандидатах) та «порівняльний» (таблиця або

подання side–by–side з використанням «теплокарт» для візуалізації того, де який кандидат кращий або гірший). Користувач може перейти до зрізу «кандидат × датасет», а також до рівня окремих прикладів: переглянути відповідь моделі, причину відхилення, позначки ризику й порушення формату, час відповіді та кількість токенів. Передбачено функцію формування «ручної вибірки» прикладів із чек–листом для людського оцінювання, а також можливість позначення спірних випадків для подальшого рецензування.

Сторінка рішень подає зведену пояснювальну картку. В ній зазначаються використаний метод багатокритеріального аналізу (MCDA), спосіб нормалізації, вектор ваг критеріїв, підсумкові бали й ранги кандидатів, а також висновки чутливісного аналізу (наприклад, стабільність рекомендації при зміні ваг на певний відсоток). З цього екрану доступні посилання на вихідні таблиці, графіки та конфігурацію запуску. Передбачено формування звіту у форматі PDF, який містить таблиці, діаграми, список порогів та коротке текстове обґрунтування.

Для артефактів реалізовано окремий інтерфейс перегляду списку файлів: «сирі» відповіді, агреговані таблиці, графіки, сформовані звіти. Доступ до цих матеріалів контролюється ролями, а посилання на завантаження мають обмежений строк дії. Журнал аудиту дозволяє фільтрувати записи за типом дії (створення, зміна, видалення, доступ), об’єктом і користувачем, для конфігурацій показуються фрагменти «до/після», що підвищує прозорість змін.

У ситуаціях, коли дані ще не створено (наприклад, відсутні конфігурації або кандидати), інтерфейс показує спеціальні «порожні стани»: коротке пояснення призначення розділу та пропозиції перших кроків («Створіть першу конфігурацію», «Імпортуйте датасет»). Довідкові підказки пояснюють ключові терміни в доступній формі, наприклад: «Поріг – мінімально прийнятне значення метрики;якщо метод його не виконує, він відсіюється з подальшого розгляду».

Для подання даних використовуються таблиці з можливістю фіксації колонок, сортування й швидкої фільтрації, а також різні типи графіків (стовпчикові, лінійні, box–plot для розподілів, «теплокарти» для порівняння кандидатів). Важливі стани позначаються бейджами–попередженнями

(перевищено поріг, не пройдено формат–валідацію, надто великий час відповіді тощо). Для будь–якої таблиці чи візуалізації надається можливість експорту у формат CSV або JSON, що спрощує подальший аналіз у зовнішніх інструментах.

Інтерфейс розробляється з урахуванням вимог доступності рівня WCAG 2.1 AA. Контраст тексту й керувальних елементів відповідає встановленим нормам, усі інтерактивні елементи доступні для клавіатурної навігації (послідовний порядок переходу клавішею Tab із чітко видимим фокусом). Кожна кнопка чи іконка має текстову мітку для скрін–рідерів (атрибути aria–label), а графіки супроводжуються альтернативними текстовими описами й табличними дублікатами даних. Стан елементів не передається виключно кольором: додатково використовуються піктограми та текстові підписи («ОК», «Помилка», «Попередження»). Передбачається режим підвищеного контрасту й можливість збільшення шрифту. Повідомлення про помилки формулюються зрозумілою мовою і супроводжуються короткими порадами щодо виправлення.

Інтерфейс є двомовним (українська та англійська мови). Числа, дати й грошові значення форматуються відповідно до вибраної локалі користувача. Усі текстові рядки винесені в словники локалізації, що полегшує подальше розширення. У звітах і журналах вказуються як абсолютні дати й час в UTC, так і локальний час користувача для полегшення інтерпретації.

Для роботи з великими обсягами даних застосовується порційне завантаження (пагінація або віртуалізація списків), а тривалі обчислення виконуються у фоні з відображенням сповіщень про завершення. Під час завантаження даних використовуються «скелетони», щоб уникнути повністю порожніх екранів. Форми підтримують збереження чернеток, автоматичну перевірку введених значень і візуальне підсвічування полів із помилками.

Видимість розділів, кнопок і окремих дій визначається роллю користувача та політиками доступу. Операції, що змінюють дані або впливають на результати експериментів (наприклад, зміна порогів, видалення прогону), супроводжуються діалогом підтвердження з коротким поясненням можливих наслідків. Секрети й

ключі доступу ніколи не відображаються повністю;журнали доступів і дій доступні лише користувачам із відповідними повноваженнями.

У цілому така організація інтерфейсу забезпечує користувачам послідовний шлях від налаштування й запуску оцінювання до отримання обґрунтованого рішення, робить процес оцінювання прозорим, а доступ – інклюзивним, контрольованим і безпечним.

Розглянемо яким чином система зберігає та застосовує налаштування, організовує захист чутливих даних (ключів, паролів, токенів), а також забезпечує спостереження за власною роботою. Основна мета полягає у тому, щоб зробити розгортання керованим і відтворюваним, а експлуатацію – прозорою, прогнозованою та такою, що піддається контролю.

Налаштування системи доцільно поділяти на дві групи: публічні та чутливі. До публічних відносяться адреси сервісів, параметри візуального оформлення інтерфейсу, значення тайм–аутів, перемикачі окремих функцій та інші технічні параметри, які не містять секретної інформації. Для кожного середовища (розробка, тестування, пілотна експлуатація) такі налаштування зберігаються окремо – у конфігураційних файлах середовища або у змінних оточення, без «жорсткого» вшивання значень у код. Чутливі налаштування (паролі до бази даних, ключі доступу до зовнішніх API, секрети вебхуків) ніколи не потрапляють до репозиторію коду та не зберігаються у відкритому вигляді у файлах конфігурації. Для забезпечення відтворюваності кожен запуск оцінювання супроводжується збереженням «знімка» застосованої конфігурації разом із контрольним хешем, що дозволяє іншому досліднику відтворити експеримент в ідентичних умовах. Тимчасові зміни для окремого прогону (наприклад, інший тайм–аут або модифікований поріг метрики) фіксуються як параметри «пісочниці»: вони діють лише в межах конкретного запуску і не впливають на глобальні налаштування.

Чутливі дані централізовано зберігаються у спеціалізованому сховищі секретів (типу Vault, Key Vault або Secrets Manager), де доступ до кожного секрету надається за принципом мінімально необхідних прав. Це означає, що

серверна частина (бекенд) одержує лише ті ключі, які потрібні для її роботи, модуль–воркер – лише ключі до моделей, а користувацький інтерфейс не має прямого доступу до секретів взагалі. Регламент ротації (планової заміни) ключів передбачає короткий період співіснування старого й нового значення, автоматичну перевірку працездатності після оновлення та можливість швидкого відкату у випадку помилки. У журналах подій та аналітичних звітах секрети відображаються лише у маскованому вигляді;усі спроби читання, зміни або створення секретів реєструються в аудиті.

Система моніторингу організовується у трьох взаємопов’язаних вимірах. По–перше, вимірюються показники роботи прикладного рівня: час відповіді API, частка технічних помилок, довжина черги запусків, прогрес обробки пакетів, частота повторних спроб, використання лімітів зовнішніх сервісів та орієнтовні витрати (якщо оцінювання здійснюється через платні API). По–друге, відстежується стан інфраструктури: завантаження процесора й пам’яті, обсяг і стан сховищ, кількість підключень до бази даних і можливі блокування, розмір ключових таблиць та індексів, доступний простір для артефактів і резервних копій. По–третє, збираються структуровані журнали подій із кореляційними ідентифікаторами запитів і запусків. Завдяки цим ідентифікаторам усі події, пов’язані з конкретним прогоном або користувацькою дією, можуть бути об’єднані у цілісний ланцюжок для аналізу й пошуку причин відхилень.

Для детального аналізу продуктивності ключові операції інструментуються засобами трасування. Кожен запит до API породжує послідовність позначених кроків, які відображають звернення до бази даних, операції з файловим або об’єктним сховищем, виклики зовнішніх моделей та інші залежності. Це дозволяє визначити, на якому саме етапі виникають затримки: у мережевій взаємодії, дискових операціях чи при роботі стороннього сервісу. Щоб не перевантажувати сховище телеметрії, застосовується вибіркове збирання: детальні траси зберігаються переважно для помилок та повільних запитів, а типові виклики агрегуються в статистичні розподіли. Водночас забезпечується дотримання вимог приватності: у телеметрію не записуються

повні відповіді моделей або фрагменти текстів, які можуть містити чутливі дані;за потреби фіксуються лише хеші або короткі технічні узагальнення.

Кожен сервіс надає стандартні точки перевірки стану: «життєздатність» (health) та «готовність» (readiness). Перша засвідчує, що процес запущений і не перебуває в аварійному стані, друга – що всі критичні залежності (база даних, сховище артефактів, зовнішні API в межах дозволених лімітів) доступні й функціонують коректно. Інструменти розгортання (оркестратор контейнерів, системний менеджер служб) використовують ці перевірки для автоматичних перезапусків і правильного розподілу навантаження. В інтерфейсі адміністратора передбачена оглядова сторінка стану, де відображаються показники справності компонентів та рекомендації щодо подальших дій у разі відхилень.

Для раннього виявлення проблем налаштовується система попереджувальних сповіщень за ключовими пороговими значеннями. Зокрема, сповіщення формуються у випадках, коли частка помилок класу 5xx перевищує заданий відсоток протягом певного інтервалу часу, коли черга запусків невиправдано зростає, а обробка не просувається, коли система наближається до лімітів зовнішнього постачальника або зменшується обсяг вільного місця на диску. Такі сигнали доставляються до спільних каналів команди (електронна пошта, месенджер, система керування інцидентами) разом із короткими інструкціями перших кроків аналізу – перевірити доступність бази, переглянути останні журнали, за потреби тимчасово призупинити нові запуски тощо.

Для пілотного розгортання формулюються прості, вимірювані цілі якості обслуговування. Для інтерактивних дій у вебінтерфейсі та через API більшість запитів має оброблятися в межах часток секунди, а обробка пакетів тестових завдань – прогресувати рівномірно, без тривалих «зависань». У разі відмови воркера або зовнішнього сервісу система повинна відновити штатну роботу впродовж прийнятного часу, а ризик втрати даних мінімізується завдяки регулярним резервним копіям і проміжному збереженню результатів. Такі

показники вимірюються автоматично й відображаються на дашбордах моніторингу.

Організація моніторингу та телеметрії здійснюється з мінімізацією потрапляння чутливих даних у журнали, сповіщення та трасування. У випадках, коли без прикладів обійтися неможливо (наприклад, при розслідуванні інциденту), доступ до відповідних артефактів мають лише користувачі з відповідними ролями, а посилання на файли мають обмежений строк дії. Строки зберігання даних налаштовано так, щоб забезпечити достатній горизонт для аналізу й одночасно не накопичувати надлишкову інформацію: технічні журнали зберігаються від кількох тижнів до місяця, агреговані метрики – довше, а детальні трасування – впродовж часу, необхідного для завершення розслідувань.

У адміністративному інтерфейсі передбачена сторінка технічного стану системи, де відображаються актуальні версії компонентів, результати останніх міграцій бази даних, поточні ліміти, активні попередження та посилання на деталізовані дашборди. Адміністратор має можливість тимчасово призупинити приймання нових запусків у разі планових робіт або розслідування інциденту;такі дії обов’язково протоколюються із зазначенням причин і часових рамок. У сукупності це забезпечує прозорий цикл життя застосунку: налаштування відокремлені від коду й версіонуються, секрети захищені та контрольовані, а робота сервісів відстежується за допомогою вимірюваних показників, журналів та трас. Така організація дозволяє своєчасно виявляти і усувати проблеми, відтворювати експерименти та поступово переходити від суто навчальної інсталяції до ширшого пілотного використання без суттєвих ризиків.

Архітектуру системи сформовано з урахуванням двох вимог: просте розгортання в навчально–пілотних умовах і можливість подальшого масштабування без радикальної перебудови. Для роботи системи обрано модульний моноліт на .NET 8 із класичним поділом на шари (вебінтерфейс, API/бізнес–логіка, доступ до даних) і виділенням доменної логіки в окремі модулі. Взаємодія з зовнішніми сервісами (API мовних моделей, файлові сховища, черги) здійснюється через адаптери зі стабільними контрактами, що

дозволяє змінювати провайдерів без переписування ядра системи. Загальну компонентну структуру можна представити як показано на рис. 3.1.

Основними принципами проектування є модульність і слабке зчеплення, незалежність від постачальників моделей, централізоване збереження версій датасетів і конфігурацій для відтворюваності експериментів, рольова модель доступу з аудитом дій користувачів, наскрізне логування та простота деплою (для пілоту достатньо одного вебзастосунку та БД).

Користувач працює через вебінтерфейс, де створює та редагує проекти, експерименти й запуски, переглядає метрики та звіти. UI взаємодіє з бекендом через HTTP/JSON–API. Бекенд керує довідниками (моделі, методи, набори даних, метрики), зберігає конфігурації оцінювання, застосовує порогові умови, нормалізує показники й обчислює підсумкові бали за методами багатокритеріального аналізу, формуючи рейтинг кандидатних підходів і короткі текстові висновки.

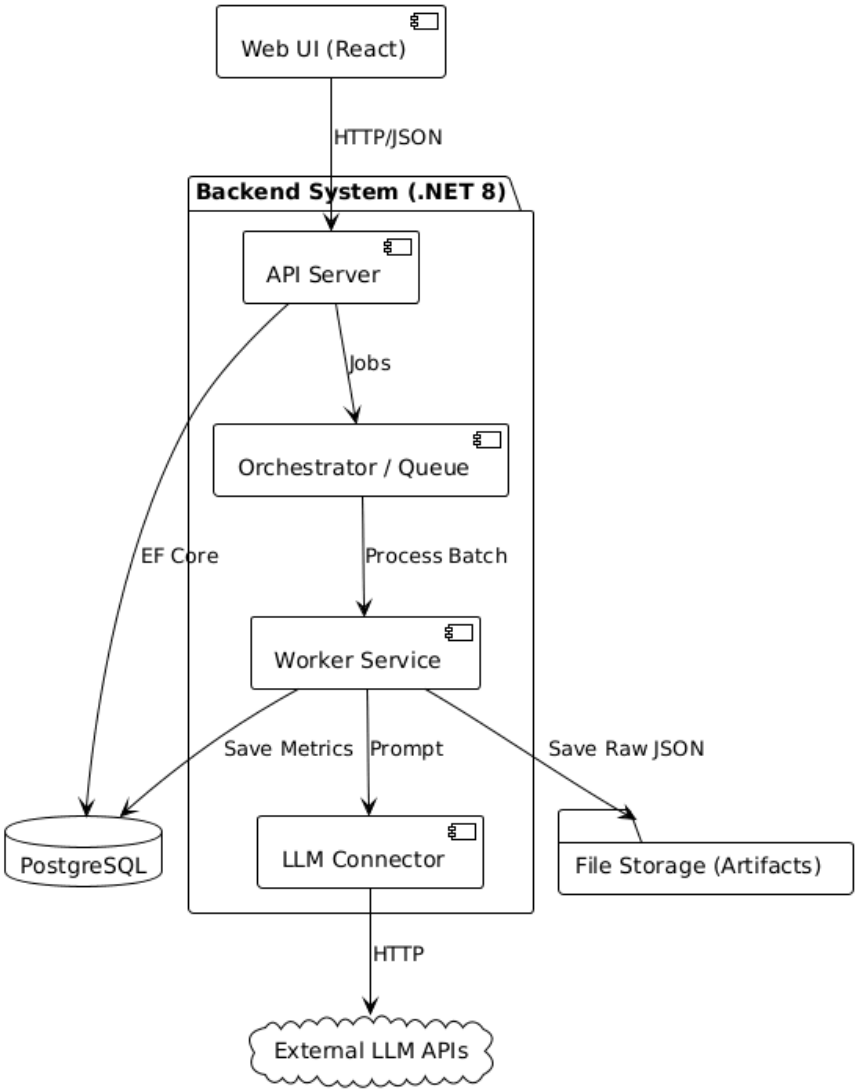


Рисунок 3.6. Компонентна архітектура системи

Обчислювально важкі операції можуть виконуватися або безпосередньо в .NET–бекенді, або у винесеному модулі оцінювання (наприклад, на Python), який отримує завдання через внутрішній HTTP/gRPC–інтерфейс чи чергу. Блок оркестрації запусків відповідає за життєвий цикл прогонів (планування, зміна статусів, повторні спроби) і фіксує службові журнали. Доступ до мовних моделей стандартизовано через конектори з єдиним інтерфейсом: вони інкапсулюють автентифікацію, тайм–аути, повторні спроби та облік витрат ресурсів.

Дані зберігаються в реляційній базі (PostgreSQL) та файловому сховищі. У БД фіксуються проекти, експерименти, запуски, конфігурації, агреговані значення метрик і записи аудиту, у файловому або S3–сумісному сховищі –

«сирі» відповіді моделей, вибірки, журнали й сформовані звіти. Це зменшує навантаження на БД і спрощує архівацію артефактів.

Безпека реалізована через RBAC–модель: різні ролі (інженер, аналітик, фахівець із безпеки, адміністратор, гість) мають відмінні права на створення запусків, зміну конфігурацій, перегляд «сирих» відповідей і роботу зі звітами. Критичні операції журналюються в аудит–лог із фіксацією користувача, часу та суті змін, а технічні логи й метрики забезпечують спостережуваність та спрощують діагностику.

Типовий сценарій роботи виглядає так: користувач у вебінтерфейсі створює конфігурацію оцінювання та ініціює запуск, бекенд зберігає параметри в БД і передає завдання оркестратору (організатору черг), той організовує виклики до мовних моделей через конектори (та, за потреби, до зовнішніх модулів оцінювання), збирає результати, зберігає артефакти й метрики, після чого бекенд застосовує порогові умови, виконує багатокритеріальне ранжування та повертає користувачеві рейтинг кандидатних методів із поясненнями й можливістю експорту звіту.

У такій конфігурації система поєднує простоту розгортання на одному вузлі з чітким розподілом відповідальностей між компонентами й можливістю подальшого винесення окремих модулів у самостійні сервіси без змін базової предметної логіки

Усі операції в системі прив’язуються до ієрархії ідентифікаторів:

project_id —> experiment_id —> run_id. Для кожного прогону додатково фіксуються dataset_version, config_version та config_hash. Це дозволяє однозначно зв’язати результати з конкретним набором даних і налаштувань та відтворювати експерименти в тих самих умовах.

Під час створення експерименту користувач у вебінтерфейсі обирає мовні моделі, методи узгодження, набори даних і правила оцінювання (метрики, порогові значення, параметри генерації, ліміти часу). Конфігурація надсилається на бекенд HTTP/JSON–запитом, проходить валідацію й фіксується як версія

конфігурації, одночасно створюється чернетка прогону, що пов’язує експеримент, налаштування та вибрані датасети.

Після ініціювання запуску бекенд створює запис про прогін і передає його модулю оркестрації. Виконавець завантажує дані за зафіксованою версією, формує запити до мовних моделей через уніфікований конектор, отримує відповіді та виконує попередню обробку: перевірку формату (зокрема коректності JSON), базові перевірки безпеки, розрахунок проміжних метрик (латентність, частка токсичних відповідей, точність на еталонних завданнях). «Сирі» відповіді й журнали виконання зберігаються у файловому сховищі, агреговані метрики та службові статуси – у реляційній БД. Статус прогону послідовно змінюється від «заплановано» до «виконується» та «завершено»/«збій», проміжні дані доступні через API.

Після завершення всіх потрібних прогонів застосовуються порогові умови: кандидати, що порушують вимоги безпеки, формату або часу відповіді, відсіюються. Релевантні метрики нормалізуються (наприклад, за схемою Min–Max з урахуванням типу показника), після чого обчислюється інтегральна оцінка: зважений бал відповідно до пріоритетів замовника або множина Парето, якщо ваги не задані. На цій основі формується рейтинг підходів і короткий текстовий опис виявлених компромісів (наприклад, вищий рівень безпеки за рахунок більшої затримки). Підсумковий звіт з таблицями та графіками зберігається як артефакт прогону й може експортуватися для зовнішнього аналізу.

Центральне положення в схемі даних посідає таблиця Projects (проєкти). Для кожного проєкту зберігаються ідентифікатор (primary key), назва, короткий опис, домен застосування, дата створення та статус (активний, архівний тощо). Ця таблиця слугує «контейнером» для всіх експериментів у межах одного прикладного сценарію, тому таблиця Experiments (експерименти) має зовнішній ключ на Projects. В Experiments, окрім ідентифікатора, зберігаються: назва експерименту, опис мети, ідентифікатор пов’язаного проєкту, посилання на обрану модель, метод узгодження, конфігурацію оцінювання, дата створення,

статус (запланований, у процесі, завершений) та ідентифікатор відповідального користувача.

Опис самих мовних моделей зосереджено в таблиці Models. У ній передбачені поля: ідентифікатор, назва моделі, постачальник (власна розробка, ПЗ з відкритим кодом, комерційний API), версія, базовий розмір (за потреби), спосіб доступу (локальна/віддалена) та службові атрибути (наприклад, підтримувані мови інтерфейсу). Таблиця Methods (методи узгодження) містить інформацію про тип методу (інструкційне донавчання, методи на основі уподобань, правилові/«конституційні» тощо), короткий опис, вимоги до даних і можливі обмеження. У таблиці Experiments зберігаються зовнішні ключі на Models і Methods, що фіксують, яку модель і який метод узгодження оцінює конкретний експеримент.

Дані, з якими працює система, описуються таблицею Datasets (набори даних). Для кожного набору фіксуються: ідентифікатор, назва, версія, тип (навчальний, валідаційний, тестовий, безпековий), джерело (публічний ресурс, внутрішній набір), ліцензія, короткий опис змісту та формат збереження. Оскільки один і той самий набір даних може використовуватися в кількох експериментах, а один експеримент може комбінувати кілька наборів, зв’язок реалізується через проміжну таблицю ExperimentDatasets з полями: ідентифікатор, зовнішній ключ на Experiments, зовнішній ключ на Datasets та, за потреби, роль набору (наприклад, «тест якості», «тест безпеки»).

Параметри оцінювання групуються в таблиці EvalConfigs (конфігурації оцінювання). Вона містить ідентифікатор, назву конфігурації, перелік активних метрик (наприклад, у вигляді текстового поля або зв’язку з іншою таблицею), параметри генерації відповіді (максимальна довжина, температура, інші налаштування), базові пороги (гранична частка небезпечних відповідей, максимальний час відповіді тощо) і службові атрибути. Таблиця Experiments містить зовнішній ключ на EvalConfigs, що дозволяє повторно використовувати стандартні конфігурації для різних експериментів.

Запуски експериментів фіксуються в таблиці Runs. Кожен запис відповідає одному фактичному прогону експерименту з конкретною моделлю, методом, наборами даних і конфігурацією. У Runs зберігаються: ідентифікатор, зовнішній ключ на Experiments, дата й час початку та завершення, технічне середовище (наприклад, локально чи через API), статус (успішно, помилка, перервано), шлях до збережених артефактів (логів, вихідних відповідей). Кожен експеримент може мати кілька запусків, що дозволяє повторювати оцінювання з іншими параметрами або в іншому середовищі.

Для зберігання довідника метрик і їхніх значень використовуються дві таблиці: MetricDefinitions і MetricValues. У MetricDefinitions фіксуються: ідентифікатор, назва метрики (наприклад, Accuracy, UnsafeRate, Latency_avg), тип (частка, середнє значення, час, вартість), короткий опис способу обчислення та напрямок оптимізації («більше – краще» або «менше – краще»). Таблиця MetricValues містить: ідентифікатор, зовнішній ключ на Runs, зовнішній ключ на MetricDefinitions, числове значення, посилання на Dataset (якщо метрика обчислюється для конкретного набору) та додаткові службові поля (наприклад, довірчий інтервал, кількість прикладів). Така схема дозволяє гнучко додавати нові метрики без зміни структури основних таблиць.

Для представлення підсумків передбачено таблицю Reports (звіти). У ній зберігаються: ідентифікатор, назва, тип (технічний, управлінський, оглядовий), зовнішній ключ на Experiments, шлях до сформованого звіту (наприклад, файл PDF, HTML або стисла текстова версія), дата створення та автор. Один експеримент може мати кілька звітів, сформованих для різних аудиторій.

Користувачі системи описуються в таблиці Users. Для кожного користувача зберігається ідентифікатор, ім’я, контактна інформація (за потреби), унікальний логін (якщо використовується автентифікація) та статус (активний/заблокований). Ролі та права доступу реалізуються через окрему таблицю Roles (перелік доступних ролей) та проміжну таблицю UserRoles, яка пов’язує користувачів з конкретними ролями (інженер, аналітик, фахівець із безпеки, адміністратор). Таблиці Projects, Experiments та Reports мають зовнішні

ключі на Users (наприклад, «власник проєкту», «відповідальний за експеримент», «автор звіту»).

Загальна схема даних побудована таким чином, щоб:

– уникати надлишкового дублювання (нормалізація основних сутностей у окремі таблиці);

– забезпечувати можливість розширення (додавання нових метрик, типів методів, ролей без кардинальної перебудови структури);

– підтримувати відтворюваність експериментів (через збереження зв’язків між проєктами, експериментами, запусками, наборами даних, конфігураціями та метриками);

– спростити побудову звітів і аналітичних запитів (завдяки чітким зв’язкам «один до багатьох» і «багато до багатьох»).

На основі цієї інформаційної моделі у подальшому можуть бути побудовані діаграми «сутність–зв’язок» та логічна схема бази даних у вигляді UML– або ER–діаграм, які деталізуватимуть структуру таблиць і ключів для конкретної реалізації.

Після того як визначено основні сутності та схему даних, опишемо, як саме відбувається повний цикл роботи з експериментами та якими принципами керуємося під час зберігання, версіонування й захисту даних.

Інтеграція з мовними моделями реалізована через уніфікований HTTP/HTTPS–конектор з обміном JSON–повідомленнями. Він відповідає за автентифікацію, тайм–аути, повторні спроби та дотримання лімітів постачальника, тож перехід на іншого провайдера чи локальний сервер інференсу зводиться до зміни конфігурації. Окремі метрики, які зручніше обчислювати поза середовищем .NET (наприклад, детектори токсичності), розраховуються зовнішніми сервісами оцінювання: бекенд передає батч (партію) пар «запит–відповідь» і отримує таблицю значень з ідентифікаторами прикладів.

Для внутрішніх і зовнішніх взаємодій застосовуються стандартизовані JSON–схеми з явним полем schema_version. Це дає змогу контролювати еволюцію контрактів (зміну формальної угоди про формат даних які

передаються), зміни ж які порушують зворотну сумісність, супроводжуються міграціями бази даних і оновленням конекторів.

Зберігання даних реалізоване комбіновано – реляційна СУБД (PostgreSQL) містить довідники (моделі, методи узгодження, датасети, метрики), описи проектів, експериментів і запусків, версії конфігурацій, агреговані значення метрик, аудиторський журнал і посилання на артефакти, операції запису виконуються транзакційно. Об’ємні артефакти – «сирі» відповіді моделей, вибірки, сформовані звіти – розміщуються у файловому або S3–сумісному об’єктному сховищі з ієрархією каталогів за схемою {project_id}/{experiment_id}/{run_id}/…, що спрощує навігацію, архівацію та очищення. Для рідко використовуваних даних застосовуються «холодні» класи зберігання.

Для забезпечення цілісності й відмовостійкості передбачено регулярні резервні копії бази даних і артефактів, журналювання змін у критичних таблицях, політики повторних спроб із джитером та обмеження інтенсивності викликів зовнішніх сервісів. Довгі прогони розбиваються на батчі, повторно обробляються лише невдалі елементи, а внутрішні операції проєктуються як ідемпотентні (повторний виклик з тим самим run_id не дублює результати). У разі тривалої недоступності окремих сервісів прогін може завершуватися зі статусом часткового успіху, відсутні метрики явно позначаються у звіті й не використовуються в порівнянні без окремого застереження.

Питання безпеки вирішуються зберіганням облікових даних до зовнішніх API поза вихідним кодом, диференційованим доступом до «сирих» відповідей (лише для обмеженого кола експертів) і ширшим доступом до агрегованих метрик та звітів. Усі запити виконуються через HTTPS та журналюються з прив’язкою до користувача й часу. Технічні й бізнес–логи (старт і завершення прогонів, помилки, зміни конфігурацій, формування звітів) використовуються для побудови панелі моніторингу з ключовими показниками: кількість активних запусків, середня затримка, частка збоїв.

Відтворюваність результатів забезпечується збереженням версій датасетів і конфігурацій, параметрів генерації (довжина відповіді, температура, тайм–аут), версії конектора й, за можливості, самої моделі або API, а також значень «зерна випадковості» для стохастичних процесів. Це дозволяє різним командам повторити експерименти та перевірити отримані висновки. Після опису потоків даних та інтеграцій далі буде розглянуто механізми розмежування доступу та аудиту дій користувачів.

Підсистема безпеки охоплює три ключові аспекти: автентифікацію користувачів, авторизацію їхніх дій та аудит подій. У пілотному варіанті підтримуються два способи входу. Перший – локальні облікові записи (електронна пошта й пароль), де паролі зберігаються лише у вигляді стійких хешів (Argon2id / PBKDF2), діють вимоги до складності, ліміти невдалих спроб та, за потреби, двофакторна автентифікація. Другий – інтеграція з корпоративною системою єдиного входу (SSO) на основі OAuth2 / OpenID Connect.

Сесії вебкористувачів підтримуються cookie з атрибутами безпеки (HttpOnly, Secure, SameSite, час життя), для API–доступу та CLI–застосунків використовуються короткочасні токени (наприклад, JWT) з окремим механізмом оновлення. Паролі, секретні ключі та токени не зберігаються у відкритому вигляді й не потрапляють до логів.

Авторизація реалізована як рольова модель доступу (RBAC) з можливістю перевірки прав на рівні окремих проєктів, експериментів і запусків. Типові ролі:

– адміністратор – керування користувачами, ролями, глобальними довідниками;

– інженер – створення експериментів, запуск прогонів, налаштування конфігурацій своїх проєктів;

– аналітик – перегляд результатів і формування звітів;

– фахівець із безпеки – налаштування порогів та політик безпеки, аналіз ризикових відповідей;

– гість – перегляд опублікованих звітів без доступу до внутрішніх налаштувань.

Розподіл повноважень у системі реалізовано відповідно до рольової моделі (RBAC), матриця доступу для якої наведена в табл. 3.1.

Таблиця 3.1. Рольова модель доступу

| Роль | Створення експериментів | Зміна конфігурацій | Перегляд звітів | Керування користувачами | Налаштування безпеки |
|---|---|---|---|---|---|
| Гість (Viewer) | – | – | + | – | – |
| Аналітик | – | – | + | – | – |
| Інженер | + | + | + | – | – |
| Фахівець з безпеки | – | – / + | + | – | + (перегляд/аудит) |
| Адміністратор | + | + | + | + | + |

Доступ організовано за принципом найменших привілеїв: за замовчуванням надається мінімально необхідний набір прав, а перегляд «сирих» відповідей моделей обмежується вузьким колом експертів.

Аудит змін забезпечується окремим журналом аудиту в базі даних, що працює за принципом append–only. У нього потрапляють події входу й виходу користувачів, операції створення/зміни/видалення проєктів, експериментів і запусків, модифікація конфігурацій, порогів, ваг критеріїв, зміна політик доступу, експорт артефактів. Для кожної події фіксуються користувач, час, тип операції та задіяні об’єкти, при цьому доступ до журналу мають лише адміністратори та уповноважені особи. Передбачені механізми експорту й архівації журналу з контролем цілісності.

Усі взаємодії між компонентами відбуваються через захищені канали (HTTPS). Секрети конфігурації (API–ключі, рядки підключення до БД) зберігаються поза вихідним кодом – у змінних середовища або спеціалізованих менеджерах секретів, і підлягають періодичній ротації. На рівні зберігання застосовується шифрування дисків або томів, а особливо чутливі артефакти можуть розміщуватися в окремому зашифрованому сховищі з жорсткішими правилами доступу та скороченими строками зберігання, агреговані метрики та звіти дозволено зберігати довше.

Для захисту від технічних і фінансових ризиків реалізовано обмеження на частоту й обсяг викликів до зовнішніх мовних моделей (rate limiting), контроль розмірів вхідних/вихідних повідомлень і тайм–аути. Всі вхідні дані проходять валідацію, і доступ до БД здійснюється через параметризовані запити або ORM, що знижує ризик SQL–ін'єкцій. Під час імпорту файлів перевіряються тип і максимальний розмір. Для вебінтерфейсу застосовуються CSRF–токени, а система логування налаштована так, щоб у журналах не з'являлися паролі, токени та інші конфіденційні дані.

Політика конфіденційності передбачає фіксацію умов використання кожного набору даних і, за потреби, анонімізацію або повну заборону зберігання персональної інформації в «сирих» відповідях. Стандартні звіти орієнтовані на агреговані метрики й узагальнені приклади, що дозволяє аналізувати якість і безпеку без зайвого розкриття вихідних даних.

Операційна безпека підтримується регулярним переглядом ролей і прав доступу, ротацією паролів та ключів до зовнішніх сервісів, періодичним створенням і тестовим відновленням резервних копій, а також базовим планом реагування на інциденти (фіксація, ескалація, ізоляція, відкат конфігурації, інформування команди).

У підсумку автентифікація обмежує доступ до системи лише довіреними користувачами, авторизація задає чіткі межі їхніх дій, а аудит формує відтворювану історію подій. Сукупність цих механізмів забезпечує прозору та безпечну експлуатацію системи оцінювання методів узгодження поведінки великих мовних моделей і створює надійну основу для подальшого масштабування.

У навчально–пілотному сценарії система працює в трьох середовищах: розробки (Dev), перевірки/інтеграції (Stage) та пілотної експлуатації (Prod). Таке розділення дозволяє безпечно перевіряти зміни, а також порівнювати фактичні характеристики системи з нефункціональними орієнтирами щодо продуктивності, надійності та безпеки. Базовим підходом до розгортання є

контейнеризація: усі компоненти пакуються в Docker–образи, що забезпечує відтворюваність оточення.

У Dev застосунок запускається локально: один процес бекенда на .NET 8, за потреби окремий воркер, PostgreSQL у Docker–контейнері. Конфігураційні параметри читаються з файлів середовища, секрети (паролі, ключі API) передаються лише через змінні оточення. У Stage розгортається повний стек за допомогою Docker Compose (бекенд, БД, файлове сховище артефактів, фонова служба), тут перевіряються міграції схеми БД, права доступу, мережеві політики, «сухі» прогони експериментів та дотримання лімітів зовнішніх API. У Prod система працює на одному або кількох Linux–вузлах, увімкнено HTTPS через реверс–проксі, налаштовані регулярні резервні копії, менеджер секретів, ротація логів і базовий моніторинг, за потреби воркер обчислень може бути винесений на окремий сервер.

Процес збірки й розгортання максимально автоматизовано – застосунок пакується у версіоновані Docker–образи (семантична версія або хеш коміту), а розгортання в усі середовища здійснюється через Docker Compose з прописаними залежностями сервісів і healthcheck–перевірками. Конвеєр CI/CD виконує автоматичні тести, статичний аналіз коду, збірку образів, програвання міграцій і короткі smoke–тести на Stage (smoke tests – максимально прості, швидкі та поверхневі перевірки, які підтверджують, що система взагалі працює, і її основні компоненти «живі» після розгортання, оновлення або зміни конфігурації), і лише після успішного проходження цих етапів і ручного підтвердження відбувається оновлення Prod. Для зменшення простоїв можуть застосовуватися поетапний рестарт сервісів або стратегія розгортання типу blue/green (стратегія розгортання коли нова версія запускається паралельно зі старою, а переключення трафіку відбувається лише після перевірки її працездатності, що дозволяє оновлювати систему без простоїв і з мінімальним ризиком).

Керування конфігураціями побудовано на чіткому поділі параметрів. Публічні налаштування (адреси сервісів, тайм–аути, фіче–флаги) зберігаються у

файлах середовища під контролем системи керування версіями, тоді як секрети розміщуються у змінних оточення або спеціалізованому сховищі секретів. Схема БД версіонується через міграції EF Core, кожен реліз супроводжується пакетом міграцій та сценарієм відкату. Конфігурації мають власні ідентифікатори версії (config_version) та «відбиток» (config_hash), які фіксуються в записах експериментів, що забезпечує відтворюваність результатів і простежуваність змін.

Нефункціональні вимоги охоплюють продуктивність, надійність, масштабованість, супроводжуваність, спостережуваність, безпеку, портативність, сумісність і відтворюваність. Для інтерактивних сценаріїв задається орієнтир: час відповіді UI й API на Stage не повинен перевищувати близько 300 мс для 95–го перцентиля, для пакетних прогонів очікується наближено лінійне зростання часу обробки зі збільшенням обсягу даних, чого досягають за рахунок паралельної обробки батчів і асинхронного вводу–виводу. Надійність забезпечується ідемпотентністю критичних операцій (повторний запис для одного run_id не створює дублікатів), контрольними точками під час обробки великих наборів, а також розділенням «сирих» артефактів і агрегованих метрик. Масштабованість передбачає як вертикальне збільшення ресурсів, так і горизонтальне розділення компонентів (винесення воркерів, БД, API–шару) завдяки чітким інтерфейсам між ними.

Супроводжуваність підкріплюється модульною структурою, тестами для ключових частин, стислими технічними описами й практикою code review. Спостережуваність включає структуровані логи (події користувачів, зміни статусів прогонів, помилки інтеграцій, перевищення квот), технічні метрики (час відповіді API, кількість активних прогонів, частка збоїв, використання квот постачальників) та базові сповіщення. Безпека забезпечується повсюдним використанням HTTPS, рольовою моделлю доступу (RBAC), аудитом змін, зберіганням секретів поза вихідним кодом і політикою життєвого циклу даних (строки зберігання «сирих» відповідей, правила архівації й видалення, обмеження доступу до чутливих даних). Портативність досягається підтримкою

Linux та Windows (з пріоритетом Linux) і мінімальним переліком зовнішніх залежностей (Docker/Compose та PostgreSQL), а сумісність і розширюваність – версіонованими JSON–контрактами з явним schema_version і супровідними міграціями даних. Відтворюваність експериментів гарантується фіксацією dataset_version, config_version, config_hash, версій конекторів, моделей і основних параметрів середовища запуску.

Резервування та відновлення даних організовано таким чином, щоб мінімізувати час простою й втрати інформації. База даних резервується повними копіями з журналами змін, на Stage періодично виконується тестове відновлення. Артефакти (сирі відповіді, журнали, звіти) зберігаються у версіонованому сховищі з політикою переведення застарілих даних у «холодне» зберігання та контролем цілісності за хеш–сумами. Для критичних таблиць може застосовуватися відновлення «до моменту часу» (point–in–time recovery).

План реагування на інциденти передбачає фіксацію події, сповіщення відповідальних осіб, тимчасове блокування нових прогонів, локалізацію причини, за потреби відкат останніх змін конфігурації або повернення до попередньої стабільної версії та відновлення незавершених завдань із контрольних точок. Після відновлення готується короткий звіт про причини, вплив та вжиті заходи. Для пілотної інсталяції орієнтовні показники: час відновлення (RTO) вимірюється годинами, а допустима втрата даних (RPO) не перевищує одного дня й відповідає графіку резервного копіювання. Такий підхід задає прозорі правила розгортання й експлуатації та забезпечує необхідні нефункціональні характеристики системи для коректного порівняння методів узгодження поведінки мовних моделей.

**Висновки до Розділу 3**

Узагальнюючи матеріали розділу 3, можна виділити кілька ключових висновків щодо архітектури, алгоритмів та інтерфейсів розробленої системи. Нижче наведено основні з них.

У розділі сформовано цілісне архітектурне рішення системи: модульний моноліт на .NET 8 із чітким поділом на шари (UI, API/бізнес–логіка, доступ до даних), виділенням оркестратора прогонів та конекторів до мовних моделей, а також комбінованим зберіганням у PostgreSQL та файловому/S3–сховищі. Такий підхід поєднує простоту розгортання в навчально–пілотних умовах із можливістю подальшого масштабування без зміни доменної логіки.

Деталізовано життєвий цикл експериментів і ідентифікацію результатів: ієрархія project -> experiment -> run доповнена версіями датасетів і конфігурацій (dataset_version, config_version, config_hash, seed), а також контрольними точками для батчів. Це дає змогу відтворювати прогін у тих самих умовах, відокремлювати «сирі» артефакти від агрегованих метрик і гарантувати коректне відновлення після збоїв.

Формалізовано моделі та алгоритми ядра: від батчевої обробки запитів до моделей, первинних перевірок формату й безпеки та обчислення базових метрик – до багатокритеріального відбору кандидатів. Пороги безпеки, формату та часу задають «санітарний фільтр», після якого застосовуються нормалізація показників, зважена сума (та, за потреби, альтернативні MCDA–методи) й чутливісний аналіз, доповнені політиками повторних спроб, circuit breaker та ідемпотентністю операцій.

Побудовано логічну модель даних і механізм зберігання артефактів: виділено блоки довідників, конфігурацій, запусків, результатів і аудиту, описано ключові таблиці (users, datasets/dataset_versions, evaluation_configs, runs, item_results, metric_results, decisions, artifacts, audit_log), зовнішні ключі, унікальні обмеження й індекси. Така схема забезпечує цілісність і відтворюваність даних, зручний аналітичний доступ до метрик і прозорий журнал дій користувачів.

# РОЗДІЛ 4. РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ ТА КЕРІВНИЦТВО КОРИСТУВАЧА

## 4.1. Середовище розроблення та розгортання

Структура репозиторію програмного комплексу побудована з урахуванням принципів модульності, чіткого розмежування відповідальностей (Separation of Concerns) та підтримки повного циклу автоматизації розгортання (CI/CD). У межах єдиного репозиторію (monorepo) зосереджено вихідний код серверної частини, клієнтського вебдодатка, модулів фонової обробки, а також конфігурації для розгортання й спостережуваності системи.

Каталоги проєкту організовано таким чином, щоб доменна логіка була максимально ізольована від інфраструктурних деталей. Це відповідає архітектурному підходу «порти й адаптери» (Hexagonal Architecture) та модульному моноліту, обґрунтованим у третьому розділі роботи. Узагальнену схему файлової структури наведено на рис. 4.1.

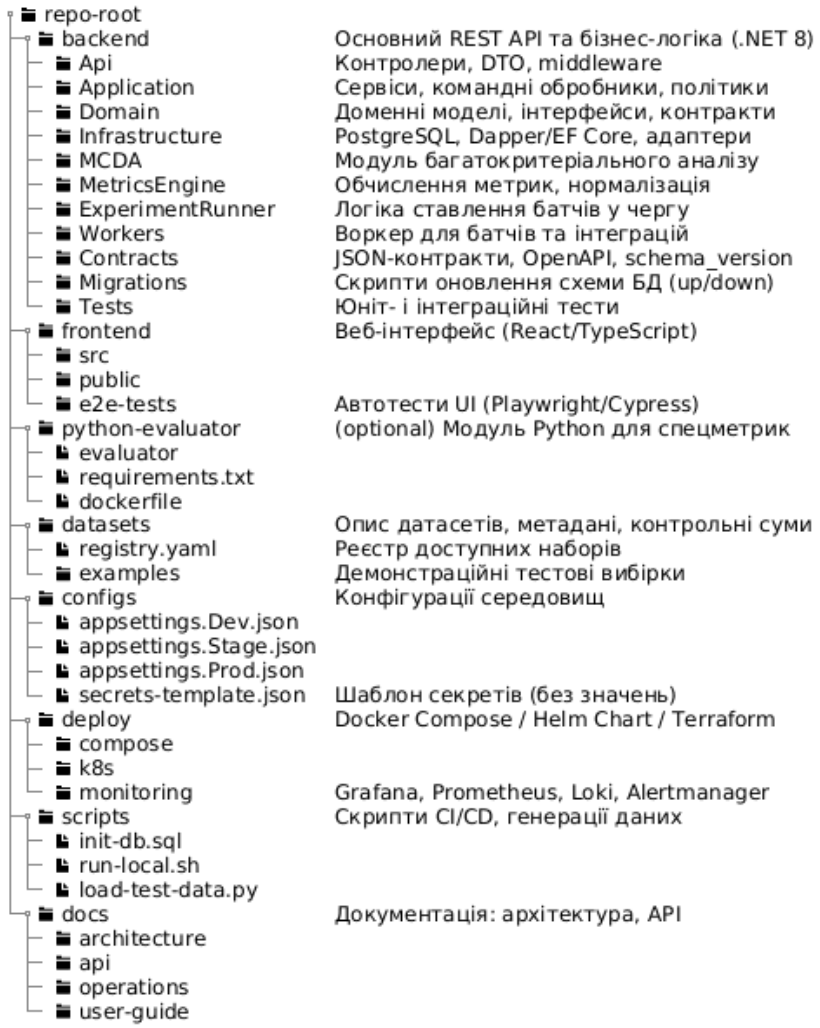


Рисунок 4.1. Структура файлової системи програмного комплексу системи

**Серверна частина** розташована в каталозі backend/ і реалізована на платформі .NET 8. Внутрішня структура відповідає принципу розділення на шари, що забезпечує слабку зв'язність компонентів:

– шар API (Api/) – містить контролери REST API, об’єкти передачі даних (DTO) та конфігурацію конвеєра обробки запитів (middleware, маршрутизація, політики автентифікації та авторизації).

– шар Домену (Domain/) – охоплює сутності предметної області, інтерфейси репозиторіїв та доменні сервіси. Цей шар є ядром системи й не має прямих залежностей від інфраструктури.

– шар Застосунку (Application/) – реалізує сценарії використання (Use Cases), обробники команд і запитів (CQRS), а також правила узгодження між доменом та зовнішніми сервісами.

– шар Інфраструктури (Infrastructure/) – містить реалізації інтерфейсів доступу до даних (EF Core), адаптери до зовнішніх мовних моделей, конектори до файлового сховища та засоби інтеграції з системами моніторингу.

Алгоритми багатокритеріального вибору та розрахунку метрик винесено в окремі логічні модулі (MCDA та MetricsEngine) всередині серверної частини, що спрощує їхнє незалежне тестування та повторне використання.

Для обробки ресурсомістких задач, таких як масові прогони оцінювання або генерація відповідей великими мовними моделями, використано *модуль фонового виконання* (Workers/). Винесення фонового виконавця в окремий процес дозволяє масштабувати обчислювальні ресурси незалежно від основного вебAPI та зберігати високу чутливість інтерфейсу користувача.

Опційно використовується окремий підпроєкт на Python (python–evaluator/), який виконує обчислення спеціалізованих метрик (наприклад, детектори токсичності або семантична близькість), реалізація яких є ефективнішою в екосистемі Python. Взаємодія з ним відбувається через внутрішній API або чергу повідомлень.

**Клієнтський інтерфейс** розташований у каталозі frontend/ і реалізований як односторінковий застосунок (SPA) на основі React і TypeScript. Структура фронтенду відображає основні функціональні області системи: роботу з наборами даних, конфігураціями, запусками та звітами. Взаємодія із сервером здійснюється через типізовані клієнти, згенеровані на основі OpenAPI–специфікації.

Для реалізації програмного комплексу використано сучасний набір бібліотек та фреймворків, що забезпечують продуктивність та надійність. Перелік ключових технологічних залежностей наведено в табл. 4.1.

Інфраструктурні конфігурації Файли, необхідні для розгортання, зосереджені в окремих каталогах (deploy/, configs/). Вони містять маніфести

Docker Compose для запуску всіх сервісів, а також налаштування для різних середовищ (development, stage, production). Секретні значення (ключі доступу, рядки підключення) не зберігаються в репозиторії, а передаються через змінні оточення, що забезпечує безпеку розробки.

Таблиця 4.1. Основні програмні залежності системи

| **Категорія** | **Бібліотека / Технологія** | **Призначення в системі** |
|---|---|---|
| Платформа | .NET 8, ASP.NET Core | Середовище виконання серверної частини |
| Робота з даними | Entity Framework Core, Dapper | Об’єктно–реляційне відображення (ORM) та виконання оптимізованих SQL–запитів |
| Архітектура | MediatR | Реалізація патерну Mediator для зменшення зв’язності між компонентами (CQRS) |
| Валідація | FluentValidation | Перевірка коректності вхідних даних та конфігурацій |
| Стійкість | Polly | Реалізація політик повторних спроб (Retry) та запобіжників (Circuit Breaker) при викликах зовнішніх API |
| Спостережуваність | Serilog, OpenTelemetry | Структуроване логування та розподілене трасування запитів |
| Клієнтська частина | React, TypeScript, MUI | Побудова компонентного інтерфейсу користувача |
| Інфраструктура | PostgreSQL, Docker | Основна реляційна СУБД та контейнеризація компонентів |

Така організація репозиторію забезпечує узгодженість із проєктними рішеннями розділу 3, високу відтворюваність експериментів та передбачуване розгортання в різних середовищах.

Для забезпечення контрольованого життєвого циклу програмного забезпечення, мінімізації ризиків при оновленні та гарантування стабільності роботи системи реалізовано стратегію розділення середовищ. Інфраструктура розгорнута у трьох ізольованих контурах: середовище розроблення (Development), середовище перевірки (Staging) та промислове середовище (Production). Характеристика кожного з них наведена в табл. 4.2.

Налаштування поведінки системи відокремлено від виконуваного коду згідно з принципами «дванадцятифакторного застосунку» (The Twelve–Factor App). У середовищі .NET це реалізовано через ієрархічну систему JSON–файлів конфігурації, розташованих у директорії configs/:

– appsettings.json – містить базові налаштування, спільні для всіх середовищ (наприклад, структура логування, ліміти на розмір запитів).

– appsettings.{Environment}.json – файли, що перевизначають специфічні параметри для конкретного середовища (наприклад, адреси зовнішніх сервісів або параметри кешування).

Таблиця 4.2. Характеристика середовищ розгортання системи

| **Середовище** | **Призначення** | **Конфігурація інфраструктури** | **Рівень логування** |
|---|---|---|---|
| Development (Dev) | Локальне розроблення, налагодження коду, попереднє тестування. | Локальні екземпляри сервісів або Docker–контейнери. База даних розгортається локально. | Debug / Information (деталізований вивід у консоль) |
| Staging (Stage) | Інтеграційне тестування, перевірка міграцій бази даних, узгодження взаємодії модулів. | Повна реплікація промислового середовища за допомогою Docker Compose. Використовуються тестові набори даних. | Warning / Error (структуровані логи) |
| Production (Prod) | Експлуатація системи кінцевими користувачами, збір реальних метрик. | Оптимізовані Docker–образи, налаштований HTTPS, регулярне резервне копіювання. | Error / Critical (мінімальний рівень шуму) |

Під час збирання та запуску застосунку відповідний файл конфігурації підтягується автоматично залежно від змінної середовища ASPNETCORE_ENVIRONMENT.

Критично важливі дані, такі як рядки підключення до бази даних, ключі доступу до API мовних моделей (LLM Providers) та ключі шифрування сесій, категорично виключено з системи контролю версій. Для роботи з ними застосовано гібридний підхід:

Локальне розроблення: тут використовується механізм .NET User Secrets («секрети користувача»), що зберігає конфіденційні дані у профілі користувача операційної системи, поза межами папки проєкту.

Контейнеризовані середовища (Stage/Prod): секрети передаються у застосунок виключно через змінні оточення (Environment Variables) під час запуску контейнера.

Для зручності розгортання в репозиторії передбачено файл–шаблон secrets–template.json, який містить перелік необхідних ключів без реальних значень. Це дозволяє адміністратору системи швидко налаштувати нове оточення, заповнивши шаблон актуальними обліковими даними, що

зберігаються у захищеному сховищі (наприклад, у менеджері паролів або CI/CD Secrets).

Такий підхід унеможливлює випадковий витік ключів доступу через репозиторій коду та забезпечує гнучкість при зміні постачальників послуг без необхідності перекомпіляції системи.

Для забезпечення цілісності програмного коду, мінімізації помилок під час інтеграції змін та стандартизації процесу випуску версій у проєкті використано практику безперервної інтеграції та розгортання (CI/CD). Автоматизація реалізована у вигляді сценарних конвеєрів (pipelines), які описані у конфігураційних файлах системи CI/CD та автоматично активуються під час змін у репозиторії (створення запиту на злиття змін або фіксація коміту в основній гілці).

Процес автоматизації логічно поділяється на два етапи:

– безперервну інтеграцію та контроль якості коду (Continuous Integration);

– підготовку артефактів для розгортання (Continuous Delivery).

*Етап 1. Безперервна інтеграція та контроль якості.*

Під час створення запиту на злиття змін (pull request) до основної гілки репозиторію автоматично запускається конвеєр перевірки коду. Його призначення полягає у тому, щоб гарантувати несуперечність нових змін із наявним функціоналом та відповідність прийнятим стандартам якості. Послідовність дій цього конвеєра наведено в табл. 4.3.

Таблиця 4.3 Етапи автоматизованої перевірки якості коду (CI–конвеєр)

| **Етап** | **Дії автоматизованої системи** | **Критерій успіху** |
|---|---|---|
| 1. Відновлення залежностей | Завантаження необхідних бібліотек (NuGet для .NET, npm для клієнтської частини) з фіксацією версій у файлах *lock*. | Успішне відновлення без конфліктів версій. |
| 2. Статичний аналіз | Перевірка коду засобами статичного аналізу та лінтерами на відповідність стильовим вимогам і виявлення потенційних помилок. | Відсутність критичних помилок рівня *Error*. |
| 3. Компіляція (Build) | Збирання серверної та клієнтської частини в режимі *Release*. | Успішне завершення компіляції без помилок. |
| 4. Тестування | Запуск модульних (unit) та інтеграційних тестів для основних сценаріїв. | Успішне проходження всіх передбачених тестів. |

Злиття змін до основної гілки дозволяється лише за умови успішного виконання всіх наведених етапів, тобто отримання конвеєром статусу «успішно». Таким чином, у гілку, яка використовується для формування релізів, потрапляє лише код, що пройшов повний цикл автоматизованих перевірок.

*Етап 2. Побудова та версіонування артефактів.*

Після затвердження змін і їх приєднання до основної гілки (main) запускається конвеєр побудови артефактів для розгортання. Оскільки система використовує контейнеризацію, основним артефактом є Docker–образи для серверної частини, фонового виконавця та допоміжних сервісів.

Процес збирання артефактів включає такі кроки:

Оптимізоване збирання образів – для зменшення розміру контейнерів застосовується багатостадійне збирання (multi–stage build): на проміжному етапі використовується повний SDK для компіляції, а у фінальний образ потрапляють лише скомпільовані бінарні файли та легковаге середовище виконання (.NET Runtime, інструменти для запуску React–застосунку тощо).

Маркування та версіонування: кожному зібраному образу присвоюється унікальний тег, що відповідає версії системи (наприклад, v1.0.1) або скороченому хешу коміту (sha–a1b2c3d). Це дає змогу однозначно встановити, яка саме версія коду використана в конкретному середовищі (Dev, Stage або Prod), і є критично важливим для відтворюваності експериментів.

Публікація в реєстрі контейнерів – після успішної побудови образи завантажуються до приватного реєстру контейнерів (Container Registry), звідки їх використовують сценарії розгортання для оновлення середовищ Stage і Prod.

У поєднанні з описаними далі процедурами розгортання така організація CI/CD–конвеєрів дозволяє виявляти більшість помилок ще на етапі розроблення, забезпечує контрольоване формування версій та гарантує, що до пілотного середовища потрапляють лише перевірені, відтворювані конфігурації системи.

Завершальним етапом життєвого циклу програмного забезпечення є його розгортання в цільових середовищах та забезпечення безперервного нагляду за

функціонуванням. Для цієї системи процедури розгортання, аварійного відновлення (відкоту) та моніторингу реалізовано з дотриманням принципів Infrastructure as Code (IaC) та спостережуваності (Observability), описаних у попередніх підрозділах.

Розгортання програмного комплексу здійснюється за допомогою контейнерної оркестрації. У рамках пілотної експлуатації як засіб оркестрації застосовано Docker Compose, що дає змогу декларувати потрібний стан інфраструктури (набір сервісів, мережеві зв’язки, томи даних) у вигляді YAML–маніфестів, розміщених у директорії deploy/. Однакові маніфести з незначними варіаціями параметрів використовуються для середовищ Dev, Stage та Prod, що зменшує ризик розбіжностей між ними.

Процес оновлення версії системи на сервері реалізовано як послідовність автоматизованих кроків:

Отримання артефактів – завантаження нових образів із заданими тегами версій, що були сформовані CI–конвеєром, із приватного реєстру контейнерів.

Міграція даних – запуск модуля міграцій у тимчасовому контейнері, який застосовує актуальні SQL–скрипти до схеми бази даних. Цей етап передує оновленню основного застосунку, що гарантує сумісність коду з оновленою структурою даних.

Ротація контейнерів – припинення роботи попередніх екземплярів сервісів після успішного завершення міграцій та запуск нових, сконфігурованих за допомогою актуальних змінних середовища. Контроль процесу здійснюється через вбудовані перевірки «готовності» (health checks), які гарантують спрямування трафіку виключно до працездатних екземплярів.

У такий спосіб конфігурація середовища та версія коду фіксуються в явному вигляді, що мінімізує ризик «дрейфу конфігурацій» між розробницьким і продуктивним оточеннями.

З метою забезпечення операційної стійкості та мінімізації впливу критичних помилок, виявлених під час експлуатації, розроблено регламент

аварійного повернення до попередньої стабільної версії (Rollback). Процедура відкоту змін реалізується на двох рівнях:

– рівень застосунку – завдяки суворому версіонуванню Docker–образів у реєстрі, повернення до стабільного стану здійснюється шляхом зміни тегів версій у декларативних маніфестах розгортання (наприклад, заміна v1.2.0 на v1.1.9) та повторного запуску оркестратора. Такий підхід дозволяє оперативно відновити перевірену конфігурацію без втручання в інфраструктурні налаштування.

– рівень даних – керування еволюцією схеми бази даних покладено на механізм міграцій Entity Framework Core. Оскільки кожна міграція містить як сценарій внесення змін (Up), так і сценарій їх скасування (Down), за потреби виконується послідовність зворотних міграцій. Це гарантує повну узгодженість структури даних із тією версією застосунку, на яку здійснюється повернення.

У комплексі ці процедури відповідають цілям щодо часу відновлення (RTO) та допустимої втрати даних (RPO), сформульованим у розділі 3.1.

Для забезпечення оперативного контролю за станом системи та своєчасного реагування на інциденти впроваджено комплексний стек спостережуваності (Observability), інтегрований в інфраструктуру розгортання. Система моніторингу базується на трьох взаємодоповнювальних компонентах:

Структуроване логування – усі архітектурні модулі (API, фонові виконавці, вебінтерфейс) налаштовано на генерацію журналів подій у машиночитному форматі JSON. Централізований збір та індексація логів здійснюється стеком Promtail та Loki. Використання наскрізних ідентифікаторів (request_id, trace_id) дозволяє відновити повний ланцюжок подій для кожного окремого запуску оцінювання або дії користувача, що значно спрощує діагностику помилок у розподіленому середовищі.

Збір метрик – серверну частину системи інструментовано засобами OpenTelemetry, що забезпечує експорт телеметрії до бази часових рядів Prometheus. Відстеження здійснюється у двох площинах: технічні показники за методологією Golden Signals (латентність, трафік, частка помилок 5xx,

насичення ресурсів CPU/RAM) та бізнес–метрики (кількість активних експериментів, середній час генерації відповіді моделлю, обсяг використаних токенів).

Візуалізація та сповіщення – поточний стан системи відображається на інформаційних панелях (дашбордах) Grafana, адаптованих для різних ролей користувачів. На основі даних Prometheus налаштовано правила автоматичного реагування: у разі виходу критичних показників за порогові значення (наприклад, недоступність бази даних або вичерпання дискового простору) компонент Alertmanager ініціює надсилання сповіщень відповідальним особам через визначені канали зв’язку.

Застосування описаних підходів до розгортання, відкоту та моніторингу забезпечує високу доступність і керованість програмного комплексу, а також прозорість обчислювальних процесів. Це, у свою чергу, створює необхідні передумови для проведення надійних експериментів та обґрунтованого порівняння методів узгодження поведінки мовних моделей.

### 4.2. Реалізація ключових модулів

Підсистема прийняття рішень реалізована як ізольований модуль backend/MCDA, який не має залежностей від інфраструктури або конкретних джерел даних. Така декомпозиція забезпечує «чистоту» обчислень, спрощує модульне тестування математичної логіки та відповідає вимогам відтворюваності експериментів, сформульованим у розділі 3.2. Архітектурно ядро побудоване як набір доменних сервісів, що послідовно перетворюють сирі значення метрик на впорядкований список альтернатив.

Логіка роботи модуля організована у вигляді конвеєра з чотирьох основних етапів.

Попередня фільтрація (thresholding) – на вхід ядра надходить матриця значень метрик для всіх кандидатів. Першим кроком застосовується компонент попередньої фільтрації (клас ThresholdFilter), який перевіряє відповідність показників обов’язковим обмеженням, визначеним у конфігурації оцінювання

(наприклад, частка токсичних відповідей не перевищує 0,01). Кандидати, що порушують хоча б один жорсткий поріг, виключаються з подальших обчислень, а у структурі результатів зберігається причина відхилення.

Нормалізація показників – оскільки вхідні метрики мають різну природу та одиниці вимірювання (секунди, відсотки, умовні бали), тому для їх подальшого поєднання застосовується сервіс нормалізації (MinMaxNormalizer). Усі значення перетворюються до безрозмірного діапазону [0;1] з урахуванням напрямку оптимізації критеріїв: для показників типу «більше – краще» застосовується пряма мін–макс нормалізація, для показників типу «менше – краще» (затримка, вартість) – обернена. У реалізації передбачено перевірку на вироджені випадки (однакові значення для всіх кандидатів), а також можливість подальшого розширення іншими схемами нормалізації.

Ранжування альтернатив – інтегральна оцінка кандидата (після нормалізації) обчислюється сервісом зваженого скорингу (WeightedSumScorer), який реалізує метод зваженої суми. Для кожного кандидата формується зважена сума нормалізованих значень за всіма критеріями, де ваги беруться з конфігурації оцінювання та відображають пріоритети користувача. Результатом цього етапу є колекція об’єктів RankedCandidate, упорядкована за спаданням підсумкового бала;додатково зберігаються проміжні компоненти вкладу окремих критеріїв у загальний результат.

Аналіз чутливості – для оцінювання стійкості рекомендації до змін у пріоритетах критеріїв використовується компонент аналізу чутливості (SensitivityAnalyzer). Він виконує серію симуляцій, у межах яких ваги критеріїв варіюються в невеликому діапазоні (орієнтовно +/– 5–10 % від базових значень), після чого повторно обчислюються рейтинги кандидатів. На основі частоти збереження лідерства базового кандидата формується показник «рівня впевненості» у рекомендації, а також, за потреби, виявляються альтернативні конфігурації ваг, за яких можливі інші рішення.

Запропонована декомпозиція дає змогу гнучко змінювати або розширювати алгоритми багатокритеріального вибору. Зокрема, метод зваженої

суми може бути замінений на альтернативні підходи (наприклад, TOPSIS або інші MCDA–методи) шляхом підміни реалізації інтерфейсу IScoringStrategy без необхідності модифікації решти підсистеми. Це спрощує експериментальне порівняння різних стратегій узгодження поведінки моделей та підтримує еволюційний розвиток системи.

За організацію процесу виконання експериментів відповідає модуль backend/ExperimentRunner, основним призначенням якого є оркестрація взаємодії між базою даних конфігурацій та зовнішніми виконавцями – API мовних моделей або обчислювальними воркерами. Архітектурно модуль реалізує шаблон «Виробник – Споживач» (Producer – Consumer), що дає змогу асинхронно обробляти великі множини завдань без блокування основного потоку виконання вебзастосунку та зберігати чутливість інтерфейсу користувача.

Для оптимізації мережевої взаємодії та зниження накладних витрат застосовується механізм декомпозиції прогону на окремі пакети (батчі) фіксованого або керованого розміру. Масив тестових завдань розбивається на послідовність таких пакетів;кожен пакет обробляється як атомарна одиниця роботи й має власний стан у таблиці batches. Це дозволяє ефективно використовувати багатопотоковість, точніше відстежувати прогрес виконання та будувати агреговані показники тривалості й успішності на рівні окремих батчів.

Описаний підхід тісно пов’язаний із вбудованою системою керування інтенсивністю запитів (rate limiting). Оскільки зовнішні постачальники мовних моделей накладають обмеження на кількість запитів або токенів за одиницю часу, модуль застосовує адаптивний обмежувач навантаження. Кількість одночасних потоків виконання й частота звернень динамічно регулюються залежно від поточного стану черги та отриманих від сервісу відповідей, що запобігає помилкам перевантаження і забезпечує рівномірне використання виділених квот.

Важливим аспектом реалізації є забезпечення відмовостійкості під час роботи з потенційно ненадійними зовнішніми мережевими сервісами. За

допомогою бібліотеки Polly у модулі реалізовано політики стійкості: для обробки короткочасних збоїв застосовується механізм повторних спроб (policy типу Retry) з експоненційною затримкою та випадковим зміщенням, а у випадку тривалої недоступності сервісу – шаблон «Запобіжник» (Circuit Breaker), який тимчасово блокує нові звернення й переводить відповідні батчі в очікувальний стан.

Додаткову надійність забезпечує механізм контрольних точок. Після завершення обробки кожного пакета результати негайно фіксуються в базі даних як ідемпотентні записи (за ключем run_id + batch_id), а статус батча оновлюється. У разі аварійної зупинки сервера або примусового перезапуску експеримент може бути відновлено з останньої успішно обробленої контрольної точки, без повторного виконання вже завершених обчислень. Це мінімізує втрати часу й обчислювальних ресурсів і відповідає вимогам відтворюваності, поставленим у попередньому розділі.

Центральним елементом забезпечення гнучкості системи є підсистема керування метаданими, яка реалізує реєстри методів узгодження та наборів даних. Вона пов’язана зі схемою бази даних (таблиці candidates, candidate_params, datasets, dataset_versions) і використовується всіма модулями, що беруть участь в організації експериментів і подальшому аналізі результатів.

Програмна реалізація реєстру методів базується на гнучкій схемі зберігання. Оскільки різні підходи (інструкційне налаштування, навчання з підкріпленням, «конституційний» ШІ тощо) потребують відмінних наборів гіперпараметрів, для їх опису використано тип поля JSONB у PostgreSQL. Це дає змогу зберігати специфічні конфігурації, зокрема шаблони системних промптів, коефіцієнти штрафів, параметри температури або обмеження довжини відповіді, у структурованому, але нефіксованому форматі, не перевантажуючи схему надмірною нормалізацією. На рівні серверної частини застосунку роботу з цими параметрами організовано за шаблоном «Стратегія»: для кожного класу методів реалізується власна стратегія формування запиту до моделі, яка обирається динамічно на основі запису в реєстрі.

Керування наборами даних спроєктовано з пріоритетом забезпечення наукової відтворюваності експериментів. Реалізація чітко розділяє поняття абстрактного набору (datasets) та його конкретної версії (dataset_versions). Будь–які зміни в складі тестових завдань або еталонних відповідей приводять до створення нової незмінної версії (immutable version), яка ідентифікується унікальною контрольною сумою й номером версії. Під час запуску експерименту система фіксує посилання саме на конкретну версію набору даних;таким чином забезпечується можливість точного повторення тесту навіть через тривалий час і на іншому середовищі.

Процес обчислення показників якості та безпеки зосереджено в окремому модулі MetricsEngine, побудованому за принципом конвеєрної обробки. Сирі відповіді моделей, збережені у вигляді артефактів і записів item_results, послідовно проходять через набір обчислювачів, кожен з яких відповідає за окремий аспект оцінювання. Для простих детермінованих метрик – таких як час відгуку (latency), довжина відповіді в токенах, відповідність формату JSON або наявність обов’язкових полів – обчислення виконуються синхронно безпосередньо в середовищі .NET із подальшим збереженням агрегованих значень у таблиці metric_results.

Для складніших семантичних метрик, зокрема оцінки токсичності, упередженості, правдоподібності або відповідності еталону, модуль формує асинхронні запити до спеціалізованого сервісу python–evaluator або використовує підхід «модель як суддя» (LLM–as–a–Judge), звертаючись до зовнішніх API. Отримані значення нормалізуються, позначаються інформацією про спосіб оцінювання (автоматичний метод, окрема модель–суддя, участь людини–рецензента) та зберігаються в нормалізованому вигляді. Це створює узгоджену базу показників, яка надалі використовується ядром багатокритеріального вибору для побудови матриць рішень і формування підсумкових рекомендацій.

Організація взаємодії з персистентним сховищем реалізована через шар доступу до даних, побудований на основі патерну «Репозиторій» (Repository).

Такий підхід забезпечує абстрагування бізнес–логіки від деталей реалізації конкретної системи керування базами даних та полегшує подальшу еволюцію схеми. Для операцій, що змінюють стан системи (команди), використано ORM Entity Framework Core, яка забезпечує транзакційну цілісність, автоматичне відстеження змін та знижує ризики SQL–ін'єкцій. Для побудови аналітичних зрізів та читання великих масивів метрик (запити) застосовано мікро–ORM Dapper, що дає змогу виконувати оптимізовані «сирі» SQL–запити з мінімальними накладними витратами на мапінг об'єктів. Особливістю реалізації є гібридна схема зберігання: структуровані метадані зберігаються в реляційних таблицях, тоді як специфічні параметри методів узгодження та результати безпекових перевірок серіалізуються у формат JSONB, що забезпечує гнучкість моделі даних без необхідності частих міграцій схеми.

Для забезпечення незалежності програмного комплексу від конкретних постачальників великих мовних моделей взаємодію із зовнішніми API організовано через шар інтеграційних адаптерів (конекторів). Усі адаптери реалізують уніфікований програмний інтерфейс, який визначає стандартизовані контракти для надсилання промптів, отримання відповідей та обліку використаних токенів. Завдяки цьому підключення нового провайдера (наприклад, перехід із хмарного API на локальний сервер інференсу) зводиться до реалізації нового класу адаптера без змін у доменній логіці та ядрових компонентах. На цьому ж рівні інкапсульовано обробку мережевих помилок і специфічних кодів відповіді зовнішніх сервісів, що підвищує стабільність роботи системи незалежно від стану зовнішніх залежностей.

Підсистема звітування відповідає за інтерпретацію накопичених експериментальних даних та їх подання у формі, придатній для прийняття рішень. Генерація звітів виконується на основі агрегованих метрик, збережених у базі даних (таблиці результатів оцінювання та ранжування), і включає побудову порівняльних таблиць та графічних матеріалів (зокрема діаграм компромісів «якість – безпека»). Передбачено формування двох основних типів вихідних документів: деталізованих технічних протоколів для інженерів і

дослідників та узагальнених управлінських зведень для менеджерів продукту й осіб, що приймають рішення. Важливою вимогою, реалізованою у цьому модулі, є простежуваність (traceability): кожен звіт містить однозначні посилання на версії використаних наборів даних та конфігурацій оцінювання, а також ідентифікатор запуску, що дає змогу відтворити умови отримання результатів. Підтримується експорт звітів у поширені формати (PDF, CSV, JSON), що спрощує зовнішній аналіз, інтеграцію з іншими системами та архівацію результатів.

### 4.3. Тестування та результати

Забезпечення надійності інформаційної системи реалізовано на основі комплексної стратегії тестування, що спирається на класичну «піраміду тестування» та принципи безперервної інтеграції. Головною метою обраного підходу є гарантування коректності роботи математичного ядра системи, стабільності інтеграційних взаємодій із зовнішніми постачальниками даних і відтворюваності результатів експериментів. Процес верифікації охоплює всі архітектурні шари – від окремих методів бізнес–логіки до наскрізних сценаріїв користувача.

Фундаментом стратегії є модульне (unit) тестування, яке покриває критично важливі компоненти. Найвищий пріоритет надано перевірці модуля багатокритеріального аналізу (MCDA) та підсистеми розрахунку метрик, оскільки саме вони містять найбільш складну обчислювальну логіку. Для цих компонентів застосовується підхід тестування «білої скриньки» з акцентом на покритті всіх гілок виконання (branch coverage). Набори тестових випадків включають перевірку граничних значень, обробку некоректних вхідних даних і верифікацію математичної точності на еталонних наборах. Для ізоляції тестованого коду від зовнішніх залежностей (бази даних, файлового сховища, мережевих викликів) активно використовуються об’єкти–імітатори (mocks, stubs).

Наступним рівнем виступає інтеграційне тестування, спрямоване на перевірку коректності взаємодії між компонентами системи та зовнішнім середовищем. На цьому етапі верифікується робота шару доступу до даних, коректність формування SQL–запитів та відображення об’єктів доменної моделі на схему бази даних. Окрема увага приділяється адаптерам до API великих мовних моделей. Щоб зменшити залежність від стабільності мережі та вартості зовнішніх викликів під час автоматизованих прогонів, для таких тестів використовуються попередньо записані відповіді реальних сервісів (механізм «record–replay») або локальні емулятори API.

Вершиною піраміди є системне та наскрізне (end–to–end, E2E) тестування, яке перевіряє роботу системи в цілому з погляду кінцевого користувача. Сценарії цього рівня охоплюють повний життєвий цикл: створення конфігурації оцінювання, запуск експерименту, моніторинг прогресу, перегляд та експорт згенерованого звіту. Тести виконуються у середовищі, максимально наближеному до виробничого, що дозволяє виявляти проблеми, пов’язані з конфігурацією, правами доступу, маршрутизацією запитів та інтеграцією клієнтської й серверної частин.

Контроль повноти тестування здійснюється за допомогою метрик покриття коду (code coverage). Цільові значення диференційовано залежно від критичності модулів: для ядра прийняття рішень встановлено вимогу покриття не менше 90 %, тоді як для інфраструктурного коду та контролерів API допустимим є нижчий поріг, достатній для перевірки основних сценаріїв успіху та обробки типових помилок. Такий підхід дає змогу раціонально розподілити зусилля на тестування та забезпечити високу якість ключових функцій системи без надмірних витрат ресурсів.

Реалізація модульного (юніт) тестування зосереджена на перевірці ізольованих фрагментів бізнес–логіки, насамперед у модулях MCDA та MetricsEngine. Для написання тестів використано фреймворк xUnit. Ключовою особливістю цього рівня є повна ізоляція від зовнішніх залежностей: взаємодія з базою даних, файловою системою та мережевими інтерфейсами замінена на

використання об’єктів–імітаторів (mocks), створених за допомогою бібліотеки Moq. Такий підхід дає змогу перевіряти коректність математичних алгоритмів нормалізації та ранжування в детермінованих умовах, охоплюючи граничні випадки (ділення на нуль, порожні вхідні масиви, екстремальні значення метрик) без витрат часу на операції вводу–виводу.

Інтеграційне тестування спрямоване на перевірку коректності роботи компонентів у зв’язці, зокрема взаємодії контролерів API із шаром доступу до даних. Для цього використано механізм WebApplicationFactory, який розгортає тестовий екземпляр вебзастосунку в пам’яті. Для забезпечення чистоти експериментів застосовано бібліотеку Testcontainers, що автоматично піднімає тимчасовий Docker–контейнер із базою даних PostgreSQL для кожного набору тестів. Це гарантує виконання перевірок на реальній СУБД, з валідацією коректності SQL–запитів, міграцій схеми та обмежень цілісності, але без впливу на робочі дані та без взаємної залежності тестів. Окрема увага приділяється тестуванню конвеєра обробки HTTP–запитів, включно з роботою проміжного програмного забезпечення (middleware) для обробки винятків та валідації вхідних DTO–об’єктів.

Функціональне (наскрізне) тестування реалізовано для верифікації ключових сценаріїв роботи користувача з вебінтерфейсом. За допомогою інструмента Playwright автоматизовано проходження критичних сценаріїв (happy paths): створення нової конфігурації оцінювання, запуск експерименту, відстеження прогресу та перегляд згенерованого звіту. Тести імітують дії реального користувача в браузері – введення даних, натискання на елементи керування, навігацію між екранами – і перевіряють коректність відображення елементів інтерфейсу, реакцію системи на дії користувача та цілісність даних під час обміну між клієнтом і сервером. Ці перевірки виконуються в середовищі CI/CD на фінальному етапі збирання, слугуючи додатковим бар’єром для запобігання регресіям перед розгортанням системи.

Оцінка продуктивності системи під навантаженням проводилася з метою перевірки стабільності роботи фонових виконавців, коректності механізмів

керування чергами та відповідності системи заданим обмеженням постачальників зовнішніх сервісів. Для моделювання високого навантаження застосовано інструмент k6, за допомогою якого відтворювалися сценарії одночасного запуску кількох масштабних експериментів із використанням великих наборів даних. Це створювало пікове навантаження на підсистему обробки черг і модуль взаємодії із зовнішніми API. Ключовим завданням навантажувального тестування було підтвердження ефективності внутрішнього обмежувача інтенсивності запитів (rate limiter): перевірялося, чи система коректно утримує вихідний трафік у межах заданих квот, не допускає неконтрольованого зростання споживання ресурсів та забезпечує стійку роботу процесів–воркерів упродовж тривалого часу.

Перевірка безпеки програмного забезпечення здійснювалася з урахуванням рекомендацій OWASP Top 10. На етапі розроблення було використано методи статичного аналізу коду (SAST), інтегровані у конвеєр CI/CD, що дало змогу автоматично виявляти потенційні вразливості, пов’язані з некоректною обробкою вхідних даних або використанням небезпечних бібліотек. Динамічний аналіз (DAST) проводився на розгорнутому тестовому середовищі з використанням сканера OWASP ZAP для виявлення типових вебвразливостей, зокрема міжсайтового скриптингу (XSS), підробки міжсайтових запитів (CSRF) та помилок конфігурації механізмів автентифікації.

Окрему увагу приділено верифікації механізмів авторизації та захисту чутливих даних. Тестування рольової моделі доступу підтвердило, що користувачі з обмеженими правами не мають можливості виконувати адміністративні операції або змінювати конфігурації, які їм не належать. Додатково було перевірено надійність системи маскування секретів: аналіз журналів подій засвідчив, що ключі доступу до зовнішніх мовних моделей та інші конфіденційні параметри не зберігаються й не відображаються у відкритому вигляді навіть у режимі налагодження. Сукупність навантажувальних і безпекових перевірок підтвердила здатність системи стабільно функціонувати в

умовах інтенсивного використання та відповідність базовим вимогам до інформаційної безпеки.

Для експериментальної перевірки системи було сформовано гетерогенний набір тестових даних, що поєднує загальнодоступні бенчмарки для оцінки якості відповідей та спеціалізовані набори для тестування безпеки. Вихідні дані попередньо нормалізовано та приведено до уніфікованого внутрішнього формату JSONL, де кожен запис містить вхідний запит (промпт), контекст, очікувану еталонну відповідь (за наявності) та метадані для категоризації. Розподіл завдань на категорії, такі як «корисність», «безпека» та «дотримання інструкцій», дав змогу перевірити коректність роботи модуля багатокритеріального аналізу за різних конфігурацій ваг і пріоритетів критеріїв.

Критично важливою вимогою до системи є забезпечення відтворюваності експериментів. Ця властивість реалізована через механізм жорсткої фіксації версій: кожен запуск оцінювання пов’язується з незмінним зліпком конфігурації та конкретною версією набору даних, що ідентифікується контрольною сумою. Для мінімізації впливу стохастичної природи великих мовних моделей у параметрах запитів фіксується значення зерна випадковості або, для детермінованих сценаріїв, встановлюється низька або нульова температура генерації. У такій постановці повторний запуск експерименту за ідентичних умов забезпечує статистично близькі (або, за детермінальних налаштувань, тотожні) результати, що є необхідною передумовою коректного порівняння методів узгодження.

Результати оцінювання подаються у вигляді інтерактивних аналітичних звітів. Система автоматично агрегує сирі вимірювання в статистичні показники (середні значення, медіани, перцентилі) та візуалізує їх за допомогою діаграм розсіювання та інших графічних представлень. Це дає змогу наочно оцінювати компроміси між потенційно конфліктними критеріями, зокрема між рівнем безпеки та корисністю відповіді. Підсумковий рейтинг методів супроводжується текстовим поясненням, сформованим на основі ваг критеріїв, що підвищує

інтерпретованість рекомендацій для користувача й полегшує прийняття управлінських рішень.

Водночас під час тестування було виявлено низку обмежень поточної реалізації. По–перше, валідність автоматичних оцінок суттєво залежить від якості та можливих упереджень моделі, яка виконує роль «судді» (підхід LLM–as–a–Judge). По–друге, технічні обмеження пов’язані з пропускною здатністю та латентністю зовнішніх комерційних API, що впливає на загальну тривалість проходження експериментів і верхні межі масштабу тестування. По–третє, зберігається методологічний ризик «забруднення» даних (data contamination), коли окремі тестові приклади можуть входити до навчальних вибірок використовуваних моделей, що потенційно призводить до завищення показників якості. Зазначені чинники розглядаються як напрямки подальшого вдосконалення системи та уточнення методики оцінювання.

### 4.4. Керівництво користувача

Розгортання програмного комплексу спроєктовано за принципом контейнеризації, що забезпечує ізоляцію залежностей, відтворюваність середовища та швидкий запуск у будь–якій інфраструктурі, де підтримується Docker. Перед початком встановлення адміністратор повинен переконатися в наявності базового програмного забезпечення: середовища виконання контейнерів (Docker Engine) та інструменту оркестрації Docker Compose.

Процедура встановлення розпочинається з отримання актуальної версії вихідного коду або маніфестів розгортання з репозиторію системи. Першим кроком початкового налаштування є створення файлів конфігурації на основі наданих шаблонів. На цьому етапі вносяться параметри, специфічні для конкретної інсталяції: рядки підключення до бази даних, секретні ключі для підпису токенів автентифікації, а також API–ключі зовнішніх постачальників мовних моделей. З міркувань безпеки ці значення не входять до вихідного коду та не зберігаються у репозиторії;їх необхідно вказати явно у файлах налаштувань або змінних середовища.

Після заповнення конфігураційних параметрів здійснюється запуск системи за допомогою команди оркестрації в терміналі, яка ініціює завантаження необхідних контейнерних образів і старт пов’язаних сервісів: серверної частини, клієнтського вебінтерфейсу, бази даних PostgreSQL та фонових виконавців. Під час першого запуску автоматично активується механізм міграцій, який створює необхідну структуру таблиць у базі даних і наповнює системні довідники початковими значеннями (зокрема, переліком ролей користувачів та базовим набором підтримуваних метрик).

Завершення встановлення підтверджується доступністю вебінтерфейсу за адресою локального хоста або виділеного домену, визначеного в конфігурації розгортання. Після завантаження сторінки входу система готова до експлуатації;для роботи кінцевого користувача не вимагається встановлення додаткових бібліотек чи спеціалізованих середовищ виконання на робочій станції, оскільки вся бізнес–логіка та обчислення виконуються на сервері.

Доступ до функціональних можливостей системи регулюється рольовою моделлю керування доступом (RBAC), що забезпечує розмежування прав залежно від завдань користувача. Початок роботи передбачає проходження процедури автентифікації через захищену форму входу, де користувач вводить облікові дані або, за наявності інтеграції, використовує корпоративний обліковий запис (SSO). Після успішної авторизації інтерфейс автоматично адаптується до призначеної ролі: аналітики отримують доступ до інструментів налаштування конфігурацій оцінювання та перегляду метрик, рецензенти (контролери якості/безпеки) – до модулів ручної перевірки і рев’ю спірних прикладів, а адміністратори – до керування користувачами, ролями та системними параметрами.

Ключовим етапом підготовки до експерименту є створення конфігурації оцінювання, яка визначає правила вимірювання та порівняння кандидатних методів. У відповідному розділі інтерфейсу користувач формує набір активних метрик, обираючи їх із системного довідника. Для кожного показника може бути задано «жорсткі» пороги (thresholds) – граничні значення безпеки, часу відгуку

або відповідності формату, порушення яких призводить до автоматичної дискваліфікації кандидата ще до етапу багатокритеріального аналізу. Такий механізм дає змогу відсіювати неприйнятні рішення на ранніх стадіях, зменшуючи обсяг подальшого аналізу для експерта.

Окремий блок налаштувань присвячено параметрам багатокритеріального вибору. Користувач задає відносну важливість критеріїв, призначаючи їм числові ваги (наприклад, надаючи вищий пріоритет безпеці порівняно зі швидкістю), та обирає метод нормалізації даних відповідно до прийнятої методики. Сформована конфігурація зберігається як незмінна версія з унікальним ідентифікатором, що прив’язується до всіх запусків, у яких вона використовується. Це гарантує можливість точного відтворення будь–якого експерименту в майбутньому та забезпечує коректну порівнянність результатів різних прогонів за однакових умов.

Ініціювання експерименту здійснюється через спеціалізований інтерфейс запуску, де користувач об’єднує попередньо підготовлені компоненти в єдиний сценарій оцінювання. У відповідній формі обираються кандидатні моделі (або їхній набір для порівняння), фіксується версія набору даних та визначається конфігурація оцінювання. Перед початком виконання система автоматично проводить валідацію доступності ресурсів та квот зовнішніх API. Після успішної перевірки запуску присвоюється унікальний ідентифікатор, і він передається до черги на обробку фоновими виконавцями.

У процесі виконання прогону користувач має можливість спостерігати за його станом у режимі, наближеному до реального часу, через панель моніторингу. Інтерфейс відображає відсоток оброблених завдань, поточний статус виконавців, а також технічні показники, такі як середня швидкість генерації відповідей та кількість помилок. Це дозволяє оперативно виявляти аномалії (наприклад, збої мережевого з’єднання або перевищення лімітів API) та, за потреби, примусово зупиняти експеримент чи коригувати навантаження без очікування повного завершення циклу.

Після закінчення обчислень система автоматично формує сторінку результатів. Інтерфейс пропонує дворівневе подання даних: зведену таблицю з агрегованими метриками якості, безпеки та ефективності для кожного кандидата, а також детальне відображення окремих прикладів. Аналітичний модуль візуалізує порівняння методів за допомогою діаграм, акцентуючи увагу на виявлених компромісах між критеріями, і додатково виділяє рекомендований підхід, визначений алгоритмом багатокритеріального вибору. Користувач може застосовувати фільтри, щоб детальніше проаналізувати випадки, у яких модель порушила правила безпеки або не дотрималася вимог до формату відповіді.

Для документування результатів та їх подальшого використання передбачено функцію експорту. Система дозволяє згенерувати звіт у форматі PDF, який містить «паспорт» експерименту (ідентифікатор прогону, версії датасетів і конфігурацій), підсумкові таблиці, графіки та текстове обґрунтування вибору рекомендованого методу. Для глибшого аналізу в зовнішніх інструментах (наприклад, Excel або Python/Pandas) доступне вивантаження повного масиву метрик та, за наявності відповідних прав доступу, «сирих» відповідей моделей у структурованих форматах CSV або JSON. Це забезпечує можливість незалежної перевірки отриманих результатів та їхнього подальшого використання в дослідницьких цілях.

Для забезпечення безперервної експлуатації системи користувачеві доцільно володіти базовими методами діагностики та усунення типових позаштатних ситуацій. Значна частина повідомлень про помилки пов’язана з взаємодією із зовнішніми постачальниками мовних моделей. Отримання кодів, що свідчать про перевищення лімітів частоти запитів або вичерпання квоти токенів, зазвичай означає необхідність перевірки балансу облікового запису на стороні провайдера або зменшення параметра паралельності у налаштуваннях запуску. У випадках, коли система фіксує тайм–аути мережевого з’єднання, рекомендовано переглянути панель стану та журнали подій, а за відсутності внутрішніх збоїв – повторити прогін пізніше, оскільки вбудовані механізми

стійкості автоматично обробляють лише короткочасні відмови зовнішніх сервісів.

Інша поширена категорія проблем стосується валідації даних та конфігурацій. Якщо запуск експерименту блокується на етапі ініціалізації, причиною найчастіше є невідповідність структури завантаженого набору даних очікуваній схемі, зокрема відсутність обов’язкових полів чи порушення формату JSON. У ситуаціях, коли прогін завершується успішно, але в підсумковому звіті відсутні рекомендовані кандидати, варто насамперед перевірити встановлені порогові значення безпеки та якості. Надмірно жорсткі обмеження можуть призвести до автоматичного відсіювання всіх варіантів;у такому разі доцільно проаналізувати проміжні метрики та поступово пом’якшити критерії відбору, контролюючи вплив цих змін на підсумковий рейтинг.

Дотримання правил інформаційної безпеки є обов’язковою умовою роботи з системою. Користувачам забороняється вводити ключі доступу до API (API keys) у текстові поля описів проєктів, коментарі або передавати їх через незахищені канали зв’язку;для цього передбачені спеціальні поля конфігурації, вміст яких маскується системою і зберігається у захищеному сховищі секретів. Окремо діють обмеження щодо приватності даних: під час формування тестових наборів необхідно уникати включення персональної інформації (PII) та комерційних таємниць, оскільки відповідні запити й відповіді можуть передаватися на обробку зовнішнім постачальникам моделей. Завершуючи сеанс роботи, користувач повинен виходити із системи через стандартну процедуру виходу, щоб запобігти несанкціонованому доступу до результатів експериментів і налаштувань під своїм обліковим записом, особливо у випадку спільного використання робочої станції.

Отже, досліджено і здійснено програмну реалізацію системи на базі архітектури модульного моноліту та налаштували автоматизовану інфраструктуру розгортання, що гарантує відтворюваність експериментів. Комплексне тестування підтверджує коректність роботи алгоритмів багатокритеріального вибору, стабільність системи під навантаженням та її

відповідність вимогам безпеки. Розроблене керівництво користувача регламентує порядок налаштування, запуску оцінювань та інтерпретації отриманих результатів.

**Висновки до Розділу 4.**

Узагальнюючи матеріали розділу 4, можна сформулювати кілька ключових висновків щодо реалізації, тестування та експлуатації розробленої системи. Нижче наведено основні з них.

По–перше, реалізовано повноцінний програмний комплекс на базі модульного моноліту й архітектури «порти й адаптери» у форматі monorepo: серверна частина на .NET 8, фронтенд на React/TypeScript, фонові воркери та опційний Python–модуль для спеціалізованих метрик, що забезпечує чітке розділення відповідальностей і ізоляцію доменної логіки від інфраструктури.

По–друге, побудовано кероване інфраструктурне оточення з трьома середовищами (Dev/Stage/Prod), окремими конфігураціями, безпечним керуванням секретами та автоматизованими CI/CD–конвеєрами, які охоплюють відновлення залежностей, статичний аналіз, збирання, тестування, побудову й версіонування Docker–образів та їх розгортання.

По–третє, реалізовано ключові доменні модулі – ядро багатокритеріального вибору (MCDA), оркестратор експериментів (ExperimentRunner), підсистему обчислення метрик (MetricsEngine), реєстри методів і датасетів, шар репозиторіїв та адаптери до зовнішніх LLM–провайдерів – у спосіб, що дозволяє масштабувати систему, змінювати стратегії оцінювання та підключати нових постачальників без переробки ядра.

По–четверте, забезпечено відмовостійке розгортання й експлуатацію: застосовано контейнерну оркестрацію (Docker Compose), міграції схеми БД з можливістю відкату, механізми rollback на рівні застосунку та даних, а також повноцінний стек спостережуваності (структуровані логи, метрики, дашборди й алерти), що мінімізує ризики збоїв і «дрейфу конфігурацій».

По–п’яте, впроваджено комплексну стратегію тестування (unit, інтеграційні, E2E, навантажувальні та безпекові тести), яка показала коректність математичних алгоритмів, стабільність роботи під навантаженням, коректність інтеграцій з БД та зовнішніми API, а також відповідність базовим вимогам до інформаційної безпеки (OWASP Top 10, захист секретів, перевірка ролей).

По–шосте, механізми версіонування конфігурацій, датасетів та запусків, а також побудова трасованих звітів з експортом у стандартні формати забезпечують наукову відтворюваність експериментів і прозорість прийняття рішень щодо вибору методів узгодження поведінки моделей.

По–сьоме, розроблене керівництво користувача та рольова модель доступу (RBAC) формалізують порядок розгортання, налаштування, запуску експериментів, аналізу результатів і реагування на типові помилки, що робить систему придатною для практичного використання аналітиками, рецензентами та адміністраторами без заглиблення в деталі внутрішньої реалізації.

По–восьме, під час тестування і експлуатаційних прогонів ідентифіковано низку обмежень (залежність від упереджень моделей–суддів, ліміти й латентність зовнішніх API, ризик «забруднення» даних), які не нівелюють корисність системи, але визначають пріоритетні напрями її подальшого розвитку та уточнення методики оцінювання.

# ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальну науково–практичну задачу підвищення обґрунтованості та ефективності вибору методів узгодження поведінки великих мовних моделей шляхом створення спеціалізованої інформаційної системи. Запропонований підхід забезпечує відтворюваний, прозорий і економний процес оцінювання методів узгодження з урахуванням якості, безпеки, вартості та обмежень даних.

В результаті роботи було створено інформаційну систему для підвищення обґрунтованості та ефективності вибору методів узгодження поведінки LLM за допомогою інформаційної системи з багатокритеріальним аналізом рішень і стандартизованими процедурами оцінювання – досягнута повністю, усі поставлені завдання виконано. Основні результати полягають у такому:

1. Виконано системний огляд підходів до узгодження та типів даних, що для них застосовуються. У розділі 1 проаналізовано сучасні класи методів узгодження поведінки LLM (інструкційне донавчання, підходи на основі преференцій, правила/guardrails, техніки безпеки тощо) та типи даних для їх застосування, а для узагальнення виконано порівняння підходів у вигляді таблиці.

2. Сформовано критерії та метрики оцінювання якості, безпеки, ефективності, надійності та вартості. Визначено набір ключових критеріїв (якість, безпека, справедливість/відсутність упередженості, керованість, надійність, ефективність і вартість) та описано принципи їх вимірювання на стандартизованих наборах завдань.

3. Побудовано концептуальну та інформаційну модель системи, визначено життєвий цикл експериментів і артефактів. У підрозділі 2.1 сформовано концептуальну модель із центральною сутністю “експеримент”, а також описано структури збереження конфігурацій, запусків (runs) та артефактів/результатів оцінювання (метрики, їх значення, зв’язки між експериментами та запусками).

4. Розроблено модель багатокритеріального вибору та методику оцінювання з чутливісним аналізом. Формалізовано MCDA-модель (адитивна модель зваженої суми) з нормалізацією метрик до єдиної шкали та двокроковою схемою вибору (фільтр порогів + ранжування), а також визначено протокол оцінювання і підхід до аналізу стійкості рекомендацій (чутливісний аналіз).

5. Підготовлено специфікацію вимог до програмного забезпечення. Для цього сформульовано функціональні та нефункціональні вимоги до системи: продуктивність, надійність, масштабованість, спостережуваність, безпека, відтворюваність, вимоги до API та середовищ розгортання. Ця специфікація стала основою для подальших архітектурних і проєктних рішень.

6. Спроєктовано архітектуру та інтерфейси системи. Розроблено архітектуру модульного моноліту на платформі .NET 8 із застосуванням принципів «чистої архітектури» та патерну «порти й адаптери». Описано схему бази даних, UML–діаграми, REST–API й контракти подій, модель безпеки (автентифікація, RBAC, аудит), а також структуру вебінтерфейсу для дослідників, інженерів і аналітиків.

7. Реалізовано прототип інформаційної системи та проведено тестування. Створено програмний комплекс на базі .NET і PostgreSQL з вебінтерфейсом, модулем оркестрації запусків, підсистемою збору та зберігання метрик, ядром багатокритеріального аналізу й модулем формування звітів. Розгорнуто інфраструктуру з використанням контейнеризації (Docker), декількох середовищ (Dev/Stage/Prod) та CI/CD–конвеєрів.

Комплексне тестування підтвердило коректність реалізованих алгоритмів нормалізації й ранжування, стійкість до збоїв, здатність обробляти великі обсяги даних, а також надійність інтеграцій з зовнішніми API мовних моделей.

Практичне значення одержаних результатів полягає у створенні інструментарію, який дозволяє дослідникам та інженерам:

– зменшити витрати часу й ресурсів на експерименти;

– уніфікувати процедури оцінювання методів узгодження поведінки LLM;

– документувати прийняті рішення з повним «слідом» конфігурацій і артефактів;

– підвищити прозорість і відтворюваність порівняння різних моделей та постачальників.

Перспективи подальших досліджень пов’язані з:

– розширенням набору методів багатокритеріального аналізу (зокрема, методи родини PROMETHEE та інші MCDA–підходи);

– інтеграцією з інструментами автоматичного добору гіперпараметрів і AutoML;

– підтримкою мультимодальних моделей (текст+зображення, аудіо тощо);

– масштабуванням системи до промислових сценаріїв із розподіленою обробкою та використанням кількох провайдерів LLM.

Таким чином, у роботі створено методичну та програмну основу для прозорого, відтворюваного й безпечного порівняння методів узгодження поведінки великих мовних моделей, яка може бути адаптована до конкретних прикладних задач і розвинена в наступних дослідженнях.

# СПИСОК ДЖЕРЕЛ ПОСИЛАННЯ

1. V ВСЕУКРАЇНСЬКА НАУКОВО–ПРАКТИЧНА КОНФЕРЕНЦІЯ «СУЧАСНІ ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В НАУЦІ ТА ОСВІТІ». Збірник тез. К.: ДУІКТ, 2025. URL: https://duikt.edu.ua/uploads/p_2779_68674368.pdf (дата звернення: 13.11.2025).

2. Бідюк, П. І., Тимощук, О. Л., Коваленко, А. Є., & Коршевнюк, Л. О. Системи і методи підтримки прийняття рішень: Підручник. Київ. КПІ ім. Ігоря Сікорського. 2022. 610с. URL: https://shron1.chtyvo.org.ua/Bidiuk_Petro/Systemy_i_metody_pidtrymky_pryiniattia_rishen.pdf (дата звернення: 13.11.2025).

3. Дмитренок, С & Кіршак, Х. ЗАСТОСУВАННЯ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ДЛЯ ІНТЕЛЕКТУАЛЬНОЇ ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ І УПРАВЛІННЯ ТЕХНОЛОГІЧНИМИ ПРОЦЕСАМИ В НАФТОГАЗОВИДОБУВНІЙ ГАЛУЗІ. Таврійський науковий вісник. Серія: Технічні науки. 85–102. 10.32782/tnv–tech.2025.3.10. 2025. URL: https://www.researchgate.net/publication/395073865_ZASTOSUVANNA_VELIKIH_MOVNIH_MODELEJ_DLA_INTELEKTUALNOI_PIDTRIMKI_PRIJNATTA_RISEN_I_UPRAVLINNA_TEHNOLOGICNIMI_PROCESAMI_V_NAFTOGAZOVIDOBUVNIJ_GALUZI (дата звернення: 13.11.2025).

4. Abeysinghe, Bhashithe & Circi, Ruhan. The Challenges of Evaluating LLM Applications: An Analysis of Automated, Human, and LLM–Based Approaches. 10.48550/arXiv.2406.03339. 2024 URL: https://www.researchgate.net/publication/381189969_The_Challenges_of_Evaluating_LLM_Applications_An_Analysis_of_Automated_Human_and_LLM–Based_Approaches (дата звернення: 13.11.2025).

5. Alok Abhishek, Lisa Erickson, Tushar Bandopadhyay. BEATS: Bias Evaluation and Assessment Test Suite for Large Language Models.31 Mar 2025. URL: https://arxiv.org/abs/2503.24310 (дата звернення: 13.11.2025).

6. Anghel C, Anghel AA, Pecheanu E, Craciun MV, Cocu A, Niculita C. PEARL: A Rubric–Driven Multi–Metric Framework for LLM Evaluation. Information. 2025;16(11):926. URL: https://www.mdpi.com/2078–2489/16/11/926 (https://doi.org/10.3390/info16110926 ) (дата звернення: 13.11.2025).

7. Bommasani, Rishi & Liang, Percy & Lee, Tony. Holistic Evaluation of Language Models. Annals of the New York Academy of Sciences. 1525. 10.1111/nyas.15007. 2023 URL: https://www.researchgate.net/publication/371046714_Holistic_Evaluation_of_Language_Models (дата звернення: 13.11.2025).

8. Courtney Patterson. Smartsheet Inc.. “Enterprise AI: Adoption, Risks, Use Cases, & Examples”. October 30, 2024 URL: https://www.smartsheet.com/content/enterprise–ai?srsltid=AfmBOopGrUqVJg4Vu_a4o7fDlAN8sx4jwg3rlCAiUIQZSYG9qIg4bJC1 (дата звернення: 13.11.2025).

9. Data Ethics in AI: 6 Key Principles for Responsible Machine Learning. July 16, 2024. URL: https://www.alation.com/blog/data–ethics–in–ai–6–key–principles–for–responsible–machine–learning/ (дата звернення: 13.11.2025).

10. Dave Davies. LLM evaluation: Metrics, frameworks, and best practices. URL: https://wandb.ai/onlineinference/genai–research/reports/LLM–evaluation–Metrics–frameworks–and–best–practices—VmlldzoxMTMxNjQ4NA (дата звернення: 13.11.2025).

11. Deliberative оцінювання: reasoning enables safer language models.December 20, 2024. URL: https://openai.com/index/deliberative–оцінювання/ (дата звернення: 13.11.2025).

12. Ethan M. Rudd, Christopher Andrews, Philip Tully. A Practical Guide for Evaluating LLMs and LLM–Reliant Systems.21 Jul 2025. URL: https://arxiv.org/abs/2506.13023 (дата звернення: 13.11.2025).

13. European Data Protection Board (EDPB). AI Privacy Risks & Mitigations – Large Language Models (LLMs). 2025. URL:

https://www.edpb.europa.eu/system/files/2025–04/ai–privacy–risks–and–mitigations–in–llms.pdf (дата звернення: 13.11.2025).

14. Expert.ai. Large Language Models: Opportunity, Risk and Paths Forward. 2023 URL: https://media.expert.ai/expertai/uploads/2023/05/LLMs–Opportunity–Risk–and–Paths–Forward–eBook.pdf (дата звернення: 13.11.2025).

15. Han, Xudong & Yang, Junjie & Wang, Tianyang & Bi, Ziqian & Hao, Jun–Feng & Song, Junhao. Towards Оцінювання–Centric Paradigm: A Survey of Instruction Tuning in Large Language Models. 10.48550/arXiv.2508.17184. 2025. URL: https://www.researchgate.net/publication/394941513_Towards_Оцінювання–Centric_Paradigm_A_Survey_of_Instruction_Tuning_in_Large_Language_Models (дата звернення: 13.11.2025).

16. How to Benchmark LLMs Without Fooling Yourself. Silhouette library. URL: https://silhouette.rocks/how–to–benchmark–llms–without–fooling–yourself (дата звернення: 13.11.2025).

17. Isabel O. Gallegos, Ryan A. Rossi, Joe Barrow, Md Mehrab Tanjim, Sungchul Kim, Franck Dernoncourt, Tong Yu, Ruiyi Zhang, and Nesreen K. Ahmed. 2024. Bias and Fairness in Large Language Models: A Survey. Computational Linguistics, 50(3):1097–1179. URL: https://aclanthology.org/2024.cl–3.8/ (дата звернення: 13.11.2025).

18. ISO/IEC TR 24028:2020. Information technology – Artificial intelligence – Overview of trustworthiness in artificial intelligence. (Edition 1, 2020). URL: https://www.iso.org/standard/77608.html (дата звернення: 13.11.2025).

19. Jan Majkutewicz, Julian Szymański. Aligning large language models with human preferences using historical text edits. URL: https://www.sciencedirect.com/science/article/pii/S0950705125006124 (дата звернення: 13.11.2025).

20. Jason Wei , Maarten Bosma , Vincent Y. Zhao , Kelvin Guu , et al. Finetuned Language Models Are Zero–Shot Learners.8 Feb 2022 . URL: https://arxiv.org/abs/2109.01652 (дата звернення: 13.11.2025).

21. Ji–Lun Peng, Sijia Cheng, Egil Diau. Yung–Yu Shih. A Survey of Useful LLM Evaluation.National Taiwan University, Taipei, Taiwan.03 Jun 2024. URL: https://arxiv.org/html/2406.00936v1 (дата звернення: 13.11.2025).

22. Justin K. Miller. Evaluating LLM Metrics Through Real–World Capabilities.School of Physics University of Sydney Camperdown. 13 May 2025. URL: https://arxiv.org/html/2505.08253v1 (дата звернення: 13.11.2025).

23. Karhu A. Data–driven Product Management: A study into discovering and crafting best practices to lead software products with metrics. 2023 URL: https://lutpub.lut.fi/bitstream/handle/10024/166704/Diplomityo_Aleksi_Karhu.pdf?isAllowed=y&sequence=1 (дата звернення: 13.11.2025).

24. Li, L., Sleem, L., Gentile, N., Nichil, G., & State, R.. Exploring the Impact of Temperature on Large Language Models: Hot or Cold? University of Luxembourg;Foyer S.A.2025, June 8/ URL: https://arxiv.org/html/2506.07295v1 (дата звернення: 13.11.2025).

25. Lianghui Zhu, Xinggang Wang, Xinlong Wang. JudgeLM: Fine–tuned Large Language Models are Scalable Judges. 1 Mar 2025 . URL: https://arxiv.org/abs/2310.17631 (дата звернення: 13.11.2025).

26. LLM evaluation metrics: Full guide to LLM evals and key metrics. Braintrust Team. 28 October 2025 URL: https://www.braintrust.dev/articles/llm–evaluation–metrics–guide (дата звернення: 13.11.2025).

27. Marcelo Pasetti, James William Santos, Nicholas Kluge Corrêa, at al. Technical, legal, and ethical challenges of generative artificial intelligence: an analysis of the governance of training data and copyrights. 31 July 2025. URL: https://link.springer.com/article/10.1007/s44163–025–00379–6 (дата звернення: 13.11.2025).

28. Miklós Sebők, Rebeka Kiss. LLM Parameters Explained: A Practical, Research–Oriented Guide with Examples. May 26, 2025. URL: https://promptrevolution.poltextlab.com/llm–parameters–explained–a–practical–research–oriented–guide–with–examples (дата звернення: 13.11.2025).

29. Mohammadi, Mahmoud & Li, Yipeng & Lo, Jane & Yip, Wendy. Evaluation and Benchmarking of LLM Agents: A Survey. 10.48550/arXiv.2507.21504. July 2025. URL: https://www.researchgate.net/publication/394100858_Evaluation_and_Benchmarking_of_LLM_Agents_A_Survey (дата звернення: 13.11.2025).

30. NIST AI Risk Management Framework (NIST AI RMF 1.0, 2023). URL: https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100–1.pdf (дата звернення: 13.11.2025).

31. NIST. AI Risk Management Framework. 2023. URL: https://www.nist.gov/itl/ai–risk–management–framework (дата звернення: 13.11.2025).

32. OpenAI. Safety best practices. URL: https://platform.openai.com/docs/guides/safety–best–practices (дата звернення: 13.11.2025).

33. Planttext – diagram forming service URL: https://www.planttext.com/ (дата звернення: 13.11.2025).

34. Rethinking the Evaluation of Оцінювання Methods: Insights into Diversity, Generalisation, and Safety. Denis Janiak, Julia Moska, Dawid Motyka, Karolina Seweryn, Paweł Walkowiak, Bartosz Żuk, Arkadiusz Janz URL: https://arxiv.org/abs/2509.12936 (https://arxiv.org/pdf/2509.12936 ) (дата звернення: 13.11.2025).

35. Satyadhar Joshi . Evaluation of Large Language Models: Review of Metrics, Applications, and Methodologies.03 April 2025. URL: https://www.preprints.org/manuscript/202504.0369/v1 (дата звернення: 13.11.2025).

36. Satyadhar Joshi. Advancing the Safety, Performance, and Adaptability of Large Language Models: Review of Fine–Tuning and Guardrails. IRJEMS International Research Journal of Economics and Management Studies. Paper Id: IRJEMS–V4I2P128, Doi: 10.56472/25835238/IRJEMS–V4I2P128. Published by Eternal Scientific Publications. ISSN: 2583 – 5238 / Volume 4 Issue 2 February 2025 / Pg. No: 253–261. .01 February 2025. URL: https://irjems.org/Volume–4–Issue–2/IRJEMS–V4I2P128.pdf (дата звернення: 13.11.2025).

37. Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, Guoyin Wang.Instruction Tuning for Large Language Models: A Survey. Help | Advanced Search. Computer Science / Computation and Language.6 Oct 2025. URL: https://arxiv.org/abs/2308.10792 (дата звернення: 13.11.2025).

38. Shiwen Ni, Guhong Chen, Shuaimin Li, Xuanang Chen, Siyi Li, Bingli Wang, Qiyao Wang, Xingjian Wang, Yifan Zhang, Liyang Fan, Chengming Li, Ruifeng Xu, Le Sun, Min Yang. Survey on Large Language Model Benchmarks.21 Aug 2025. URL: https://arxiv.org/abs/2508.15361 (дата звернення: 13.11.2025).

39. Timnit Gebru, Jamie Morgenstern, Briana Vecchione, et al. Datasheets for Datasets. 1 Dec 2021 URL: https://arxiv.org/abs/1803.09010 (дата звернення: 13.11.2025).

40. Valerio Zanini. The Four Types of Metrics for AI Product Managers. 2025. URL: https://www.5dvision.com/post/metrics–for–ai–pms/ (дата звернення: 13.11.2025).

41. Warudkar S. Skyhigh Security. “Enterprise AI Adoption & Security Risk – Now with 100% more chaos”. May 8, 2025 URL: https://www.skyhighsecurity.com/industry–perspectives/enterprise–ai–adoption–security–risk–now–with–100–more–chaos.html (дата звернення: 13.11.2025).

42. Więckowski J, Sałabun W, Kizielewicz B, et al. Recent advances in multi–criteria decision analysis: A comprehensive review of applications and trends. International Journal of Knowledge–Based and Intelligent Engineering Systems. 2023;27(4):367–393. doi:10.3233/KES–230487, URL: https://journals.sagepub.com/doi/full/10.3233/KES–230487 (дата звернення: 13.11.2025)."

43. Yuntao Bai, Saurav Kadavath, Sandipan Kundu et al. Constitutional AI: Harmlessness from AI Feedback. 15 Dec 2022. URL: https://arxiv.org/abs/2212.08073 (дата звернення: 13.11.2025).

44. Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang,

Yi Chang, Philip S. Yu, Qiang Yang, Xing Xie. A Survey on Evaluation of Large Language Models. 29 Dec 2023. URL: https://arxiv.org/pdf/2307.03109 (дата звернення: 13.11.2025).

45. Zhang, B., & Wang, J. et al. A Survey on Data Selection for LLM Instruction Tuning.23 Jun 2025. URL: https://arxiv.org/html/2402.05123v2 (дата звернення: 13.11.2025).