

Харківський національний університет радіоелектроніки

Факультет Навчально-науковий центр заочної форми навчання

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_____» _____ 20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Абросімову Вадиму Павловичу
(прізвище, ім'я, по батькові)

1. Тема роботи Система виявлення дезінформації та логічних помилок у медіатекстах з використанням пояснюваного штучного інтелекту (XAI)

затверджена наказом університету від 18 травня 2026 р. № 76Стз

2. Термін подання здобувачем роботи до екзаменаційної комісії 7 липня 2026 р.

3. Вихідні дані до роботи науково-технічні публікації з NLP, XAI та QLoRA; набір даних MIPD; локальні артефакти Bielik-4.5B і LoRA-адаптерів; кодові матеріали backend, frontend та MLOps-пайплайна.

4. Перелік питань, що потрібно опрацювати в роботі _____

1) Аналіз предметної галузі та теоретичні основи

2) Методи та проєктування системи

3) Програмна реалізація системи

4) Експериментальні дослідження та оцінка результатів

КАЛЕНДАРНИЙ ПЛАН

[illegible]

Дата видачі завдання 18 травня 20 26 р.

Здобувач _____
(підпис)

Керівник роботи _____
(підпис)

ас. Дмитро МАЛЄЄВ
(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 63 с., 9 рис., 5 табл., 1 дод., 26 джерел.

ДЕЗІНФОРМАЦІЯ, ЛОГІЧНІ ПОМИЛКИ, МАНІПУЛЯЦІЯ, ПОЯСНЮВАНИЙ ШТУЧНИЙ ІНТЕЛЕКТ, BIELIK, CHAIN-OF-THOUGHT, FASTAPI, HUMAN-IN-THE-LOOP, LLM, MLOPS, OLLAMA, QLORA, XAI.

Об'єкт дослідження – процес автоматичного виявлення дезінформації, технік маніпуляції та логічних помилок у текстових медіаматеріалах. Предмет дослідження – методи NLP, тонкого налаштування великих мовних моделей і генерації пояснень природною мовою.

Мета роботи – розроблення локальної інформаційної системи, яка виявляє маніпулятивні техніки у медіатекстах, повертає структурований JSON-результат і формує текстове пояснення логіки рішення засобами пояснюваного штучного інтелекту.

Методи дослідження – аналіз джерел, порівняння існуючих рішень, дистиляція знань Teacher-Student, Supervised Fine-Tuning з QLoRA та експериментальне оцінювання за метриками Parsing Success Rate, Exact-Match Accuracy і Mean Document-Level F1.

У результаті роботи реалізовано прототип вебсистеми з backend FastAPI, frontend React/Vite, локальним сервером Ollama, LoRA-адаптерами до Bielik-4.5B і MLOps-петлею донавчання з участю експерта. Практична новизна полягає у поєднанні локального інференсу, відновлення некоректних JSON-відповідей і швидкого оновлення адаптера.

Результати можуть використовуватись для медіамоніторингу, навчання медіаграмотності та підтримки роботи аналітика.

ABSTRACT

Bachelor's thesis contains: 63 pp., 9 fig., 5 tabl., 1 ann., 26 references.

BIELIK, CHAIN-OF-THOUGHT, DISINFORMATION, EXPLAINABLE ARTIFICIAL INTELLIGENCE, FASTAPI, HUMAN-IN-THE-LOOP, LARGE LANGUAGE MODELS, LLM, LOGICAL FALLACIES, MLOPS, OLLAMA, QLORA, XAI.

The object of research is the process of automatic identification of disinformation, manipulation techniques and logical fallacies in textual media content.

The subject of research is natural language processing methods, large language model fine-tuning and natural language explanation generation for interpreting classifier decisions.

The goal of the thesis is to design and implement a local information system that detects manipulative techniques in media texts, returns a structured JSON result and generates a human-readable explanation using explainable artificial intelligence principles.

The research methods include literature analysis, comparison of existing approaches, Teacher-Student knowledge distillation, supervised fine-tuning with QLoRA and experimental evaluation using Parsing Success Rate, Exact-Match Accuracy and Mean Document-Level F1.

The result is a working prototype that combines a FastAPI backend, a React/Vite frontend, a local Ollama inference server, LoRA adapters for Bielik-4.5B and a Human-in-the-Loop MLOps retraining loop. The practical value of the system is its local deployment model, explainable output and ability to update the model adapter without relying on external commercial APIs.

The system can be applied in media monitoring, information security and educational media literacy scenarios.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	8
Вступ.....	9
1 Аналіз предметної галузі та теоретичні основи.....	11
1.1 Сучасний стан проблеми дезінформації у медіасередовищі	11
1.2 Таксономія технік маніпуляції та логічних помилок	12
1.3 Огляд комерційних та академічних підходів	14
1.4 Теоретичні засади пояснюваного штучного інтелекту	15
1.5 Великі мовні моделі як основа локального аналізу.....	16
1.6 Постановка задачі та критерії успішності	17
1.7 Висновки до розділу 1	18
2 Методи та проєктування системи.....	19
2.1 Загальна концепція рішення	19
2.2 Підготовка даних і набір MIPD	20
2.3 Дистиляція знань і генерація пояснень.....	21
2.4 Тонке налаштування моделі засобами QLoRA	22
2.5 Архітектура інференсу	23
2.6 MLOps-петля з участю експерта.....	24
2.7 Висновки до розділу 2	26
3 Програмна реалізація системи	27
3.1 Структура репозиторію та середовища виконання	27
3.2 Backend API та обробка запитів.....	28
3.3 Нормалізація відповідей LLM	29
3.4 Реалізація тренування та оркестрації.....	30
3.5 Модель даних і збереження стану	31
3.6 Frontend-інтерфейс користувача та експерта	32
3.7 Інсталяція, запуск і відновлення стану	34
3.8 Тестування та контроль якості.....	34
4 Експериментальні дослідження та оцінка результатів.....	38

4.1 Методика експериментального оцінювання	38
4.2 Порівняння варіантів моделі.....	39
4.3 Аналіз матриць помилок	41
4.4 Аналіз продуктивності та ресурсних обмежень	43
4.5 Сценарії використання системи.....	46
4.6 Порівняння з аналогами та практична цінність	48
4.7 Обмеження та напрями подальшого розвитку.....	51
4.8 Додатковий аналіз впровадження	55
Висновки	58
Перелік джерел посилання	60
Додаток А Відомість кваліфікаційної роботи	63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface – програмний інтерфейс прикладного рівня;

CoT – Chain-of-Thought – ланцюжок міркувань;

EM – Exact Match – повний збіг множини прогнозованих і еталонних тегів;

F1 – гармонічне середнє точності та повноти;

GGUF – формат збереження моделей для llama.cpp та Ollama;

JSON – JavaScript Object Notation – формат структурованого обміну даними;

LLM – Large Language Model – велика мовна модель;

LoRA – Low-Rank Adaptation – низькорангова адаптація параметрів;

MLOps – практики життєвого циклу моделей машинного навчання;

NLE – Natural Language Explanation – пояснення природною мовою;

NLP – Natural Language Processing – обробка природної мови;

PSR – Parsing Success Rate – частка успішно розібраних структурованих відповідей;

QLoRA – Quantized Low-Rank Adaptation – LoRA з квантуванням базової моделі;

SFT – Supervised Fine-Tuning – контрольоване тонке налаштування;

WSL2 – Windows Subsystem for Linux другого покоління;

XAI – Explainable Artificial Intelligence – пояснюваний штучний інтелект.

ВСТУП

Актуальність теми зумовлена тим, що онлайн-медіа і соціальні мережі стали основним середовищем поширення суспільно значущої інформації. Разом із корисним контентом у цьому середовищі швидко поширюються дезінформаційні повідомлення, маніпулятивні заголовки, емоційно забарвлені твердження та логічні помилки. Їхній вплив посилюється алгоритмічними стрічками, які винагороджують залучення користувача, а не достовірність повідомлення.

Наявні підходи до автоматичної детекції дезінформації мають обмеження. Закриті комерційні LLM забезпечують сильне мовне висновування, але створюють залежність від зовнішнього API та ризики приватності. Класичні класифікатори на основі BERT працюють швидко, проте часто повертають лише мітку або ймовірність без пояснення. Фактчекінгові портали забезпечують високу якість, але не масштабуються до потоку повідомлень у реальному часі.

Мета роботи полягає у проєктуванні та програмній реалізації локальної інформаційної системи для виявлення дезінформації та логічних помилок у медіатекстах з використанням пояснюваного штучного інтелекту. Система має не тільки класифікувати техніки маніпуляції, а й генерувати зрозуміле пояснення рішення у природній мові.

Для досягнення мети поставлено такі задачі: виконати аналіз предметної галузі та існуючих рішень; підготувати набір даних із поясненнями; виконати тонке налаштування локальної LLM; розробити архітектуру інференсу і MLOps-петлі; реалізувати frontend, backend і модуль нормалізації відповідей; провести експериментальну оцінку якості моделей; визначити обмеження та напрями розвитку.

Об'єктом дослідження є процес автоматичної ідентифікації технік маніпуляції у текстових медіаматеріалах. Предметом дослідження є методи NLP, великі мовні моделі, QLoRA-донавчання, Natural Language

Explanations і архітектурні рішення для локального MLOps-життєвого циклу.

Практичне значення роботи полягає у створенні працездатного прототипу, який може використовуватися для медіамоніторингу, навчання медіаграмотності та підтримки роботи аналітика. Рішення працює локально, використовує модель Bielik-4.5B з LoRA-адаптером, backend FastAPI, frontend React/Vite і сервер інференсу Ollama.

Оформлення пояснювальної записки виконано з урахуванням вимог ДСТУ 3008:2015, ДСТУ 8302:2015 та методичних вказівок ХНУРЕ [1], [2], [3]. Теоретична й інженерна база роботи спирається на Transformer-архітектуру [4], дослідження пояснюваного III [5], [6], Natural Language Explanations [7], [8], Chain-of-Thought [9], дистиляцію знань [10], LoRA та QLoRA [11], [12], RoPE-масштабування [13], модель Bielik [14], Qwen [15], корпус MIPD [16], бібліотеки Transformers і Unsloth [17], [18], Ollama та llama.cpp [19], [20], FastAPI [21], React [22], SQLite і SQLAlchemy [23], [24], scikit-learn [25] і WSL2 [26].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ТЕОРЕТИЧНІ ОСНОВИ

1.1 Сучасний стан проблеми дезінформації у медіасередовищі

Цифрові медіа змінили швидкість поширення інформації настільки, що класичні процедури редакційної перевірки більше не покривають реальний обсяг контенту. Дезінформаційний текст може отримати широку аудиторію до того, як людина-експерт встигне оцінити джерела, логіку аргументації та контекст публікації. Саме тому автоматизована підтримка аналізу медіатекстів стала прикладною задачею інформаційної безпеки, а не лише академічною справою з класифікації текстів.

Особливість досліджуваної задачі полягає у тому, що дезінформація рідко зводиться до формально неправдивого факту. У багатьох випадках текст використовує емоційні маркери, вибіркового добір фактів, хибні причинно-наслідкові зв'язки або маніпулятивні запитання. Такі прийоми не завжди можуть бути виявлені простим порівнянням із базою перевірених тверджень, тому система має аналізувати логічну структуру повідомлення.

Розроблена робота розглядає медіатекст як об'єкт багатоміткової класифікації, де один документ може містити кілька технік маніпуляції або не містити жодної. Такий підхід ближчий до реальних умов, ніж бінарна класифікація «правда/неправда», оскільки він дозволяє показати користувачу характер проблеми, а не лише кінцеву оцінку.

Практична актуальність задачі підсилюється тим, що користувачеві недостатньо отримати лише мітку класу. Якщо система позначає фрагмент як маніпулятивний, вона повинна пояснити, за якими ознаками це зроблено. Без пояснення навіть формально правильна відповідь сприймається як рішення «чорної скриньки», а отже не підвищує довіру до інструмента і не виконує освітню функцію.

У межах роботи проблему сформульовано як інженерну: потрібно створити не ізольований експериментальний ноутбук, а систему, придатну

для локального запуску, інтеграції з вебінтерфейсом і подальшого донавчання. Такий фокус визначив вибір технологій, структуру експериментів і спосіб оцінювання результатів.

1.2 Таксономія технік маніпуляції та логічних помилок

Для формалізації задачі використано одинадцять категорій маніпуляцій, що відповідають структурі набору MIPD та реалізовані у програмному модулі нормалізації тегів. До них належать `REFERENCE_ERROR`, `WHATABOUTISM`, `STRAWMAN`, `EMOTIONAL_CONTENT`, `CHERRY_PICKING`, `FALSE_CAUSE`, `MISLEADING_CLICKBAIT`, `ANECDOTE`, `LEADING_QUESTIONS`, `EXAGGERATION` і `QUOTE_MINING`. Кожна категорія описує не тему тексту, а спосіб аргументаційного впливу.

`REFERENCE_ERROR` характеризує ситуації, коли джерело не підтверджує тезу, є ненадійним або подається в оманливий спосіб. `WHATABOUTISM` відводить увагу від основного питання через згадування інших порушень, а `STRAWMAN` спрощує або перекручує позицію опонента. Ці категорії є особливо важливими для політичних і суспільних текстів, де маніпуляція часто прихована в логічному русі аргументу.

`EMOTIONAL_CONTENT`, `EXAGGERATION` і `MISLEADING_CLICKBAIT` пов'язані з емоційним впливом на читача. Вони не обов'язково містять фактичну помилку, але змінюють сприйняття повідомлення через страх, обурення, перебільшення або сенсаційне оформлення. Для системи важливо розрізняти ці техніки, оскільки вони по-різному проявляються в лексичних і композиційних ознаках.

`CHERRY_PICKING`, `FALSE_CAUSE`, `ANECDOTE`, `LEADING_QUESTIONS` і `QUOTE_MINING` описують порушення коректності доказової бази. Текст може вибирати лише зручні факти, будувати причинність з кореляції, узагальнювати одиничний приклад,

ставити запитання із прихованою передумовою або виривати цитату з контексту. Саме ці випадки показують, чому задача потребує аналізу змісту, а не лише поверхневих ключових слів.

У системі назви тегів залишено англійською мовою, оскільки вони виконують роль машинного контракту між моделлю, backend-модулем і frontend-інтерфейсом. Для користувача теги перекладаються у локалізованому інтерфейсі, але внутрішнє представлення залишається стабільним. Це зменшує ризик помилок під час оцінювання й дозволяє зіставляти результати з вихідним набором даних.

У таблиці 1.1 наведені техніки маніпуляції, що розпізнаються системою.

Таблиця 1.1 – Техніки маніпуляції, що розпізнаються системою

Тег	Українська назва	Стислий зміст
REFERENCE_ERROR	Помилка посилання	джерело не підтверджує тезу або є ненадійним
WHATABOUTISM	Вотабаутизм	відведення уваги через інше звинувачення
STRAWMAN	Солом'яне опудало	атака на перекручену позицію опонента
EMOTIONAL_CONTENT	Емоційна мова	емоційний тиск замість раціонального доказу
CHERRY_PICKING	Вибірковий добір	подання лише зручних фактів
FALSE_CAUSE	Хибна причина	помилковий причинно-наслідковий зв'язок
MISLEADING_CLICKBAIT	Маніпулятивний заголовок	сенсаційний заголовок, що спотворює зміст
ANECDOTE	Анекдотичний доказ	узагальнення з одиничного прикладу
LEADING_QUESTIONS	Навідні запитання	питання з прихованою передумовою
EXAGGERATION	Перебільшення	гіперболізація фактів
QUOTE_MINING	Цитата поза контекстом	використання фрагмента для зміни змісту цитати

1.3 Огляд комерційних та академічних підходів

Першу групу аналогів становлять універсальні комерційні LLM, зокрема GPT-подібні та Gemini-подібні сервіси. Їхня перевага полягає у високій якості мовного висновування та здатності працювати з довгими текстами. Водночас використання закритого API створює залежність від зовнішнього постачальника, вартісні обмеження, ризики приватності і проблему відмови від аналізу політично чутливого контенту через політики безпеки.

Другу групу утворюють класичні класифікатори на основі BERT або RoBERTa. Вони швидкі, відносно дешеві в інференсі та добре вивчені у задачах класифікації тексту. Однак їхній типовий результат – це ймовірність або набір міток без розгорнутого пояснення. Для користувача, який хоче зрозуміти, що саме в тексті є проблемним, така відповідь має обмежену практичну цінність.

Третю групу становлять фактчекінгові портали та експертні редакційні процеси. Вони забезпечують високу якість перевірки, але не масштабуються до потоку повідомлень у соціальних мережах. Людина-експерт залишається необхідною для складних випадків, проте автоматизована система може виконувати попередню фільтрацію, зменшуючи навантаження на аналітика.

Запропоноване рішення займає проміжну позицію: воно використовує генеративну модель, але працює локально; воно класифікує техніки, але також генерує пояснення; воно передбачає участь експерта, але не вимагає від нього ручного перенавчання моделі на рівні коду. Саме ця комбінація відрізняє систему від одноразового класифікатора або зовнішнього чат-бота.

Аналіз аналогів показав, що ключовими критеріями для роботи є автономність, пояснюваність, здатність до оновлення, структурованість результату та прийнятні вимоги до апаратних ресурсів. Ці критерії стали

основою подальшого проєктування архітектури й експериментального порівняння варіантів моделі.

1.4 Теоретичні засади пояснюваного штучного інтелекту

Пояснюваний штучний інтелект розглядає не лише точність моделі, а й можливість людини зрозуміти причину рішення. Для задачі детекції дезінформації це критично, оскільки неправильне або необґрунтоване звинувачення тексту в маніпуляції може мати репутаційні наслідки. Тому результат системи повинен містити не тільки список технік, а й природномовне пояснення.

У роботі використано підхід Natural Language Explanations. Його сутність полягає у формуванні текстового обґрунтування, яке описує, які фрагменти або логічні прийоми вплинули на класифікацію. Такий формат є зручним для неекспертного користувача, на відміну від теплових карт або ваг уваги, що потребують спеціальної інтерпретації.

Механізм Chain-of-Thought використовується як концептуальна основа для навчання моделі будувати проміжне міркування перед фінальним набором тегів. У практичній реалізації зовнішній результат подається як поле reasoning у JSON-об'єкті. Це дозволяє поєднати читабельне пояснення з машинно оброблюваною структурою відповіді.

Водночас пояснюваність створює додаткову складність для моделі. Вона повинна одночасно виконати класифікацію, сформувати логічний опис і зберегти валідний формат JSON. Експериментальні результати підтвердили, що додавання пояснень може знизити класифікаційну метрику порівняно з чистим прототипним адаптером, що в роботі розглядається як «податок на пояснюваність».

Отже, теоретична основа роботи поєднує багатоміткову класифікацію, генерацію пояснень і контроль структурованого виводу.

Саме ця комбінація визначає вимоги до даних, моделі, backend-нормалізації та метрик оцінювання.

1.5 Великі мовні моделі як основа локального аналізу

Великі мовні моделі стали придатними для задач, де потрібне не лише розпізнавання шаблонів, а й мовне узагальнення. Для аналізу маніпуляцій це важливо, оскільки модель повинна зіставити зміст повідомлення з логічною категорією та пояснити результат людською мовою. Водночас розмір моделі безпосередньо впливає на вимоги до пам'яті й швидкість інференсу.

У роботі обрано сімейство Bielik як базову локальну модель для польськомовного медіаконтенту. Вибір зумовлений тим, що модель орієнтована на польську мову, має розмір 4,5 млрд параметрів і може бути запущена на споживчому обладнанні після квантування. Це відповідає вимозі автономності та зменшує залежність від зовнішніх API.

Для генерації навчальних пояснень використано модель Qwen-2.5-7B-Instruct як сильнішу модель-вчитель. Такий поділ ролей відповідає практиці дистиляції знань: більша модель генерує складніші пояснення, а менша модель-учень навчається відтворювати прийнятну поведінку в локальному режимі. У такий спосіб частина можливостей більшого LLM переноситься до дешевшого в експлуатації рішення.

Ключовим обмеженням локальних моделей є нестабільність структурованого виводу. Навіть після донавчання модель може помилятися у назвах ключів JSON або генерувати варіанти тегів з орфографічними відхиленнями. Тому архітектура системи не покладається на ідеальність LLM, а включає окремий модуль відновлення та нормалізації відповіді.

Таким чином, LLM у роботі використовується не як самодостатній сервіс, а як компонент ширшої інженерної системи. Його результат

контролюється промптом, адаптером, сервером інференсу, backend-перевіркою і frontend-представленням.

1.6 Постановка задачі та критерії успішності

Задача кваліфікаційної роботи полягає у розробленні програмної системи, що приймає медіатекст, передає його локальній мовній моделі, отримує структуровану відповідь і відображає користувачу знайдені техніки маніпуляції разом із поясненням. Система має бути придатною до локального запуску на Windows з використанням WSL2 для тренувальних процедур.

Функціональні вимоги охоплюють аналіз довільного тексту, відображення тегів у зрозумілому інтерфейсі, роботу експертного режиму, завантаження нового навчального набору у форматі JSONL, запуск донавчання, отримання прогресу через WebSocket, автоматичну оцінку кандидата та можливість розгортання нового адаптера.

Нефункціональні вимоги включають автономність інференсу, відсутність обов'язкових зовнішніх API, відновлюваність результату, стабільний JSON-контракт, прийнятний час відповіді, безпечне повторне виконання скриптів налаштування і можливість діагностики помилок. Для інженерної системи ці вимоги не менш важливі, ніж сама якість моделі.

Критеріями успішності обрано Parsing Success Rate для оцінки структурної придатності відповіді, Exact-Match Accuracy для жорсткого порівняння множини тегів і Mean Document-Level F1 для якості багатоміткової класифікації на документах з непорожніми еталонними мітками. Такий набір метрик дозволяє відокремити технічну стабільність формату від змістової якості класифікації.

Додатковим критерієм є практична інтегрованість: модель повинна бути не лише навчена в ноутбучі, а й доступна через вебінтерфейс, пов'язана з backend-сервісом і включена в цикл оновлення. Це переводить результат із

рівня дослідного експерименту до рівня працездатного інженерного прототипу.

1.7 Висновки до розділу 1

У першому розділі показано, що задача виявлення дезінформації потребує не лише класифікації, а й інтерпретованого пояснення. Сучасні комерційні LLM забезпечують сильне мовне висновування, але створюють залежність від зовнішніх сервісів. Класичні BERT-подібні моделі швидкі, однак не дають природномовного обґрунтування.

Для подальшої роботи обґрунтовано використання локальної LLM з LoRA-адаптером, багатоміткової таксономії маніпуляцій і XAI-підходу на основі Natural Language Explanations. Сформульовані критерії успішності визначили, що система повинна оцінюватися одночасно за якістю класифікації, стабільністю JSON-виводу та придатністю до повторного донавчання.

2 МЕТОДИ ТА ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Загальна концепція рішення

Проектowana система складається з трьох взаємопов'язаних контурів: контуру інференсу, контуру підготовки даних і контуру донавчання. Контур інференсу відповідає за взаємодію користувача з моделлю, контур підготовки даних формує приклади з поясненнями, а контур донавчання дозволяє адаптувати модель до нових зразків маніпуляцій.

Основний сценарій роботи починається з введення медіатексту у frontend-інтерфейсі. Далі текст надходить до FastAPI-сервісу, який формує запит до Ollama. Сервер Ollama запускає модель Bielik-4.5B з активним LoRA-адаптером і повертає JSON-подібну відповідь. Backend нормалізує її та віддає frontend-частині очищений результат.

Експертний сценарій передбачає завантаження JSONL-файлу з новими прикладами. Backend запускає процес тренування у WSL2, отримує прогрес через HTTP callback, після завершення проводить benchmark і показує оператору порівняння нових метрик із базовими. Рішення про розгортання залишається за людиною, що відповідає принципу Human-in-the-Loop.

Система не намагається автоматично замінити експерта у всіх рішеннях. Її роль – пришвидшити первинний аналіз, зробити аргументацію моделі видимою та надати інженерний механізм для швидкого оновлення поведінки моделі. Такий підхід знижує ризики неконтрольованого автоматичного навчання.

Загальна концепція орієнтована на відтворюваність: залежності описані у requirements.txt, requirements-wsl.txt і package.json, а запуск автоматизовано через setup.cmd та run_app.cmd. Це важливо для кваліфікаційної роботи, оскільки результатом має бути не лише опис, а й артефакт, який можна перевірити.

2.2 Підготовка даних і набір MIPD

Основним джерелом навчальних і тестових прикладів став набір MIPD, що містить польськомовні тексти з мітками маніпуляцій. Його структура відповідає поставленій задачі, оскільки один документ може містити кілька технік або не містити жодної. Такий формат дозволяє будувати багатоміткову класифікацію без спрощення реального медіаконтексту.

Початкова проблема полягала у відсутності для кожного прикладу розгорнутих пояснень природною мовою. Наявні мітки достатні для тренування класифікатора, але недостатні для навчання моделі генерувати поле reasoning. Тому було спроектовано окремий пайплайн синтетичного збагачення даних поясненнями.

Підготовка даних виконувалася у notebook-середовищі Google Colab. Це дало змогу використати GPU-ресурси без розгортання локального тренувального кластера. У репозиторії збережено ноутбуки `MIPD_processing.ipynb`, `synthetic_data_gen.ipynb`, `bielik_qwen_4_5b_sft.ipynb` і `Benchmark.ipynb`, що відображають етапи перетворення даних, генерації пояснень, навчання і оцінювання.

Для практичного використання в системі підготовлені файли `mipd_train_cot_clean.jsonl`, `mipd_val.jsonl` і `mipd_test.jsonl`. Навчальний файл містить пари input-output, де output є JSON-структурою з reasoning та discovered_techniques. Такий формат напрямку відповідає контракту, який очікує backend і frontend.

Важливо, що дані не розглядаються як статичний завершений ресурс. Через експертний режим система може приймати нові JSONL-зразки, створюючи передумови для ітеративного розширення корпусу. Це відповідає природі дезінформації, де нові наративи й прийоми з'являються швидше, ніж оновлюються класичні датасети.

2.3 Дистиляція знань і генерація пояснень

Для створення пояснень застосовано підхід Teacher-Student. Модель Qwen-2.5-7B-Instruct виконувала роль моделі-вчителя, генеруючи обґрунтування для прикладів MIPD з урахуванням заданих еталонних міток. Модель Bielik-4.5B у подальшому навчалася відтворювати цю поведінку у локальному режимі.

Важливою деталлю стала умовна генерація пояснень за міткою. Модель-вчитель не мала самостійно змінювати клас прикладу, а повинна була пояснити вже задану експертну мітку. Це зменшувало ризик конфлікту між корпусом і синтетичним поясненням, хоча повністю не усувало можливості помилок або галюцинацій у reasoning.

Пайплайн генерації містив перевірку формату, повторні спроби та логіку відновлення після переривання сесії. Такі інженерні механізми були необхідні, оскільки генерація тисяч пояснень є тривалою процедурою, а нестабільність формату JSON могла зіпсувати значну частину корпусу.

Окремим рішенням було програмне складання структури JSON. Модель відповідала за зміст пояснення, тоді як ключі та список технік контролювалися кодом. Це підвищувало якість навчального набору, хоча під час інференсу модель усе одно навчалася генерувати повну структуру самостійно.

Перевагою дистиляції є можливість отримати пояснювану поведінку без ручного написання тисяч rationale. Недоліком є ризик наслідування упереджень і помилок моделі-вчителя. Тому у висновках і напрямках розвитку окремо виділено потребу формальної перевірки якості пояснень.

Процес генерації у `synthetic_data_gen.ipynb` мав кілька інженерних запобіжників: використання Qwen2.5-7B-Instruct як моделі-вчителя, максимум `MAX_RETRIES = 3` для повторної генерації, програмне складання JSON-структури, журналювання відхилених зразків у `denied_samples.txt`, фіксоване нейтральне пояснення для прикладів без

технік і функцію `validate_reasoning()`, яка перевіряла згадування очікуваних назв технік у `reasoning`. Після очищення допоміжні колонки не переносилися до фінального навчального файлу.

Детермінований аудит `model/dataset/mipd_train_cot_clean.jsonl` зафіксував 9903 записи. Усі 9903 записи мають валідний JSON у полі `output` і непорожнє `reasoning`. Після нормалізації відомого скорочення `MISLEADING_CLICKBAI` до `MISLEADING_CLICKBAIT` не залишилося записів з невідомими тегами; 52 записи потребували саме такої нормалізації. Корпус містить 8136 нейтральних і 1767 маніпулятивних прикладів. Перевірка відповідності `reasoning` очікуваним тегам дала 9851/9903 записів, тобто 99,47%, якщо вимагати згадування вже нормалізованої повної назви тегу. Скрипт відтворення аудиту наведено у `documentation/ua/evidence/audit_mipd_train_cot_clean.py`.

Цей результат показує, що пайплайн генерації (рисунок 2.1) забезпечував структурні та `label-consistency` обмеження, але не доводить експертну семантичну правильність кожного пояснення. Тому якість NLE у роботі розглядається як контрольована на рівні формату й узгодженості з мітками, але не як повністю експертно валідована або еквівалентна поясненням, написаним людиною-фахівцем.



Рисунок 2.1 – Пайплайн генерації пояснень для навчального набору

2.4 Тонке налаштування моделі засобами QLoRA

Повне донавчання моделі з 4,5 млрд параметрів потребувало б значних обчислювальних ресурсів. Для зменшення вимог використано QLoRA, яка поєднує 4-бітове представлення базових ваг із навчанням

невеликих низькорангових адаптерів. Це дозволяє змінювати поведінку моделі без оновлення всіх параметрів.

У реалізації застосовано бібліотеку Unsloth, що оптимізує пам'ять і швидкість тренування. Код `trainer.py` створює LoRA-адаптер з рангом $r=16$, $\alpha=16$ і цільовими модулями `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj` та `down_proj`. Такий вибір охоплює ключові проєкції трансформера і відповідає практиці адаптації LLM.

Для зменшення споживання пам'яті використано `gradient checkpointing`, `batch size 1`, накопичення градієнтів і 8-бітовий оптимізатор `paged_adamw_8bit`. У локальному MLOps-процесі довжину послідовності обмежено для стабільності, тоді як експериментальні ноутбуки досліджували довші контексти.

Навчання організовано як Supervised Fine-Tuning на прикладах діалогу користувача й асистента. Текст статті подається як повідомлення користувача, а очікувана JSON-відповідь – як повідомлення асистента. Це узгоджується з ChatML-шаблоном, що надалі використовується у Modelfile для Ollama.

Результатом тренування є каталог `adapter` із файлами адаптера й токенизатора. У подальшому адаптер може бути перетворений у GGUF і підключений до Ollama. Таким чином, метод QLoRA забезпечує не лише експериментальне навчання, а й шлях до практичного розгортання.

2.5 Архітектура інференсу

Інференс організовано як послідовність взаємодій між React frontend, FastAPI backend і Ollama (рисунок 2.2). Frontend не звертається до моделі напряму, оскільки саме backend відповідає за нормалізацію відповіді, обробку помилок і збереження стабільного API-контракту для клієнтської частини.

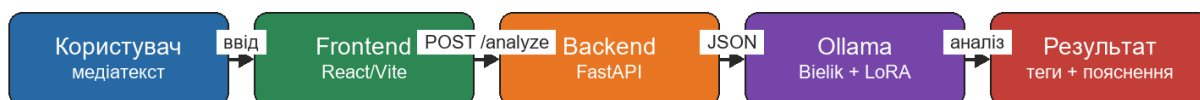


Рисунок 2.2 – Архітектура інференсу системи

Backend endpoint `/analyze` приймає JSON із полем `text`, формує запит до Ollama API `/api/chat`, встановлює `stream=false` та `format=json`. Параметр `keep_alive=0` використовується для контролю життєвого циклу моделі у локальному середовищі. Після отримання відповіді backend виділяє поле `message.content` і передає його до `normalize_llm_response`.

Моделі у Ollama визначена у файлі `Modelfile`. Він посилається на базовий GGUF-файл `Bielik-4.5B`, підключає LoRA-адаптер, задає `temperature=0.1`, stop-токен `<|im_end|>`, контекст `num_ctx=16384` і ChatML-шаблон. Низька температура обрана для підвищення детермінізму структурованого JSON-виводу.

Сильна сторона такої архітектури полягає у відокремленні ролей. Ollama відповідає за ефективний локальний інференс, FastAPI – за бізнес-логіку й стабільність відповіді, React – за взаємодію з користувачем. Це спрощує тестування і дозволяє замінити один компонент без повного перепроектування системи.

У разі помилки зв'язку з Ollama backend повертає HTTP 500 з описом проблеми. Така поведінка є прийнятною для прототипу і допомагає діагностувати налаштування локального середовища. Для промислової версії доцільним є додавання черг, таймаут-політик і централізованого логування.

2.6 MLOps-петля з участю експерта

MLOps-петля (рисунок 2.3) реалізує ідею, що модель не повинна залишатися статичною після початкового навчання. Нові інформаційні

кампанії можуть використовувати інші наративи, тому система має підтримувати донавчання на свіжих прикладах. Водночас рішення про розгортання нового адаптера не автоматизується повністю, а передається експерту.

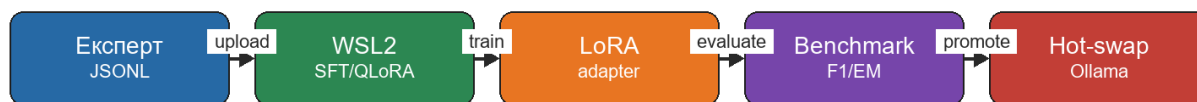


Рисунок 2.3 – MLOps-петля донавчання з участю експерта

Backend endpoint `/training/upload` приймає JSONL-файл, зберігає його в `uploads` і запускає `start_manual_training`. Оркестратор створює запис `TrainingRun` у `SQLite`, формує WSL-команду і запускає `trainer.py` в окремому процесі. Прогрес тренування передається назад до backend через endpoint `/training/progress`.

Після завершення тренування endpoint `/training/complete` переводить систему в стан `evaluating` і запускає `benchmark`. Результати порівнюються з поточними базовими метриками, що читаються з `current_baseline_report.txt`. Frontend отримує `status`, `training_progress`, `evaluation_progress` і нові метрики через `WebSocket`.

Якщо експерт погоджується з якістю кандидата, endpoint `/training/promote` запускає конверсію адаптера в GGUF і оновлює ADAPTER-рядок у `Modelfile`. Далі виконується `ollama create` для hot-swap моделі. Успішне розгортання оновлює стан системи на `deployment_success`.

Такий цикл є компромісом між автоматизацією та контролем. Система автоматизує технічну рутину – запуск, оцінку, конверсію, оновлення, – але людина зберігає право прийняти або відхилити модель. Для предметної області дезінформації це важливо, бо сліпе автоматичне розгортання може підсилити помилки або упередження.

2.7 Висновки до розділу 2

У другому розділі обґрунтовано методику створення системи: використання MIPD як базового корпусу, генерацію пояснень моделювачем, тонке налаштування Bielik-4.5B через QLoRA і побудову окремих контурів інференсу та донавчання.

Запропонована архітектура враховує обмеження локального середовища, потребу у структурованому JSON-виводі та необхідність контролю з боку експерта. Вона створює підґрунтя для програмної реалізації, описаної у наступному розділі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Структура репозиторію та середовища виконання

Репозиторій має модульну структуру: backend містить FastAPI-застосунок, модулі тренування, базу SQLite та тести; frontend містить React/Vite-інтерфейс; model містить локальні артефакти моделі, набори даних і звіти benchmark; colab-notebooks зберігає дослідні ноутбуки. Такий поділ відображає розділення відповідальностей між сервісами.

Backend-залежності описані у requirements.txt і включають fastapi, uvicorn, sqlalchemy, pydantic, httpx, pandas, scikit-learn, python-multipart і websockets. Окремий файл requirements-wsl.txt призначений для тренування у WSL2 і містить unsloth, bitsandbytes, accelerate, torch, trl, datasets та gguf.

Frontend-застосунок використовує React 19, Vite 7 і ESLint. Пакет package.json містить скрипти dev, build, lint і preview. Наявність ESLint важлива для підтримки якості клієнтського коду, а Vite забезпечує швидкий локальний запуск інтерфейсу.

Кореневі скрипти setup.cmd, run_app.cmd і reset_state.cmd автоматизують основні сценарії. setup.cmd перевіряє наявність Python, npm, Ollama, ініціалізує submodule llama.cpp, налаштовує Python/Node-залежності та WSL2-середовище. run_app.cmd запускає Ollama, backend і frontend у правильній послідовності.

Середовище model не версіонується повністю через великий розмір файлів. README.md описує, що каталог містить базову модель, адаптери, датасети та Modelfile. Це рішення відповідає практиці зберігання великих ML-артефактів поза основним Git-репозиторієм.

3.2 Backend API та обробка запитів

Backend реалізовано у файлі `app/main.py`. Об'єкт FastAPI має назву `Disinformation Detector Backend` і налаштований з CORS-політикою, що дозволяє frontend-застосунку надсилати запити з локального порту Vite. Основним endpoint для користувача є `POST /analyze` (таблиця 3.1).

Таблиця 3.1 – Основні endpoint-и backend-сервісу

Endpoint	Метод	Призначення
<code>/analyze</code>	POST	аналіз медіатексту через локальну LLM
<code>/training/upload</code>	POST	завантаження JSONL і запуск тренування
<code>/training/status</code>	GET	отримання поточного стану MLOps-пайплайна
<code>/ws/training/status</code>	WebSocket	передача прогресу тренування й оцінювання у реальному часі
<code>/training/promote</code>	POST	розгортання нового адаптера після схвалення експертом
<code>/training/progress</code>	POST	callback від WSL-процесу тренування або оцінювання

Клас `AnalysisRequest` описує вхідну модель з одним полем `text`. Після отримання запиту backend формує `payload` для Ollama, у якому зазначає модель `bielik-lora-mipd:latest`, повідомлення користувача, режим без потокової передачі та формат `json`. Це дозволяє зберегти простий контракт між frontend і backend.

Для HTTP-взаємодії використано `httpx.AsyncClient` з таймаутом 120 секунд. Такий таймаут враховує, що локальна LLM може відповідати повільніше за звичайні REST-сервіси, особливо на великих текстах. У разі винятку backend повертає `HTTPException` з кодом 500 і описом помилки Ollama.

Крім `/analyze`, backend містить endpoint-и тренувального контуру: `/training/upload`, `/training/status`, `/training/promote`, `/training/progress` і

/training/complete. WebSocket /ws/training/status забезпечує push-оновлення для frontend-панелі експерта, що усуває потребу в постійному polling.

Загалом backend виконує роль оркестратора між користувачем, моделлю, файловою системою, SQLite і WSL2. Така роль є центральною для системи, бо саме тут поєднуються інференс, донавчання, оцінювання та розгортання.

3.3 Нормалізація відповідей LLM

Модуль `app/llm_processor.py` реалізує `normalize_llm_response`. Його призначення – перетворити потенційно нестабільний текст LLM у надійний словник з ключами `reasoning` і `discovered_techniques`. Це критичний компонент, оскільки навіть добре донавчена модель може порушити JSON-схему.

Перший етап нормалізації намагається виконати `json.loads`. Якщо відповідь є коректним JSON-об'єктом, модуль переходить до перевірки ключів і тегів. Якщо parsing не вдавсь, застосовується `regex recovery`: з тексту вилучається поле `reasoning` і список тегів у квадратних дужках.

Другий етап виконує `fuzzy key mapping`. Якщо ключ `reasoning` відсутній, код шукає ключі, що починаються з `reason`. Якщо `discovered_techniques` відсутній, модуль шукає ключі з `technique` або `discovered`. Це покриває типові помилки LLM, наприклад `reason` замість `reasoning`.

Третій етап очищає теги. Валідні теги порівнюються з множиною `VALID_TAGS`, а часті помилки нормалізуються евристично: `EMOTIO` перетворюється на `EMOTIONAL_CONTENT`, `CHERRY` – на `CHERRY_PICKING`, `CLICKBAIT` – на `MISLEADING_CLICKBAIT`, `QUOTE` – на `QUOTE_MINING`. Це підвищує стійкість системи до дрібних орфографічних відхилень.

Нормалізація не повинна приховувати всі помилки моделі. Якщо тег невідомий і не підпадає під евристику, він залишається у результаті. Frontend має локалізований fallback для нерозпізнаної техніки, що дозволяє показати користувачу потенційну галюцинацію замість тихого видалення даних.

3.4 Реалізація тренування та оркестрації

Клас `MLOpsOrchestrator` у `app/training/orchestrator.py` управляє станами `idle`, `training`, `evaluating`, `ready_to_promote`, `deploying`, `deployment_success` і `deployment_error`. Він зберігає прогрес тренування та оцінювання, базові й нові метрики, шлях до останнього адаптера і callback-и для WebSocket-сповіщень.

Процедура `start_manual_training` перевіряє, чи система перебуває у стані, з якого можна почати новий цикл. Далі вона скидає стан кандидата, створює запис `TrainingRun` у базі, конвертує Windows-шляхи у WSL-шляхи та формує команду запуску `trainer.py`. Це дозволяє backend на Windows керувати Linux-середовищем навчання.

Модуль `trainer.py` встановлює офлайн-режими `HF_HUB_OFFLINE` і `TRANSFORMERS_OFFLINE`, завантажує локальну базову модель, ініціалізує LoRA-адаптери, форматує приклади через `chat template` і запускає `SFTTrainer`. Прогрес передається через `ProgressCallback` до backend endpoint `/training/progress`.

Після завершення тренування оркестратор запускає `benchmark.py`. У поточній реалізації інтерактивний benchmark може працювати на скороченій вибірці для швидкого зворотного зв'язку, тоді як основні експериментальні звіти у `model/benchmark-reports` сформовані для повного тестового набору $N=1521$. Це розділяє швидку інженерну перевірку і повну дослідну оцінку.

Конверсія адаптера у GGUF виконується через `converter.py` і `vendored llama.cpp`. Перед конверсією з конфігурації базової моделі може вилучатися `quantization_config`, оскільки `convert_lora_to_gguf.py` не завжди коректно працює з `bitsandbytes`-метаданими. Це приклад практичної інженерної адаптації сторонніх інструментів.

3.5 Модель даних і збереження стану

Для збереження метаданих тренувальних запусків використано SQLite. Файл бази `disinfo_system.db` розташований у каталозі `backend`, а SQLAlchemy-модель `TrainingRun` містить `id`, `start_time`, `end_time`, `fl_score_before`, `fl_score_after`, `status` і `adapter_path`. Така структура достатня для прототипу та не потребує окремого серверного СКБД.

Дані тренувальних і модельних артефактів зберігаються у файловій системі. Це природно для ML-проєкту, де адаптери, GGUF-файли та звіти мають значний розмір і не повинні зберігатися у реляційній таблиці. Отже, система використовує двошарову модель: SQLite для метаданих і файлову систему для артефактів.

Перевагою SQLite є простота локального запуску. Користувачу не потрібно встановлювати PostgreSQL або іншу СКБД, а база створюється автоматично через `Base.metadata.create_all`. Для навчального та прототипного середовища це зменшує поріг відтворення системи.

Обмеженням такого підходу є слабша підтримка паралельних записів і відсутність складних механізмів аудиту. Якщо систему переносити у багатокористувацьке середовище, доцільно замінити SQLite на серверну базу даних і додати таблиці користувачів, ролей, журналу дій та версій датасетів.

У межах кваліфікаційної роботи обрана модель даних відповідає цілі: забезпечити простий, відтворюваний та достатньо надійний облік MLOps-циклів без ускладнення архітектури.

3.6 Frontend-інтерфейс користувача та експерта

Frontend реалізовано як React-застосунок. Головний компонент App.jsx керує станами результатів аналізу, помилок, режиму експерта і статусу тренування. Компонент InputSection.jsx відповідає за введення тексту та запуск аналізу, а service disinformationDetector.js ізолює HTTP-запит до backend.

Інтерфейс користувача дозволяє вставити статтю, натиснути кнопку аналізу і отримати список локалізованих технік (рисунок 3.1–3.2). Теги з backend залишаються англійськими, але frontend відображає українські або польські назви через locale-файли. Такий підхід відокремлює машинний контракт від мови інтерфейсу.

У застосунку передбачено перемикач мови. Якщо користувач змінює мову після аналізу, rawTags залишаються в стані, а useМемо повторно мапить їх на поточну локалізацію. Це покращує UX і демонструє правильне розділення даних та їх представлення.

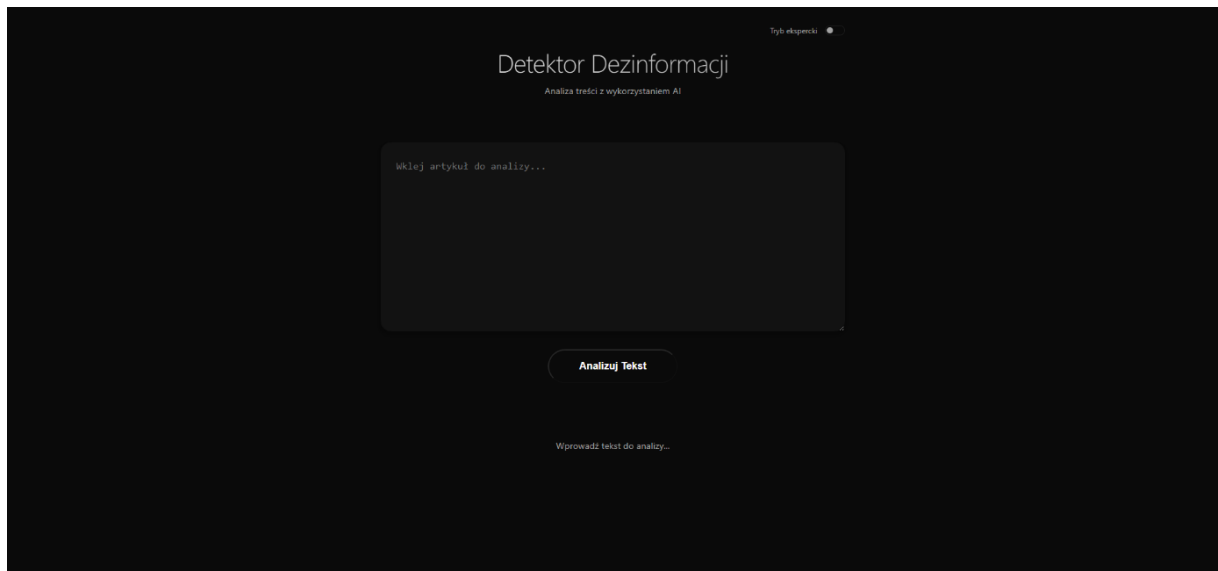


Рисунок 3.1 – Головне вікно застосунку для введення тексту

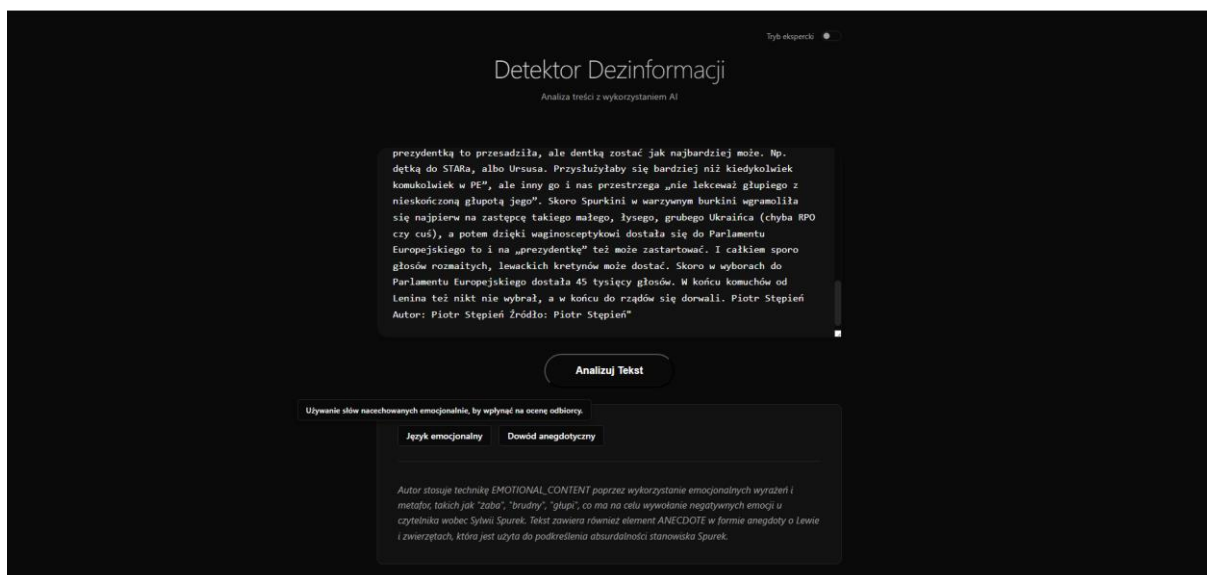


Рисунок 3.2 – Відображення результатів аналізу і пояснення

Режим експерта відкриває sidebar з полем завантаження датасету, прогрес-барями тренування й оцінювання, таблицею метрик і кнопкою розгортання моделі. WebSocket-з'єднання відкривається лише тоді, коли режим експерта активний, що зменшує зайві з'єднання у звичайному сценарії (рисунок 3.3).

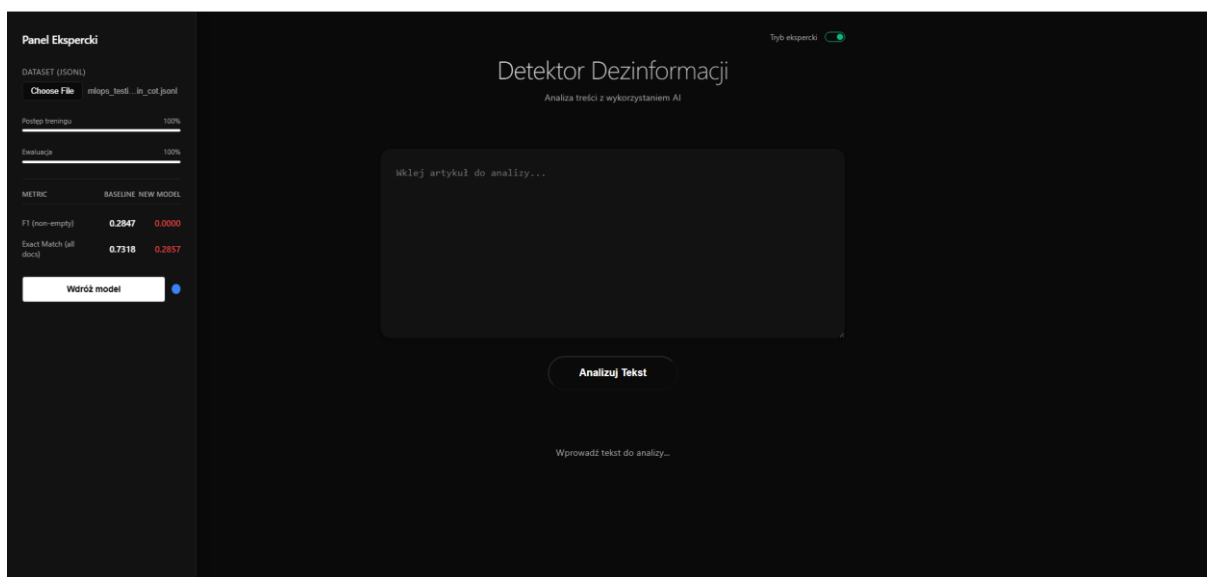


Рисунок 3.3 – Панель експерта для запуску MLOps-циклу

Frontend також містить попередження при розгортанні моделі з нижчим F1, ніж базовий варіант. Це важлива деталь Human-in-the-Loop: система не блокує експерта повністю, але явно підсвічує ризик погіршення якості.

3.7 Інсталяція, запуск і відновлення стану

Процедура інсталяції описана у README.md і автоматизована через setup.cmd. Скрипт перевіряє Python, npm, Ollama, ініціалізує submodule llama.cpp, створює backend/.venv, встановлює Node-залежності, перевіряє WSL2 і готує .venv-wsl для тренування.

Окремим кроком є підготовка каталогу model, що містить великі файли базової моделі та адаптерів. Через розмір ці файли не включені до Git-репозиторію. README.md описує очікувану структуру model, включно з bielik-4.5b-base, xai-adapter, dataset, benchmark-reports і Modelfile.

Запуск системи виконується через run_app.cmd. Скрипт стартує Ollama на порті 11434, backend FastAPI на порті 8000 і frontend Vite на порті 5173. Порядок запуску важливий, оскільки backend залежить від доступності Ollama, а frontend – від backend API.

Скрипт reset_state.cmd повертає систему до початкового стану: активним залишається xai-adapter, очищаються тимчасові артефакти тренування і відновлюється baseline-звіт. Це корисно для демонстрації та тестування, коли потрібно швидко повернути прототип до відомої конфігурації. Інженерна цінність цих скриптів полягає у зменшенні ручних дій.

3.8 Тестування та контроль якості

У backend/tests реалізовано unit- і integration-тести. Тести llm_processor перевіряють коректний JSON, fuzzy keys і відновлення з

невалідного тексту. Це покриває найбільш критичний ризик інференсу – нестабільний формат відповіді LLM.

Тести `orchestrator_helpers` перевіряють конвертацію Windows-шляхів у WSL, читання `baseline`-метрик у різних форматах звітів і коректне скидання стану кандидата після `deployment_success`. Ці перевірки важливі для MLOps-контуру, оскільки помилки стану можуть призвести до неправильного розгортання адаптера.

Integration-тести перевіряють доступність Ollama, готовність WSL, живий `uvicorn`-сервер, `WebSocket`-оновлення прогресу та запуск WSL-команди через `endpoint /training/upload`. Вони позначені маркерами `integration`, `ollama`, `wsl` і `websocket`, щоб їх можна було запускати лише за наявності локальних передумов.

`pytest.ini` вимикає зайві `plugin`-и, задає `testpaths=backend/tests` і налаштовує `norecursedirs` для `.venv`, `.venv-wsl`, `vendor` та `cache`-каталогів. Це робить тестовий запуск стабільнішим і зменшує ризик випадкового обходу великого `vendored llama.cpp`.

Тестування не замінює повну валідацію ML-моделі, але підвищує надійність програмної обв'язки. Саме обв'язка відповідає за те, щоб модельний результат був доставлений, очищений, показаний користувачу і за потреби використаний у наступному MLOps-циклі.

Для контролю достовірності результатів доцільно фіксувати не лише відповідь моделі, а й технічний контекст її отримання. Мінімальний журнал запиту може містити ідентифікатор запуску, назву активного адаптера, довжину вхідного тексту, час відповіді, режим обробки відповіді та ознаку застосованих відновлень. Такий журнал не призначений для кінцевого користувача, але потрібний розробнику й експерту для відтворення помилок.

Окремим критерієм практичної готовності є відтворюваність експериментів. Для кожного `benchmark`-запуску варто зберігати версію датасету, назву адаптера, параметри інференсу, дату запуску, розмір

тестової вибірки та спосіб агрегації метрик. Без такого опису однакові числові значення важко порівнювати між ітераціями, а покращення моделі може бути випадковим наслідком зміни умов оцінювання.

Під час аналізу тестів було враховано, що частина перевірок залежить від локальної інфраструктури: Ollama, WSL2, WebSocket-з'єднання і встановленої моделі. Тому integration-тести позначені окремими маркерами і мають пропускатися з явними причинами, якщо передумови відсутні. Така організація дозволяє відокремити помилку коду від відсутності зовнішнього сервісу.

Unit-тести мають покривати насамперед чисті функції та інженерні контракти: перетворення шляхів Windows у WSL, читання baseline-звітів, оновлення станів MLOps-циклу, локалізацію тегів і підготовку результату для frontend. Саме ці частини можна перевіряти детерміновано без запуску великої мовної моделі, тому вони є основою швидкого регресійного контролю.

Для повнішого контролю якості доцільно розширити набір тестових вхідних даних. Окремо мають перевірятися порожній текст, дуже довгий матеріал, текст із кількома мовами, нестандартні лапки, Markdown-фрагменти, URL-адреси, а також випадки паралельної роботи користувача й експерта. Це дозволить виявити дефекти інтерфейсу та backend-обробки до того, як вони проявляться під час демонстрації.

З погляду безпеки важливо враховувати prompt injection як частину тестового набору. Медіатекст може містити інструкції на кшталт прохання ігнорувати попередні правила або повернути відповідь в іншому форматі. Для такого сценарію перевіряється не лише результат класифікації, а й здатність системи зберегти службовий контракт між backend, моделлю та frontend.

Для тренувального контуру корисним є окремий аудит дій експерта. Запис TrainingRun має пов'язувати завантажений JSONL-файл, його контрольну суму, шлях до кандидата-адаптера, baseline-метрики, нові

метрики та рішення promote або reject. Це створює мінімальну історію походження моделі, яка потрібна для пояснення, чому саме певний адаптер став активним.

На рівні контролю якості окремо перевірявся інженерний контракт між моделлю і backend-сервісом. Для LLM-систем він є не менш важливим за сам prompt, оскільки визначає, які дані можуть бути безпечно передані до інтерфейсу користувача, збережені у журналі та використані в наступному MLOps-циклі.

4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ

4.1 Методика експериментального оцінювання

Експериментальна оцінка виконувалася на тестовому наборі MIPD обсягом 1521 документ. Оскільки задача є багатомітковою, документ може мати нуль, одну або кілька правильних міток. Це ускладнює інтерпретацію простих accuracy-метрик і потребує окремого аналізу структурної стабільності та класифікаційної якості.

Parsing Success Rate вимірює частку відповідей, які є коректним JSON у строгому режимі. Для системи з LLM ця метрика є базовою умовою застосовності: навіть змістовно правильний текст не може бути використаний frontend-інтерфейсом, якщо він не розбирається як структурований об'єкт.

Exact-Match Accuracy показує частку документів, у яких множина передбачених тегів повністю збігається з еталонною множиною. Метрика є суворою: зайвий або пропущений тег робить документ неправильним. Вона добре відображає користувацький сценарій, де список технік має бути максимально точним.

Mean Document-Level F1 для документів з непорожніми gold labels є основною змістовною метрикою. Вона не дозволяє моделі штучно покращити результат за рахунок великої кількості порожніх документів. Саме тому у звітах окремо наведено F1 для всіх документів і F1 без порожніх еталонних прикладів.

Порівнювалися три варіанти: базова модель без адаптера, Prototype Adapter і XAI Adapter. Перший варіант показує стартову здатність моделі, другий – ефект донавчання на класифікацію, третій – ефект додавання пояснюваності.

4.2 Порівняння варіантів моделі

Результати підтвердили, що базова модель без адаптера не є достатньо придатною для продукційного сценарію (таблиця 4.1).

Таблиця 4.1 – Результати експериментальної оцінки моделей на тестовому наборі MIPD

Метрика	No Adapter	Prototype Adapter	XAI Adapter
Parsing Success Rate (Strict JSON)	20,05%	99,93%	96,25%
Mean Document-Level F1 (непорожні документи)	0,0979	0,4912	0,2847
Exact-Match Accuracy	38,07%	72,91%	73,18%

Її Strict JSON PSR становить 20,05%, а Mean Document-Level F1 на непорожніх документах – 0,0979. Це означає, що без спеціалізованого донавчання модель погано дотримується формату і слабо відтворює таксономію маніпуляцій.

Prototype Adapter істотно покращив структурну стабільність: Strict JSON PSR досяг 99,93%. F1 на непорожніх документах становить 0,4912, що є найкращим результатом серед розглянутих варіантів. Це демонструє ефективність QLoRA-донавчання для задачі чистої класифікації технік маніпуляції.

XAI Adapter має Strict JSON PSR 96,25% і Exact-Match Accuracy 73,18%, але F1 на непорожніх документах знижується до 0,2847. Така поведінка показує, що інтеграція пояснень у вихід моделі створює додаткове навантаження і може відволікати модель від точного відтворення міток.

Порівняння Exact-Match показує, що Prototype і XAI близькі за повним збігом на всіх документах. Однак через велику частку порожніх прикладів ця метрика менш чутлива до помилок на документах з реальними

маніпуляціями. Саме тому F1 на непорожніх прикладах краще відображає практичний ризик пропуску техніки.

Отримані результати не означають, що XAI-підхід є невдалим. Вони показують, що якість пояснень і якість класифікації потрібно оптимізувати разом. Для прикладної системи пояснюваність має цінність, але її слід підтримати кращою фільтрацією синтетичних *rationale*, балансуванням даних і додатковою валідацією.

Щоб пояснити падіння F1 з 0,4912 для Prototype Adapter до 0,2847 для XAI Adapter, додатково виконано якісний аналіз 30 непорожніх тестових документів. Вибірка сформована детерміновано з *seed* = 43995 і пріоритетом рідкісних тегів. У ній XAI Adapter мав 28 строгих JSON-відповідей, 2 помилки парсингу, 2 повні збіги, 12 часткових збігів, 8 випадків лише *false negative* і 6 випадків повністю неправильного набору тегів; середній *document-level* F1 для цієї вибірки становив 0,3135.

Найпоказовіший клас помилок – консервативні відповіді, де модель повертає порожній список і *reasoning* на кшталт «текст має інформаційний характер», хоча *gold labels* містять маніпулятивні техніки. Наприклад, у матеріалі про вакцини з емоційно насиченими формулюваннями *gold labels* містили *CHERRY_PICKING*, *EMOTIONAL_CONTENT* і *REFERENCE_ERROR*, але XAI Adapter не повернув жодного тегу. Подібна помилка у тексті про 5G і COVID-19 пропустила *ANECDOTE*, *CHERRY_PICKING*, *EMOTIONAL_CONTENT* і *EXAGGERATION*.

Інший тип помилки полягає у тому, що *reasoning* локально виглядає правдоподібним, але список тегів не збігається з еталоном. У прикладі про пам'ятник Греті Тунберг модель пояснила *REFERENCE_ERROR* і *EXAGGERATION*, тоді як *gold labels* містили *EMOTIONAL_CONTENT*, *FALSE_CAUSE* і *WHATABOUTISM*. Це демонструє, що природномовне пояснення може бути зв'язним, але спрямованим на неправильну таксономічну інтерпретацію тексту.

4.3 Аналіз матриць помилок

Матриці помилок у звітах агрегують усі теги маніпуляції в універсальну бінарну схему: передбачено техніку або ні. Для базової моделі без адаптера характерні велика кількість false positives і false negatives (рисунок 4.1). Це підтверджує, що модель не має стабільного внутрішнього уявлення про задану таксономію без спеціального навчання.

Матриця помилок: базова модель без адаптера

Фактично: ні	14103	1779
Фактично: так	738	111
	Передбачено: ні	Передбачено: так

Рисунок 4.1 – Узагальнена матриця помилок базової моделі без адаптера

Prototype Adapter зменшує кількість помилок і збільшує true positives до 475 у зведеній матриці (рисунок 4.2). Особливо помітне покращення для EXAGGERATION і CHERRY_PICKING, які мають більше позитивних прикладів і виразніші мовні ознаки. Це узгоджується з очікуванням, що частотні та семантично явні категорії навчаються краще.

XAI Adapter має менше false positives, але більше false negatives порівняно з Prototype (рисунок 4.3). Така консервативність означає, що модель частіше не позначає техніку, коли еталон її містить. Для інформаційної безпеки це є суттєвим обмеженням, оскільки пропущена маніпуляція може бути небезпечнішою за зайве попередження.

Матриця помилок: Prototype Adapter

Фактично: ні	15311	571
Фактично: так	374	475
	Передбачено: ні	Передбачено: так

Рисунок 4.2 – Узагальнена матриця помилок Prototype Adapter**Матриця помилок: XAI Adapter**

Фактично: ні	15525	357
Фактично: так	589	260
	Передбачено: ні	Передбачено: так

Рисунок 4.3 – Узагальнена матриця помилок XAI Adapter

На рівні окремих тегів найскладнішими залишаються рідкісні категорії, зокрема `LEADING_QUESTIONS`, `QUOTE_MINING` і `STRAWMAN`. Їх важко навчити через малу кількість прикладів і потребу у глибшому дискурсивному контексті. Це вказує на необхідність балансування корпусу та цільового збору прикладів для слабких класів.

Матриці помилок також демонструють, що технічна стабільність JSON не гарантує змістовної точності. Модель може коректно повернути JSON, але помилитися у виборі тегів. Тому backend-нормалізація є необхідною, але не достатньою умовою якості всієї системи.

Якісний аналіз XAI Adapter деталізує цю картину на п'ять типів помилок: `parse failure`, `false negative only`, `wrong tags`, `partial overlap` і `exact match`. `Parse failure` означає, що пояснення взагалі не може бути використане без `repair`-механізму; у вибірці таких випадків було 2. `False negative only` є найнебезпечнішим для детектора, бо користувач отримує нейтральну відповідь там, де еталон містить техніки маніпуляції.

`Partial overlap` показує, що модель часто вловлює частину сигналу, але втрачає повну багатоміткову структуру документа. Наприклад, у матеріалі «Sikora o pandemii: Skończy się kiedy zarobią» `gold labels` містили сім технік, а модель повернула лише `EXAGGERATION` і `REFERENCE_ERROR`. Така відповідь є кориснішою за повний пропуск, але вона приховує `ANECDOTE`, `CHERRY_PICKING`, `EMOTIONAL_CONTENT`, `FALSE_CAUSE` і `STRAWMAN`.

`False positives` також пов'язані з `reasoning`: модель інколи додає тег, для якого може сформулювати пояснення, навіть якщо він не є еталонним. У прикладі з емоційним текстом про візит до гінеколога `gold label` був `EMOTIONAL_CONTENT`, але модель додала `EXAGGERATION`. Це підтверджує, що `reasoning` не можна оцінювати ізольовано від списку тегів: переконливий текст пояснення може маскувати зайву або пропущену категорію.

4.4 Аналіз продуктивності та ресурсних обмежень

Система спроектована для локального запуску на споживчому обладнанні з GPU NVIDIA. Базова модель Bielik-4.5B у форматі GGUF Q8_0 має розмір близько 4,7 ГБ, а LoRA-адаптер у GGUF – близько 190 МБ. Такий

розподіл дозволяє оновлювати адаптер без повторного розповсюдження всієї базової моделі.

Локальний інференс через Ollama забезпечує час відповіді порядку кількох секунд для одного запиту залежно від довжини тексту і GPU. Це прийнятно для інтерактивного інструмента аналізу статей, хоча не є достатнім для високонавантаженого потокового моніторингу без серверного масштабування.

Тренування виконується у WSL2 через Unsloth і QLoRA. У локальному експертному сценарії короткий цикл донавчання на зменшеному наборі може тривати близько кількох хвилин, тоді як повне дослідне навчання на великому корпусі потребує Google Colab або іншого GPU-середовища. Тому система розрізняє production inference і training workflow.

Використання WSL2 є практичним компромісом. Багато ML-бібліотек, зокрема bitsandbytes та Unsloth, краще підтримують Linux-середовище, тоді як frontend і backend зручно запускати у Windows. Оркестратор приховує цей розрив, автоматично конвертуючи шляхи та запускаючи процеси у WSL.

Головним ресурсним обмеженням залишається довгий контекст. Modelfile задає num_ctx=16384, але реальна швидкість і стабільність залежать від доступної VRAM. Для майбутнього масштабування доцільним є перехід на серверні GPU з більшим обсягом пам'яті та підтримка паралельної обробки запитів.

Ще одним фактором є довжина медіатекстів. Довгі статті можуть містити кілька незалежних фрагментів, з яких лише частина є маніпулятивною. Модель, що повертає один список тегів для всього документа, може втрачати локальність. У майбутньому доцільно додати сегментацію тексту та прив'язку пояснень до конкретних фрагментів.

Для довгих статей можливим є ієрархічний режим аналізу: спочатку система оцінює окремі абзаци або фрагменти, а потім агрегує знайдені

техніки на рівні всього матеріалу. Такий підхід краще відповідає природі медіатекстів, де маніпуляція може з'являтися локально, а не в кожному реченні.

Під час експлуатації потрібно обмежувати довжину вхідного тексту й повідомляти користувачу про обрізання або неможливість обробки. Якщо система мовчки скорочує матеріал, пояснення може стосуватися лише частини документа, хоча користувач очікує аналізу всього тексту. Коректна поведінка полягає у прозорому повідомленні про обсяг аналізу.

У майбутній версії можна розглянути каскадну архітектуру: швидкий легкий класифікатор попередньо відбирає потенційно проблемні фрагменти, а LLM аналізує лише складні випадки. Такий підхід зменшить навантаження на GPU і дозволить обробляти більші потоки текстів без втрати пояснюваності там, де вона справді потрібна.

Під час аналізу продуктивності важливо розрізняти холодний і повторний запуск моделі. Перший запит після створення моделі в Ollama включає завантаження ваг і адаптера, тому може бути помітно повільнішим за наступні звернення. Для коректної оцінки часу відповіді benchmark має фіксувати тип запуску, довжину тексту і параметри інференсу.

Для інтерактивного сценарію більш важливою є не максимальна пропускна здатність, а передбачуваність очікування. Якщо аналіз одного матеріалу триває кілька секунд, frontend має показувати стан виконання і не створювати враження зависання. Це особливо важливо для експертного режиму, де тренування та оцінювання займають довше, ніж звичайний аналіз статті.

Оцінка зручності супроводу показує, що найскладнішою частиною системи є межа між Windows і WSL2. Саме тут виникають проблеми шляхів, callback-адрес, доступності Python, драйверів GPU і сумісності ML-бібліотек. Допоміжні функції перетворення шляхів і визначення WSL-host адреси зменшують кількість ручних дій і роблять запуск тренування відтворюваним.

Для подальшої експлуатації доцільно додати короткий smoke-test після запуску `run_app.cmd`. Він має перевіряти доступність Ollama, backend endpoint `/analyze`, WebSocket експертної панелі та наявність активного адаптера. Така перевірка швидко відокремить помилки середовища від помилок моделі й спростить демонстрацію системи на іншій машині.

4.5 Сценарії використання системи

Перший сценарій – індивідуальний аналіз статті. Користувач вставляє текст, натискає кнопку аналізу і отримує перелік знайдених технік. Пояснення reasoning допомагає зрозуміти, чому система вважає текст маніпулятивним або нейтральним. Такий сценарій корисний у медіаграмотності та журналістській підготовці.

Другий сценарій – робота аналітика або редактора. Експерт може використовувати систему як попередній фільтр, що швидко підсвічує потенційні маніпуляції. Остаточне рішення залишається за людиною, але автоматична підказка економить час на первинному сортуванні матеріалів.

Третій сценарій – адаптація до нових наративів. Якщо аналітик накопичив нові приклади маніпуляцій, він може сформувати JSONL-файл, завантажити його в експертному режимі та запустити донавчання. Після benchmark система покаже, чи новий адаптер покращив метрики відносно baseline.

Четвертий сценарій – демонстраційне й навчальне використання. Завдяки локальному запуску, відкритій структурі коду та наявності скриптів налаштування систему можна використовувати як навчальний приклад MLOps для LLM. Вона показує весь цикл від даних до інтерфейсу користувача.

Обмеження сценаріїв також чіткі. Система не повинна використовуватися як єдине джерело істини для юридично значущих рішень. Її результат є аналітичною підказкою, а не остаточним

фактчекінговим висновком. Це обмеження має бути явно зазначене у разі практичного впровадження.

З організаційного погляду система може бути впроваджена поступово. На першому етапі вона працює як допоміжний інструмент для навчальних завдань або внутрішнього аналізу. На другому етапі експерти використовують її для збору помилок і формування нового датасету. На третьому етапі метрики, журнали й контрольні набори дозволяють приймати рішення про оновлення моделі.

Для журналістського сценарію пріоритетом може бути висока чутливість, щоб не пропустити ризиковий матеріал. Для освітнього сценарію важливішою може бути якість пояснення, навіть якщо частина тегів неідеальна. Тому майбутній інтерфейс може мати профілі використання, які змінюють поріг або спосіб представлення результату.

З погляду користувача, корисним буде збереження прикладів нейтральних текстів, бо вони навчають модель не перебільшувати ризик. Надто агресивний детектор, який бачить маніпуляцію всюди, може бути так само шкідливим, як і модель, що все пропускає. Баланс між false positives і false negatives повинен налаштовуватися під сценарій застосування.

У frontend-частині корисним розвитком буде візуальне підсвічування речень, на які посилається reasoning. Це потребує зміни формату виходу моделі: окрім тегів і пояснення, вона повинна повертати evidence spans або індекси речень. Така функція наблизить систему до повноцінного ХАІ-інструмента для редакційної роботи.

Також варто розрізняти пояснення для аналітика і пояснення для широкої аудиторії. Аналітик може працювати з назвами технік, метриками, порогами і деталями адаптера. Звичайному читачеві потрібне коротше формулювання, яке пояснює, чому саме фрагмент може бути маніпулятивним, і не створює хибного враження остаточного вердикту.

Ще одним можливим розвитком є введення режиму порівняння моделей. Експерт міг би запускати один текст через базовий адаптер,

Prototype Adapter і XAI Adapter, а потім бачити різницю у тегах і reasoning. Такий режим допоможе приймати обґрунтовані рішення про promote і краще розуміти поведінку кожного варіанта.

Систему також можна доповнити режимом експорту звіту аналізу. Користувач міг би зберегти текст, знайдені техніки, пояснення, дату, версію моделі й попередження про автоматичний характер висновку. Такий звіт буде корисний для навчальних занять або внутрішньої редакційної документації.

Для навчального середовища корисно залишати в системі приклади помилкових відповідей моделі. Вони показують, що автоматичний аналіз не є безпомилковим, і водночас допомагають пояснювати студентам різницю між синтаксично коректною відповіддю, семантично правильною класифікацією та якісним природномовним поясненням.

Для застосування у навчальному процесі система може використовуватися як демонстратор типових логічних помилок. Студент або слухач курсу медіаграмотності може порівняти автоматичну відповідь із власним аналізом. Такий режим не потребує ідеальної точності моделі, але потребує зрозумілого reasoning і прозорого переліку категорій.

Окремої уваги потребує питання інтерпретації негативного результату. Якщо система повертає порожній список технік, це не означає гарантованої правдивості тексту. Це означає лише, що в межах заданої таксономії і поточної здатності моделі не виявлено маніпулятивних ознак. Таке розмежування важливе для етичного використання системи.

4.6 Порівняння з аналогами та практична цінність

Порівняно з комерційними API-сервісами, розроблена система має перевагу локальності. Тексти не передаються сторонньому постачальнику, що важливо для редакцій, дослідницьких груп або навчальних середовищ,

де матеріали можуть бути чутливими. Відсутність плати за кожен запит також робить прототип зручним для експериментів.

Порівняно з класичними BERT-класифікаторами, система повертає не лише мітки, а й пояснення природною мовою. Це підвищує прозорість і створює освітній ефект. Користувач бачить не тільки назву техніки, а й логіку її застосування в тексті.

Порівняно з ручним фактчекінгом, система значно швидша і масштабованіша, але поступається глибиною перевірки джерел. Вона не замінює експертний аналіз фактів, а доповнює його виявленням логічних і риторичних ознак маніпуляції. Такий розподіл ролей є реалістичним для практичного застосування.

Унікальною частиною роботи є MLOps-петля з можливістю локального донавчання і hot-swap адаптера. Багато демонстраційних ML-проектів зупиняються на inference demo, тоді як ця система реалізує механізм оновлення моделі в тому самому інтерфейсі, де виконується аналіз.

Практична цінність полягає в інженерній завершеності прототипу. Він має frontend, backend, модельний сервер, тести, скрипти запуску, датасети, звіти оцінювання та документацію. Це дозволяє розглядати роботу як комплексний програмний артефакт, а не лише як експеримент з LLM.

У роботі також важливо розрізняти дезінформацію і маніпуляцію. Не кожний маніпулятивний текст містить неправдиві факти, і не кожна фактична помилка є свідомою дезінформацією. Обрана таксономія зосереджується на техніках аргументаційного впливу, тому результат системи слід інтерпретувати саме в цьому контексті.

У навчальному контексті розробка може бути використана як приклад поєднання дисциплін: NLP, веброзробки, баз даних, операційних систем, тестування і MLOps. Це відповідає профілю освітньої програми «Штучний інтелект», де важливо не лише знати модель, а й уміти вбудувати її у працюючу систему.

З погляду супроводу програмного продукту важливо, що критичні параметри винесено у зрозумілі місця: назва моделі в backend, Modelfile для Ollama, requirements для середовищ і locale-файли для інтерфейсу. Це спрощує пошук помилок, коли система переноситься на іншу машину або коли змінюється активний адаптер.

Важливим результатом проєктування є відокремлення дослідних ноутбуків від робочого застосунку. Ноутбуки залишаються місцем експериментів з даними та навчанням, тоді як backend і frontend реалізують стабільний сценарій використання. Такий поділ зменшує залежність демонстраційної системи від ручного виконання комірок.

Інженерна цінність роботи полягає у поєднанні кількох складних компонентів у працездатний цикл. Модель, frontend, backend, WSL-тренування, GGUF-конверсія, Ollama і WebSocket-статуси працюють як єдина система. Саме інтеграція цих компонентів створює результат, який можна демонструвати й розвивати.

Наукова цінність роботи полягає у фіксації практичного конфлікту між пояснюваністю та класифікаційною ефективністю. Отриманий розрив між Prototype і XAI Adapter показує, що просто додати reasoning до відповіді недостатньо. Потрібна методологія навчання, яка перевіряє узгодженість пояснення з міткою і текстом.

Результати експериментів також свідчать про потребу розділяти два типи якості: форматну і семантичну. Prototype Adapter майже ідеально дотримується JSON, але це не означає абсолютної точності тегів. XAI Adapter генерує пояснення, але має нижчий F1. Повна якість системи виникає тільки тоді, коли обидва аспекти контролюються разом.

Окремий інтерес становить порівняння локального і хмарного підходів. Хмарні API можуть бути сильнішими за якість reasoning, але вони не дають повного контролю над моделлю і даними. Локальна система вимагає більше інженерного налаштування, зате надає контроль над версіями, prompt, адаптером і політикою обробки текстів.

4.7 Обмеження та напрями подальшого розвитку

Першим обмеженням є якість синтетичних пояснень. Модель-вчитель могла генерувати неідеальні або надто узагальнені *rationale*, які потім наслідувала модель-учень. Для зменшення цього ризику потрібна вибіркова ручна валідація, автоматична оцінка пояснень і фільтрація слабких прикладів.

Другим обмеженням є дисбаланс класів. Рідкісні техніки мають менше прикладів, тому модель гірше їх розпізнає. Подальша робота має включати цільове збирання даних, *oversampling*, *hard negative mining* і аналіз перетину близьких категорій.

Третім обмеженням є локальна продуктивність. Споживчий GPU достатній для прототипу, але не для багатокористувацького сервісу. Масштабування потребує серверного інференсу, черг задач, кешування, контролю завантаження та, можливо, використання меншого класифікаційного каскаду перед LLM.

Четвертим обмеженням є безпека експертного режиму. Поточний прототип орієнтований на локальне використання і не реалізує повноцінну автентифікацію, RBAC та аудит усіх змін. Перед продукційним впровадженням ці механізми мають бути додані.

Перспективним напрямом є LLM-as-a-Judge для оцінки пояснень. Сильніша модель може перевіряти, чи *reasoning* справді відповідає тексту і тегам, а людина може ревізувати вибірку таких оцінок. Це дозволить масштабувати контроль якості NLE без повністю ручної розмітки.

Іншим напрямом є підтримка кількох мов. Поточний модельний контур орієнтований на польськомовні тексти, але *frontend* уже має українську локалізацію. Подальша робота може включати український корпус дезінформації, окремий адаптер або багатомовну модель для аналізу матеріалів українських медіа.

З точки зору користувацького досвіду важливо, щоб система не переважувала інтерфейс технічними метриками у звичайному режимі. Саме тому метрики F1 і Exact Match винесені в панель експерта. Звичайний користувач бачить лише практичний результат аналізу, тоді як інженер отримує достатньо інформації для оцінки моделі.

Поточна система підтримує локалізацію назв технік, але reasoning генерується польською моделлю і тому переважно польською мовою. Це коректно для польськомовного корпусу, однак для українського використання потрібне або окреме українське донавчання, або post-processing-переклад пояснення. При цьому переклад не повинен змінювати зміст аргументації.

Для кращого аналізу помилок benchmark має зберігати не лише агреговані метрики, а й перелік прикладів із false positives і false negatives. Такий звіт допоможе експерту зрозуміти, які типи текстів модель плутає, і сформулювати наступний навчальний набір не випадково, а на основі виявлених слабких місць.

Під час подальшого розвитку доцільно додати версіонування адаптерів. Зараз система може оновити активний адаптер, але повний production-процес потребує можливості швидкого rollback до попередньої версії. Це можна реалізувати через каталог deployed adapters, таблицю версій у базі даних і явне зберігання baseline-звіту для кожної версії.

При впровадженні в редакційний процес необхідно визначити, як зберігаються аналізовані тексти. Локальний запуск мінімізує передачу даних назовні, але якщо результати будуть записуватися у базу, потрібно врахувати політики приватності, строки зберігання і можливість видалення матеріалів на вимогу користувача.

Етичне використання системи вимагає прозорого повідомлення про її обмеження. Інтерфейс має пояснювати, що результат є автоматичною оцінкою, а не офіційною експертизою. Це особливо важливо, якщо

інструмент буде використано для навчання або редакційної роботи з реальними авторами текстів.

Окремої уваги потребує локалізація назв маніпулятивних технік. Англомовні терміни з датасетів не завжди мають однозначні українські відповідники, а буквальный переклад може бути незручним для користувача. Тому система повинна зберігати внутрішні стабільні ідентифікатори категорій, але показувати в інтерфейсі локалізовані назви і короткі пояснення.

Для розширення на українську мову потрібно сформувати окрему таксономію або перевірити відповідність наявних одинадцяти тегів українському медіапростору. Частина технік є універсальною, але мовні маркери, політичний контекст і типові джерела можуть відрізнятися. Тому просте перенесення польського адаптера не гарантує якісного результату.

Окремим напрямом є виявлення невизначеності. Поточний формат повертає список тегів без *confidence score*. Для користувача було б корисно знати, коли модель впевнена, а коли результат є сумнівним. Проте *confidence* для генеративної LLM складно калібрувати, тому це потребує окремого дослідження.

Модельне оцінювання варто доповнити людською експертною ревізією. Автоматичні метрики порівнюють теги з еталонним набором, але не оцінюють переконливість *reasoning* для людини. Невелика вибірка ручних оцінок допоможе зрозуміти, чи справді пояснення корисні, чи лише формально супроводжують відповідь.

Окремим напрямом розвитку є контроль якості *reasoning*. У поточній реалізації пояснення є текстовим полем, яке допомагає користувачу зрозуміти причину класифікації. Проте для суворішої ХАІ-оцінки можна додати рубрику експертного оцінювання: відповідність пояснення знайденому тегу, наявність посилання на конкретний фрагмент тексту, відсутність вигаданих фактів і зрозумілість формулювання.

З технічного погляду важливо не змішувати журнали користувацької взаємодії з навчальними даними автоматично. Реальні тексти можуть містити персональні дані або матеріали з обмеженим доступом. Тому будь-який перехід від запиту користувача до датасету повинен мати явну дію експерта, механізм очищення тексту і зрозуміле пояснення подальшого використання прикладу.

У подальших версіях можна додати механізм ручного підтвердження результату без негайного тренування. Експерт міг би позначати відповідь як корисну, частково корисну або помилкову, а система накопичувала б такі оцінки окремо від навчального корпусу. Це дозволить спочатку аналізувати якість моделі на реальних запитах.

Для експертного циклу важливо мінімізувати тертя під час додавання нових прикладів. Якщо експерту потрібно вручну редагувати складні JSONL-файли, зростає ризик помилок і зменшується регулярність оновлення датасету. Тому майбутній інтерфейс може мати форму розмітки, яка автоматично формує валідний запис для тренування.

Після накопичення достатньої кількості експертних виправлень можна запровадити політику відбору даних для нового тренування. Не кожний приклад однаково корисний: найбільшу цінність мають випадки помилок моделі, неоднозначні тексти й приклади з рідкісними техніками. Такий активний підхід до формування навчального набору ефективніший за просте додавання всіх доступних даних.

Важливо також контролювати, щоб модель не генерувала нові неузгоджені категорії. Поточний backend може показати невідомий тег як потенційну галюцинацію, але для тренування краще не допускати таких тегів у датасет. Валідація корпусу перед SFT має бути обов'язковим етапом майбутньої версії.

Для підтримки якості даних доцільно додати інструмент перевірки JSONL перед запуском тренування. Він має перевіряти наявність input, output, валідність JSON усередині output, допустимість тегів і відсутність

порожніх або надто довгих прикладів. Це зменшить ризик, що помилка у файлі експерта зіпсує тренувальний запуск.

При розширенні набору даних слід враховувати не лише кількість прикладів, а й їхню аргументаційну різноманітність. Для однієї й тієї самої техніки можуть існувати різні мовні прояви: від прямого емоційного тиску до тонкого контекстного перекручення. Якщо корпус містить лише очевидні приклади, модель навчиться виявляти грубі випадки, але пропускатиме складніші.

4.8 Додатковий аналіз впровадження

Додатковий аналіз впровадження показує, що роботу доцільно оцінювати не лише за метриками моделі, а й за простежуваністю виконаних інженерних задач. Для цього можна скласти матрицю відповідності між завданнями кваліфікаційної роботи, реалізованими модулями, тестами, експериментальними звітами та рисунками у пояснювальній записці. Така матриця допоможе під час захисту показати, що кожна заявлена задача має конкретний результат.

З операційного погляду система має щонайменше три ролі: звичайний користувач аналізує текст, експерт керує датасетами й адаптерами, а адміністратор відповідає за середовище виконання. У прототипі ці ролі частково збігаються, але їхнє розмежування корисне для майбутнього розвитку, оскільки різні дії мають різний рівень ризику.

Для керування моделями доцільно запровадити життєвий цикл версії: `candidate`, `evaluated`, `promoted`, `rejected` і `archived`. `Candidate`-адаптер створюється після тренування, `evaluated` має `benchmark`-звіт, `promoted` стає активним у `Modelfile`, `rejected` зберігається лише для аналізу помилок, а `archived` може бути використаний для `rollback`. Такий підхід робить MLOps-процес контрольованішим.

Окремим елементом впровадження є процедура підготовки нових навчальних прикладів. Експертні виправлення мають проходити валідацію формату, перевірку допустимих тегів, очищення службових або персональних даних і короткий перегляд узгодженості між текстом, мітками та поясненням. Це зменшує ризик того, що локальне донавчання погіршить модель через неякісний корпус.

Валідність експериментальних висновків має власні обмеження. Тестовий набір MIPD відображає конкретну мову, домен і таксономію, тому результати не можна автоматично переносити на всі медіатексти. Крім того, велика частка нейтральних документів впливає на Exact-Match, а синтетичні пояснення не дорівнюють експертним *rationale*. Ці фактори потрібно прямо враховувати при інтерпретації метрик.

Корисним артефактом наступної ітерації може бути каталог типових помилок. Для кожної групи варто зберігати короткий приклад, *gold labels*, прогноз моделі, тип дефекту і можливу причину. Такий каталог перетворює *error analysis* із разового опису на робочий інструмент планування нових даних і *regression*-тестів.

Після впровадження важливо контролювати не самі тексти користувачів, а агреговані технічні показники: частку помилок обробки, середній час відповіді, кількість невідомих тегів, частку порожніх результатів і частоту ручних виправлень експерта. Такі сигнали можуть вказувати на дрейф даних або невдалу версію адаптера без збереження зайвого вмісту користувацьких матеріалів.

Практична експлуатація потребує явних правил зберігання даних. Для навчального режиму достатньо тимчасового збереження прикладів, тоді як редакційний сценарій може вимагати строків зберігання, права видалення матеріалів і відокремлення журналів від навчального корпусу. Це організаційне рішення має бути прийняте до використання системи з реальними чутливими текстами.

Перед передачею системи іншому користувачу доцільно мати короткий checklist готовності: встановлено Ollama, доступний потрібний GGUF-файл, створено backend/.venv і .venv-wsl, відкриваються порти backend і frontend, проходить тестовий /analyze-запит, а baseline-звіт відповідає активному адаптеру. Такий список краще відповідає практичному характеру роботи, ніж лише опис архітектури.

У навчальному процесі система може використовуватися як лабораторний приклад повного LLM-життєвого циклу. Студент може простежити шлях від датасету і синтетичних пояснень до адаптера, API, frontend-відображення, benchmark-звіту і рішення про розгортання. Саме така наскрізна демонстрація відповідає вимозі показати не тільки теоретичну основу, а й практичний стан розробки.

Таким чином, додатковий аналіз впровадження уточнює перехід від дослідного прототипу до контрольованого інструмента. Подальша цінність системи залежить не лише від підвищення F1, а й від якості даних, трасованості версій, прозорих процедур експертного рішення, відтворюваних експериментів і зрозумілих обмежень для користувача.

ВИСНОВКИ

У кваліфікаційній роботі вирішено задачу проектування та реалізації локальної системи виявлення дезінформації та логічних помилок у медіатекстах із використанням пояснюваного штучного інтелекту. Результатом є працездатний інженерний прототип, який поєднує LLM-інференс, вебінтерфейс, backend-оркестрацію та MLOps-петлю донавчання.

Проведено аналіз предметної галузі, існуючих комерційних, академічних і експертних підходів до детекції дезінформації. Показано, що ключовими недоліками аналогів є залежність від зовнішнього API, відсутність природномовного пояснення або слабка масштабованість ручної перевірки. Це обґрунтувало вибір локальної LLM з механізмом XAI.

Розроблено методику підготовки даних на основі MIPD і синтетичних Natural Language Explanations. Для генерації пояснень використано підхід Teacher-Student із моделлю Qwen-2.5-7B-Instruct як моделлю-вчителем і Bielik-4.5B як моделлю-учнем. Такий підхід дозволив сформувати навчальні приклади, що містять як мітки, так і reasoning.

Виконано тонке налаштування моделі засобами QLoRA та Unsloth. Використання LoRA-адаптерів дозволило адаптувати модель без повного перенавчання всіх параметрів, а GGUF-конверсія і Ollama забезпечили локальний інференс. Запропонована схема дає змогу оновлювати адаптер без повторного розповсюдження базової моделі.

Реалізовано backend FastAPI з endpoint-ами аналізу, завантаження даних, контролю статусу, WebSocket-сповіщень і розгортання моделі. Окремий модуль `normalize_llm_response` підвищує стійкість системи до невалідного JSON, fuzzy keys і помилок у назвах тегів. Це вирішує практичну проблему нестабільності структурованого виводу LLM.

Реалізовано frontend React/Vite з режимом користувача і режимом експерта. Користувач може аналізувати медіатекст і бачити знайдені техніки з поясненням, а експерт – завантажувати JSONL-набір, стежити за

тренуванням, порівнювати метрики і приймати рішення про розгортання нового адаптера.

Проведено експериментальне оцінювання трьох варіантів моделі на тестовому наборі MIPD. Базова модель без адаптера показала низький Strict JSON PSR 20,05% і F1 0,0979 на непорожніх документах. Prototype Adapter підвищив PSR до 99,93% і F1 до 0,4912. XAI Adapter досяг PSR 96,25%, Exact-Match 73,18%, але F1 0,2847, що показує практичний компроміс між пояснюваністю та класифікаційною точністю.

Порівняння з аналогами показало, що система має переваги локальності, прозорості та можливості донавчання. Водночас вона не замінює повний фактчекінг людиною і повинна використовуватися як інструмент підтримки аналізу. Найважливішими обмеженнями є якість синтетичних пояснень, дисбаланс класів, ресурсні вимоги LLM та потреба у формальній валідації reasoning.

Матеріали роботи можуть використовуватися у медіамоніторингу, навчальному процесі з медіаграмотності та як основа для подальших досліджень локальних XAI-систем. Подальший розвиток доцільно спрямувати на українськомовний корпус, LLM-as-a-Judge для перевірки пояснень, сувору JSON Schema-валідацію, RBAC для експертного режиму і серверне масштабування інференсу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ, 2015.
2. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ, 2015.
3. Харківський національний університет радіоелектроніки. Методичні вказівки до організації виконання та захисту кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти спеціальності 122 Комп'ютерні науки. Харків: ХНУРЕ, 2026. 44 с.
4. Attention is all you need / A. Vaswani et al. *arXiv*. 2023. URL: <https://doi.org/10.48550/arXiv.1706.03762> (date of access: 15.06.2026).
5. Miller T. Explanation in artificial intelligence: insights from the social sciences. *Artificial intelligence*. 2019. Vol. 267. P. 1–38. URL: <https://doi.org/10.1016/j.artint.2018.07.007> (date of access: 15.06.2026).
6. XAI–Explainable artificial intelligence / D. Gunning et al. *Science robotics*. 2019. Vol. 4, no. 37. P. eaay7120. URL: <https://doi.org/10.1126/scirobotics.aay7120> (date of access: 15.06.2026).
7. E-SNLI: natural language inference with natural language explanations / O.-M. Camburu et al. *arXiv*. 2018. URL: <https://doi.org/10.48550/arXiv.1812.01193> (date of access: 15.06.2026).
8. Lei T., Barzilay R., Jaakkola T. Rationalizing neural predictions. *Proceedings of the 2016 conference on empirical methods in natural language processing*, Austin, Texas. Stroudsburg, PA, USA, 2016. URL: <https://doi.org/10.18653/v1/d16-1011> (date of access: 15.06.2026).
9. Chain-of-Thought prompting elicits reasoning in large language models / J. Wei et al. *arXiv*. 2023. URL: <https://doi.org/10.48550/arXiv.2201.11903> (date of access: 15.06.2026).
10. Distilling step-by-step! Outperforming larger language models with less training data and smaller model sizes / C.-Y. Hsieh et al. *Findings of the*

association for computational linguistics: ACL 2023, Toronto, Canada. Stroudsburg, PA, USA, 2023. URL: <https://doi.org/10.18653/v1/2023.findings-acl.507> (date of access: 15.06.2026).

11. LoRA: low-rank adaptation of large language models / E. J. Hu et al. *arXiv*. 2021. URL: <https://doi.org/10.48550/arXiv.2106.09685> (date of access: 15.06.2026).

12. QLoRA: efficient finetuning of quantized llms / T. Dettmers et al. *arXiv*. 2023. URL: <https://doi.org/10.48550/arXiv.2305.14314> (date of access: 15.06.2026).

13. Scaling laws of rope-based extrapolation / X. Liu et al. *arXiv*. 2024. URL: <https://doi.org/10.48550/arXiv.2310.05209> (date of access: 15.06.2026).

14. Speakleash/Bielik-4.5B-v3. *Hugging Face – The AI community building the future*. URL: <https://huggingface.co/speakleash/Bielik-4.5B-v3> (date of access: 14.06.2026).

15. Qwen2.5 technical report / A. Yang et al. *arXiv*. 2025. URL: <https://doi.org/10.48550/arXiv.2412.15115> (date of access: 15.06.2026).

16. MIPD: exploring manipulation and intention in a novel corpus of polish disinformation / A. Modzelewski et al. *Proceedings of the 2024 conference on empirical methods in natural language processing*, Miami, Florida, USA. Stroudsburg, PA, USA, 2024. P. 19769–19785. URL: <https://doi.org/10.18653/v1/2024.emnlp-main.1103> (date of access: 15.06.2026).

17. Transformers. *Hugging Face – The AI community building the future*. URL: <https://huggingface.co/docs/transformers> (date of access: 14.06.2026).

18. Unsloth documentation. *Unsloth Docs*. URL: <https://docs.unsloth.ai/> (date of access: 14.06.2026).

19. Ollama documentation. *Ollama*. URL: <https://ollama.com/> (date of access: 14.06.2026).

20. Gerganov G. ggml-org/llama.cpp: LLM inference in C/C++. *GitHub*. URL: <https://github.com/ggml-org/llama.cpp> (date of access: 14.06.2026).
21. FastAPI documentation. *FastAPI*. URL: <https://fastapi.tiangolo.com/> (date of access: 14.06.2026).
22. React documentation. *React*. URL: <https://react.dev/> (date of access: 14.06.2026).
23. SQLite documentation. *SQLite Home Page*. URL: <https://www.sqlite.org/docs.html> (date of access: 14.06.2026).
24. SQLAlchemy 2.0 documentation. *SQLAlchemy Documentation*. URL: <https://docs.sqlalchemy.org/> (date of access: 14.06.2026).
25. Scikit-learn 1.9.0 documentation. *scikit-learn: machine learning in Python*. URL: <https://scikit-learn.org/stable/> (date of access: 14.06.2026).
26. Windows subsystem for linux documentation. *Microsoft Learn: Build with answers in reach*. URL: <https://learn.microsoft.com/windows/wsl/> (date of access: 14.06.2026).

ДОДАТОК А

Відомість кваліфікаційної роботи

Позначення					Найменування		Дод. відомості	
					Текстові документи			
1.					Пояснювальна записка		62 с.	