Міністерство освіти і науки України
Харківського національного університету імені В.Н. Каразіна
Навчально-наукового інституту комп'ютерних наук та штучного інтелекту
Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Комп'ютерні науки»

Рекомендовано до захисту:

протокол засідання кафедри

№ ___ від ____.____.2025 р.

Завідувач кафедри _____

_____ _____

(підпис) (прізвище та ініціали)

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему: «Аналіз коду програмного забезпечення з використанням методів та засобів штучного інтелекту»

Захищено на засіданні ЕК № _____
протокол № ____ від ____.____.2025 р.
Оцінка _____ / _____
Голова ЕК
_____ _____
(підпис) (прізвище та ініціали)

Виконав:
студент 4 курсу, групи КС- 41
Спеціальності:
122 Комп'ютерні науки
(шифр і назва спеціальності підготовки)
Жигайло Іван Віталійович
(прізвище, ім'я та по батькові, підпис)
Науковий керівник ст. викл. Трусов М.А.
(ступень, звання, прізвище та ініціали, підпис)
Рецензент _____
(ступень, звання, прізвище та ініціали, підпис)

Харків 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра кібербезпеки інформаційних систем, мереж і технологій

Рівень вищої освіти (освітньо-кваліфікаційний рівень) бакалавр

Галузь знань: 12 Інформаційні технології

Спеціальність: 122 Комп'ютерні науки
Освітня програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
завідувача кафедри _____
Олег ОЛЕШКО
«____» _____20___ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЕКТ)

Жигайло Іван Віталійович
(прізвище, ім'я, по батькові студента)

1. Тема роботи Аналіз коду програмного забезпечення з використанням методів та засобів штучного інтелекту
керівник роботи Трусов Михайло Андрійович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "16" квітня 2025 року №4101-5/962

2. Строк подання студентом роботи 01.06.2025

3. Перелік питань, які потрібно розробити
1. Визначення мети та завдань кваліфікаційної роботи: Опис об'єкта та предмета дослідження, формулювання основної мети роботи та декомпозиція її на конкретні завдання.

2. Аналіз існуючих методів та засобів аналізу програмного коду: Проведення огляду літератури, що включає традиційні методи (статичний, динамічний, ручний аналіз) та сучасні підходи на основі ШІ (LLM, GNN). Обґрунтування вибору OpenAI API як ключової технології

3. Формалізація вимог до вебсервісу "CodeAnalyzer": Збір та структурування функціональних (завантаження коду, аналіз, порівняння, управління історією) та нефункціональних (продуктивність, надійність, зручність використання) вимог до системи.

4. Проєктування архітектури веб-додатку: Розробка загальної архітектури клієнт-серверного рішення, визначення основних програмних модулів (маршрутизатор, обробники запитів, утиліти для взаємодії з API, система зберігання даних) та зв'язків між ними.

5. Розробка модуля інтелектуального аналізу коду: Реалізація логіки формування промптів до OpenAI API для аналізу коду за різними критеріями (помилки, вразливості, продуктивність, стиль) та парсингу структурованих JSON-відповідей.

6. Розробка модуля порівняння версій коду: Створення алгоритму для генерації відмінностей (diff) між двома версіями коду та формування спеціалізованого промпту до ШІ для інтелектуальної оцінки внесених змін.

7. Реалізація користувацького інтерфейсу: Розробка клієнтської частини вебсервісу "CodeAnalyzer" з використанням HTML, CSS, JavaScript та Bootstrap для забезпечення зручної взаємодії користувача з усіма функціями системи.

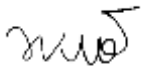
8. Розробка серверної логіки та системи зберігання даних: Реалізація на PHP обробки запитів, управління сесіями, системи реєстрації та автентифікації, а також механізму збереження даних користувачів та історії аналізів у файловій системі.

9. Проведення експериментального дослідження: Розробка методики тестування, підготовка тестових даних та проведення експериментів для оцінки якості аналізу, зручності інтерфейсу та стабільності роботи розробленого вебсервісу.

10. Аналіз результатів та формулювання висновків: Систематизація результатів експериментів, їх аналіз, порівняльна оцінка розробленого рішення з існуючими аналогами та формулювання висновків за результатами кваліфікаційної роботи.
4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Визначення теми, мети та завдань кваліфікаційної роботи	01.03.2025
2	Попередній огляд літератури за темою дослідження.	02.03.2025
3	Складання детального плану роботи та узгодження його з керівником.	03.03.2025
4	Визначення основних інструментів та технологій, що будуть використовуватися.	03.03.2025
5	Глибокий аналіз існуючих методів перевірки та оптимізації програмного коду.	05.03.2025
6	Огляд сучасних моделей штучного інтелекту, що застосовуються для роботи з програмним кодом.	06.04.2025
7	Детальне вивчення можливостей та обмежень OpenAI API для задач аналізу коду.	14.04.2025
8	Формулювання висновків за результатами теоретичного огляду.	22.04.2025
9	Проєктування архітектури веб-додатку «CodeAnalyzer»	26.04.2025
10	Розробка веб-додатку «CodeAnalyzer»	05.05.2025
11	Розробка методики проведення експериментального дослідження.	10.05.2025
12	Збір та систематизація отриманих результатів.	11.05.2025
13	Написання текстової частини роботи	12.05.2025
14	Формування переліку використаних джерел	14.05.2025
15	Підготовка додатків	15.05.2025
16	Оформлення пояснювальної записки згідно з вимогами	20.05.2025
17	Перевірка роботи на відповідність вимогам, вичитка та коректура.	26.05.2025

5. Дата видачі завдання 16.04.2025



Студент Жигайло І.В.

ініціали, прізвище підпис

Керівник роботи Трусов М.А.

ініціали, прізвище підпис

АНОТАЦІЯ

Кваліфікаційна робота бакалавра на тему: «Аналіз коду програмного забезпечення з використанням методів та засобів штучного інтелекту» складається зі вступу, трьох розділів, висновків, переліку використаних джерел та додатків. Загальний обсяг роботи становить 147 сторінки, з них основного тексту – (орієнтовно) 101 сторінка. Робота містить 5 малюнків, не містить таблиць у основному тексті, перелік використаних джерел включає 30 найменувань, а також 3 додатки (Лістинг основних частин коду вебсервісу, Приклади запитів та відповідей OpenAI API, Інструкція користувача). Метою роботи є розробка та дослідження вебсервісу "CodeAnalyzer", що дозволяє здійснювати автоматизований аналіз програмного коду та порівняння його версій з використанням штучного інтелекту на основі інтеграції OpenAI API та мови програмування PHP. У роботі використовувалися методи обробки програмного коду, технології штучного інтелекту (велика мовна модель GPT-4o від OpenAI), засоби веб-розробки (PHP для серверної частини, HTML, CSS, JavaScript, Bootstrap для клієнтської частини), HTTP-запити (бібліотека cURL в PHP для взаємодії з API). Результатом роботи є функціональний веб-додаток "CodeAnalyzer", який дозволяє користувачам завантажувати або вставляти програмний код, отримувати його детальний аналіз від OpenAI API (включаючи виявлення потенційних помилок, вразливостей, проблем продуктивності, оцінку якості та рекомендації щодо покращення), порівнювати дві версії коду з візуалізацією відмінностей та аналізом змін, а також (для зареєстрованих користувачів) зберігати історію аналізів. Отримані результати демонструють практичну реалізацію вебсервісу, що використовує ШІ для покращення якості коду, та можуть бути використані програмістами для прискорення процесу розробки, полегшення виявлення помилок та навчання найкращим практикам програмування. Ключові слова: ШТУЧНИЙ ІНТЕЛЕКТ, PHP, OPENAI API, АНАЛІЗ ПРОГРАМНОГО КОДУ, GPT, ВЕБСЕРВІС, ПОРІВНЯННЯ КОДУ, ЯКІСТЬ КОДУ

ABSTRACT

The Bachelor's qualification thesis on the topic: "Analysis of Software Code Using Artificial Intelligence Methods and Tools" consists of an introduction, three chapters, conclusions, a list of references, and appendices. The total volume of the work is 144 pages, of which the main text is (approximately) 101 pages. The work contains 5 figures, no tables in the main text, the list of references includes 30 titles, and there are 3 appendices (Listing of the main parts of the web service code, Examples of OpenAI API requests and responses, User manual).

The aim of the work is the development and research of the "CodeAnalyzer" web service, which allows for automated analysis of program code and comparison of its versions using artificial intelligence based on the integration of the OpenAI API and the PHP programming language. The work utilized methods of program code processing, artificial intelligence technologies (the large language model GPT-4o from OpenAI), web development tools (PHP for the server-side, HTML, CSS, JavaScript, Bootstrap for the client-side), and HTTP requests (cURL library in PHP for API interaction).

The result of the work is a functional web application "CodeAnalyzer," which allows users to upload or paste program code, receive its detailed analysis from the OpenAI API (including the detection of potential errors, vulnerabilities, performance issues, quality assessment, and improvement recommendations), compare two versions of code with visualization of differences and analysis of changes, and also (for registered users) save the history of analyses. The obtained results demonstrate the practical implementation of a web service that uses AI to improve code quality and can be used by programmers to accelerate the development process, facilitate error detection, and learn best programming practices.

Keywords: ARTIFICIAL INTELLIGENCE, PHP, OPENAI API, SOFTWARE CODE ANALYSIS, GPT, WEB SERVICE, CODE COMPARISON, CODE QUALIT

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface (інтерфейс програмування застосунків)

GPT – Generative Pretrained Transformer (генеративна попередньо навчена модель-трансформер)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

LLM – Large Language Model (велика мовна модель)

NLP – Natural Language Processing (обробка природної мови)

PHP – Hypertext Preprocessor (мова програмування)

ШІ – штучний інтелект

OpenAI – компанія-розробник штучного інтелекту та API до моделей на основі ШІ

Вебсервіс – програмний засіб, доступ до якого забезпечується через HTTP

ВСТУП

Розвиток інформаційних технологій спричинив значне зростання обсягів і складності програмного забезпечення, що робить процеси його розробки, аналізу та підтримки дедалі більш трудомісткими. Сучасні підходи до аналізу програмного коду включають статичні та динамічні методи, які, незважаючи на поширеність і ефективність, мають низку недоліків: високу складність ручної роботи, значні часові витрати, можливість людських помилок. Таким чином, постає необхідність у розробці нових, ефективніших підходів, здатних значно автоматизувати процес аналізу програмного забезпечення.

Одним із найбільш перспективних напрямів вирішення цієї проблеми є застосування технологій штучного інтелекту (ШІ), зокрема великих мовних моделей (LLM), таких як GPT (Generative Pretrained Transformer), створених компанією OpenAI. Дані моделі вже показали високу ефективність у задачах обробки природної мови, що робить можливим їхнє застосування для аналізу програмного коду з метою виявлення помилок, оптимізації структури та покращення загальної якості програмного продукту.

На сьогодні існує кілька інструментів і систем, які використовують технології ШІ для аналізу програмного коду. Серед них варто згадати такі системи, як DeepCode, Sourcery та Codota, що базуються на різних алгоритмах машинного навчання і пропонують широкий спектр функцій – від автоматичного виправлення помилок до рекомендацій з написання ефективнішого коду. Проте, незважаючи на наявні рішення, подальше вдосконалення таких інструментів є актуальним завданням, оскільки вони часто не враховують специфіку програмування для конкретних мов чи технологій, зокрема PHP.

Об'єктом дослідження в роботі виступає програмний код як складна інформаційна система, що потребує аналізу та оптимізації для забезпечення якісної роботи програмних продуктів. Вибір саме програмного коду як об'єкта обумовлений його фундаментальним значенням у процесі розробки програмного забезпечення та великим впливом на кінцевий продукт.

Предметом дослідження є методи автоматизованого аналізу програмного коду, засновані на застосуванні технологій штучного інтелекту, зокрема інтеграції мовних моделей GPT від OpenAI. Розгляд предмету дослідження передбачає аналіз переваг і обмежень застосування таких моделей, а також їх ефективність для вирішення конкретних завдань програмної інженерії.

Метою роботи є створення вебсервісу на основі мови програмування PHP з інтеграцією API OpenAI для автоматизованого аналізу та оптимізації програмного коду. Вибір саме цієї мети визначається необхідністю забезпечення доступності, простоти використання та ефективності такого рішення у реальних умовах розробки програмного забезпечення.

Для досягнення поставленої мети необхідно вирішити такі завдання: провести детальний аналіз існуючих методів і засобів аналізу програмного коду, дослідити потенціал застосування API OpenAI у задачах програмної інженерії, спроектувати та реалізувати вебсервіс з використанням PHP і API OpenAI, а також здійснити експериментальну перевірку роботи розробленого сервісу для визначення його практичної ефективності.

Актуальність роботи визначається потребою у нових, ефективних та автоматизованих інструментах аналізу програмного коду, що можуть суттєво прискорити процеси розробки та покращити якість програмних продуктів. Новизною роботи є застосування сучасних моделей штучного інтелекту (LLM) у практичних завданнях програмної інженерії для створення функціонального інструменту автоматизованого аналізу програмного забезпечення.

Отримані результати дозволять програмістам і розробникам програмного забезпечення значно скоротити витрати часу на аналіз коду, підвищити продуктивність та загальну якість створюваних продуктів, що є важливим кроком до вдосконалення сучасних процесів розробки. Крім того, впровадження такого вебсервісу може сприяти розширенню можливостей для співпраці в розподілених командах розробників, що є особливо актуальним у контексті сучасних реалій дистанційної роботи.

Таким чином, проведення цього дослідження має значну теоретичну та практичну цінність і сприяє розвитку сфери програмної інженерії через інтеграцію сучасних технологій штучного інтелекту

1 ТЕОРЕТИЧНІ ОСНОВИ АНАЛІЗУ ПРОГРАМНОГО КОДУ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

1.1. Аналіз існуючих методів перевірки та оптимізації коду

Розробка програмного забезпечення незмінно постає як складний, багатоетапний процес, де забезпечення якості, надійності, безпеки та ефективності кінцевого продукту відіграє абсолютно критичну роль. Якість програмного забезпечення може бути визначена через різні характеристики, описані, наприклад, у стандарті ISO/IEC 25010:2011, що включає функціональну придатність, продуктивність, сумісність, зручність використання, надійність, безпеку, супроводжуваність та переносимість. Помилки, вразливості або прояви неефективності, що виникають на етапі написання коду, мають потенціал перерости у значні фінансові та репутаційні втрати на пізніших стадіях життєвого циклу програмного продукту. Саме тому глибоке розуміння та застосування методів перевірки й оптимізації коду стає невід'ємною частиною сучасної інженерії програмного забезпечення. Аналізуючи історію розвитку цієї галузі, можна констатувати, що сформувався цілий набір традиційних підходів, які й досі, незважаючи на активний розвиток нових технологій, зокрема штучного інтелекту, продовжують широко використовуватися та слугують фундаментальною основою для багатьох сучасних інструментів. У цьому підрозділі маємо на меті розглянути ключові з цих методів, які умовно можна поділити на статичний аналіз, динамічний аналіз та ручний аудит коду, кожен з яких характеризується власними сильними сторонами та специфічними сферами застосування. Окремої уваги заслуговує також оптимізація коду, адже вона безпосередньо спрямована на підвищення його продуктивності та ефективності використання ними обчислювальних ресурсів. Розуміння цих традиційних підходів та їхніх іманентних обмежень є важливим для обґрунтування необхідності розробки більш досконалих інструментів, здатних доповнити або частково автоматизувати найскладніші аспекти аналізу, що і стало однією з передумов для створення мого вебсервісу "CodeAnalyzer".

Статичний аналіз програмного коду

Статичний аналіз є процесом дослідження програмного коду без його фактичного виконання, що дозволяє виявляти потенційні дефекти, вразливості та відхилення від стандартів на ранніх етапах розробки. Аналізу може піддаватися як вихідний код, так і його проміжні представлення чи скомпільовані файли. Це сприяє зменшенню вартості виправлення помилок.

Однією з найпростіших форм є лінтинг, що перевіряє код на відповідність стилістичним правилам, конвенціям іменування, форматуванню та виявляє деякі поширені помилки програмування. Хоча лінтери покращують читабельність, вони зазвичай не виявляють складних логічних помилок, що залишає відкритим питання ефективного виявлення більш глибоких дефектів.

Більш просунутий аналіз на основі Абстрактного Синтаксичного Дерева (AST) передбачає парсинг коду та побудову його ієрархічного структурного представлення. Це дозволяє інструментам перевіряти коректність типів даних, виявляти неправильне використання API та аналізувати структурні аномалії. Багато SAST-інструментів та компілятори використовують AST-аналіз для глибшої перевірки. Важливість таких підходів підкреслюється в контексті створення екосистем для аналізу програм, як, наприклад, описано в роботі про Tricorder.

Аналіз потоку управління (Control Flow Analysis - CFA) моделює можливі шляхи виконання програми за допомогою графа потоку управління (CFG), дозволяючи виявляти недосяжний код або проблеми з обробкою помилок. Аналіз потоку даних (Data Flow Analysis - DFA) фокусується на життєвому циклі даних у програмі, виявляючи проблеми як використання неініціалізованих змінних чи витоки ресурсів. Специфічним типом DFA є аналіз "забруднених" даних (Taint Analysis), орієнтований на виявлення вразливостей безпеки шляхом відстеження поширення даних з ненадійних джерел.

Символьне виконання (Symbolic Execution) використовує символьні змінні для дослідження багатьох шляхів виконання, генерації тестів та формальної верифікації, хоча стикається з проблемою "вибуху шляхів". Перевірка моделей (Model Checking) – це формальний метод верифікації моделі системи на відповідність заданим властивостям, корисний для перевірки протоколів та властивостей безпеки, але обмежений проблемою "вибуху станів".

Загалом, статичний аналіз дозволяє виявляти помилки на ранніх стадіях та автоматизувати процес без запуску програми. Однак він схильний до хибнопозитивних/негативних спрацювань та обмежено розуміє динаміку. Ці обмеження вказують на потребу в доповненні іншими методами та на потенціал інтелектуальних систем.

Динамічний аналіз програмного коду

Динамічний аналіз передбачає виконання програми для виявлення помилок часу виконання, оцінки продуктивності та перевірки функціональності.

Найпоширенішою формою є тестування програмного забезпечення, що включає модульне (ізольована перевірка частин), інтеграційне (взаємодія модулей), системне (повністю інтегрована система), приймальне (з точки зору замовника) та регресійне (перевірка після змін) тестування.

Профілювання збирає дані про продуктивність: CPU Profiling находит "гарячі точки", Memory Profiling відстежує використання пам'яті, I/O Profiling аналізує операції вводу-виводу. Аналіз покриття коду вимірює, яка частина коду виконана тестами, а фазинг-тестування подає програмі невалідні дані для провокації збоїв та виявлення уразливостей.

Перевагами динамічного аналізу є виявлення помилок часу виконання. Однак його ефективність залежить від тестових даних, він може бути трудомістким, а моніторинг – впливати на поведінку програми.

Ручний аудит коду (Рев'ю коду)

Ручний аудит – це ретельне вивчення коду розробниками-рецензентами для виявлення помилок, недоліків дизайну та покращення якості коду, використовуючи людський інтелект та досвід. Процес включає планування, індивідуальне вивчення, обговорення, доопрацювання та верифікацію. Існують різні типи: від неформального рев'ю до строгої інспекції коду та поширеного сьогодні рев'ю через системи контролю версій (Pull/Merge Request Reviews). Дослідження показують, що навіть мова коментарів під час рев'ю має значення. Перевіряються коректність, дизайн, читабельність, безпека, продуктивність, тестованість та стандарти кодування.

Перевагами є виявлення складних проблем та обмін знаннями. Обмеження: трудомісткість, суб'єктивність, складність масштабування, ризик пропуску помилок.

Методи оптимізації програмного коду

Оптимізація коду спрямована на підвищення продуктивності та ефективності використання ресурсів. Оптимізація на рівні компілятора відбувається автоматично (усунення загальних підвиразів, розгортання циклів). Ручна оптимізація розробником важлива для "гарячих точок" і включає вибір ефективних алгоритмів та структур даних, оптимізацію циклів, роботи з пам'яттю та операцій вводу-виводу. Критично важливо проводити профілювання перед оптимізацією та дотримуватися балансу між оптимізацією, читабельністю та супровідністю.

На завершення цього огляду традиційних методів, варто підкреслити, що їхня найбільша ефективність досягається при комплексному застосуванні. Вони створюють фундамент для більш просунутих підходів, включаючи ті, що використовують штучний інтелект, які будуть розглянуті далі. Розуміння обмежень цих традиційних підходів обґрунтовує актуальність розробки інтелектуальних інструментів аналізу.

1.2. Огляд сучасних моделей ШІ для роботи з програмним кодом

Перехід від традиційних методів аналізу коду, розглянутих раніше, до використання штучного інтелекту (ШІ) знаменує якісно новий етап в інженерії програмного забезпечення [1]. Сучасні моделі ШІ, особливо великі мовні моделі (LLM) та спеціалізовані архітектури, навчені на величезних масивах програмного коду, демонструють значні здібності до його розуміння, генерації та аналізу [2]. Це відкриває широкі перспективи для автоматизації складних завдань, підвищення продуктивності розробників та покращення якості програмних продуктів. Цей підрозділ присвячений огляду ключових типів таких моделей, їхніх архітектурних особливостей та можливостей у контексті роботи з програмним кодом.

Великі мовні моделі (LLM) загального призначення з розширеними можливостями кодування

Великі мовні моделі (LLM) справили значний вплив на програмування, оскільки вони здатні «розуміти» синтаксис, семантику та прагматику мов програмування. В основі більшості LLM лежить архітектура Transformer [3] з її механізмом уваги (attention), що дозволяє вловлювати довгострокові залежності в коді, що було проблемою для попередніх рекурентних архітектур [4]. LLM проходять попереднє навчання на великих текстових та кодових корпусах, а потім можуть бути тонко налаштовані для конкретних завдань [5; 6].

Серед провідних сімейств LLM варто виділити розробки OpenAI, зокрема сімейство GPT (включаючи GPT-3.5, GPT-4, GPT-4o та спеціалізовану на коді модель Codex), які демонструють високі здібності до генерації коду, його пояснення, налагодження та перекладу [7; 8], причому GPT-4/4o вирізняються більшим контекстним вікном та кращим розумінням складних інструкцій; варто відзначити, що ранні експерименти з GPT-4 навіть показали проблиски загального штучного інтелекту [9]. Конкуренцію їм складають моделі Google, такі як PaLM 2, що показує хороші результати в задачах кодування, та мультимодальна Gemini, яка конкурує з GPT-4 за продуктивністю. Окремої уваги заслуговують відкриті LLM, наприклад, Llama та Code Llama від Meta AI, а також моделі від Mistral AI, що надають спільноті можливість вільного експериментування, тонкого налаштування та локального розгортання, що є важливим для контролю даних та оптимізації витрат[10].

Ключові можливості LLM для роботи з кодом є досить широкими: вони охоплюють генерацію коду за текстовим описом [11], глибоке розуміння та пояснення наявного коду, його переклад між різними мовами програмування, а також базове виявлення помилок та потенційних вразливостей, хоча в останньому аспекті вони не можуть повністю замінити спеціалізовані SAST/DAST інструменти. Крім того, LLM здатні пропонувати варіанти для рефакторингу коду, автоматично генерувати документацію та коментарі, а також надавати інтерактивну допомогу розробникам у форматі «питання-відповідь». Саме такий широкий спектр функціональних можливостей, включаючи роботу з семантичними представленнями коду [12], робить великі мовні моделі надзвичайно приваблі-

вими для створення багатофункціональних інструментів аналізу програмного забезпечення. Існують також дослідження, присвячені навчанню моделей на основі ідентифікаторів у коді, як, наприклад, у Codet5 [13], або використанню попередньо навчених моделей для мов програмування та природних мов, таких як CodeBERT [14]. Широкі дослідження в галузі машинного навчання для "великого коду" та його "природності" також підтверджують потенціал цих технологій [15].

Спеціалізовані моделі ШІ для аналізу програмного коду

Поряд з універсальними LLM, активно розвиваються і вузькоспеціалізовані моделі ШІ, які часто пропонують вищу точність або менші обчислювальні вимоги для конкретних завдань аналізу коду. До таких моделей належать, по-перше, рішення для статичного аналізу на основі ШІ (AI-enhanced Static Analysis), що використовуються для зменшення хибнопозитивних спрацьовувань традиційних SAST-інструментів та пріоритезації знахідок. По-друге, це моделі на основі графових нейронних мереж (GNN), які ефективно аналізують код, представлений у вигляді графів (AST, CFG, DFG, CPG), для виявлення вразливостей, клонів коду та розуміння його семантики [16]. По-третє, існують моделі, спеціально орієнтовані на виявлення вразливостей безпеки, які поєднують глибоке навчання з традиційними методами та навчаються на датасетах вразливого коду. Четвертим напрямком є моделі для автоматичної генерації тестів, що використовують ШІ для створення значущих тестових випадків, які покращують покриття коду. Нарешті, амбітною галуззю є розробка моделей для автоматичного ремонту програм (APR), які намагаються не лише знаходити, а й автоматично виправляти помилки, використовуючи шаблонний, семантичний або генеративний підходи, в тому числі з використанням LLM, хоча ця сфера все ще стикається зі значними викликами.

Порівняльний аналіз та синергія різних підходів

LLM загального призначення та спеціалізовані моделі ШІ мають свої унікальні переваги та недоліки[17]. LLM вирізняються універсальністю, гнучкістю та легкістю адаптації до нових завдань, проте можуть поступатися в точ-

ності для вузькоспеціалізованих проблем, бути більш витратними в обчислювальному плані та менш інтерпретованими. Спеціалізовані моделі, навпаки, зазвичай точніші у своїй конкретній ніші, можуть бути ефективнішими з точки зору обчислень та пропонувати кращу інтерпретованість результатів, але при цьому вони менш гнучкі та адаптивні.

Найбільш перспективним напрямком, на мою думку, є розробка гібридних підходів, які б поєднували сильні сторони обох типів моделей. У таких системах LLM могли б використовуватися для забезпечення зручної взаємодії з користувачем, для попереднього аналізу коду та формулювання гіпотез, тоді як спеціалізовані моделі могли б застосовуватися для глибшого та точнішого дослідження конкретних аспектів коду. Такий синергетичний підхід дозволив би об'єднати широту охоплення LLM з точністю спеціалізованих рішень.

Переваги та виклики використання сучасних ШІ-моделей

Застосування ШІ в аналізі програмного коду, як показують численні дослідження [18], обіцяє значні переваги: підвищення продуктивності розробників, покращення якості та надійності коду, прискорення циклу розробки, покращення безпеки, демократизацію доступу до просунутих інструментів та підтримку навчання.

Однак існують і серйозні виклики та обмеження. До них належать проблеми точності та «галюцинацій» LLM, що вимагає ретельної перевірки людиною [19]. Суттєвим обмеженням є розуміння глобального контексту великих проєктів. Якість роботи ШІ-моделей залежить від якості навчальних даних. Обчислювальні ресурси та вартість навчання й використання великих моделей залишаються значними. Питання безпеки та конфіденційності коду при передачі стороннім сервісам викликають занепокоєння. Крім того, існує комплекс етичних аспектів (авторське право, відповідальність, упередженість). Нарешті, необхідна постійна потреба в оновленні знань моделей через еволюцію технологій [20].

Роль та місце OpenAI API серед розглянутих підходів

OpenAI API, що надає доступ до моделей сімейства GPT, таких як GPT-4 [16], посідає важливе місце в екосистемі ШІ-інструментів для роботи з кодом.

Його можна віднести до категорії потужних LLM загального призначення з розширеними можливостями кодування. Сильні сторони OpenAI API, що стали визначальними при виборі цього інструменту для мого дипломного проєкту «CodeAnalyzer», включають його універсальність, високу якість генерації тексту та коду, відносну простоту інтеграції та доступ до передових технологій ШІ без необхідності самостійного навчання моделей.

Однак важливо усвідомлювати й обмеження OpenAI API: моделі GPT не є спеціалізованими інструментами для SAST чи формальної верифікації; обмежене контекстне вікно може створювати труднощі при аналізі великих проєктів; результати завжди потребують перевірки розробником.

Мій проєкт «CodeAnalyzer», використовуючи OpenAI API, фокусується на використанні сильних сторін LLM: наданні користувачам можливості отримувати пояснення коду, виявляти поширені помилки та отримувати пропозиції щодо покращення. Це позиціонує мій вебсервіс як інтелектуального асистента, що доповнює, а не замінює експертизу людини. Глибоке розуміння можливостей та обмежень OpenAI API є ключовим для реалістичної оцінки потенціалу мого проєкту.

1.3. Використання OpenAI API у задачах аналізу програмного коду

Поява програмних інтерфейсів доступу (API) до передових великих мовних моделей, таких як розроблені компанією OpenAI (включаючи GPT-3.5, GPT-4 та GPT-4o) [16], справила значний вплив на різні галузі, і розробка програмного забезпечення не стала винятком. Можливість інтегрувати потужний штучний інтелект безпосередньо в інструменти розробки та аналізу коду відкрила нову еру взаємодії людини з кодом. У цьому підрозділі розглядається потенціал використання OpenAI API для вирішення конкретних завдань аналізу програмного коду, аналізуються його сильні сторони, обговорюються неминучі виклики та обмеження, а також визначаються практичні аспекти його застосування, зокрема, в контексті мого дипломного проєкту «CodeAnalyzer».

Ключові можливості OpenAI API для аналізу коду

OpenAI API надає інтерфейс для взаємодії з попередньо навченими моделями, що володіють глибокими знаннями синтаксису, семантики та поширених патернів у багатьох мовах програмування. Ці знання дозволяють реалізувати широкий спектр функцій аналізу. Однією з фундаментальних можливостей є розуміння та пояснення коду (Code Comprehension and Explanation). Моделі GPT здатні аналізувати наданий фрагмент коду та генерувати його опис природною мовою, пояснювати логіку роботи, призначення конструкцій і навіть відповідати на контекстно-залежні питання про код. Це особливо корисно для вивчення нового або складного успадкованого коду. Автоматизоване документування та коментування – ще одна важлива функція. Моделі можуть генерувати рядкові або блокові коментарі, створювати структуровані docstring-и для функцій та класів (наприклад, у стилі Javadoc або Python-докстрингів), що може слугувати основою для автоматичної генерації документації API [21]. У контексті мого проєкту «CodeAnalyzer» особливо актуальним є виявлення помилок, вразливостей та «запахів коду». Моделі GPT можуть знаходити синтаксичні та деякі логічні помилки, ідентифікувати відомі «запахи коду» (ознаки потенційних проблем у дизайні) і навіть проводити базовий аналіз безпеки, розпізнаючи патерни, пов'язані з поширеними вразливостями (наприклад, використання небезпечних функцій, відсутність валідації вхідних даних) [22]. Однак такий аналіз не замінює спеціалізовані інструменти SAST/DAST. Моделі також можуть пропонувати варіанти обробки винятків. OpenAI API може сприяти рефакторингу та покращенню якості коду, пропонуючи варіанти оптимізації, покращення читабельності та стилю, а також спрощення складних умовних конструкцій. Здатність до перекладу коду між мовами програмування, хоча й часто потребує ручного доопрацювання, може бути корисною для міграції проєктів або адаптації алгоритмів. Хоча це більше стосується генерації, можливість написання коду за описом та інтелектуальне автодоповнення (Code Completion) [23] опосередковано впливають на аналіз, оскільки модель, яка вміє писати код, краще його «розуміє». Нарешті, OpenAI API має значний потенціал для допомоги в навчанні програмуванню, виступаючи в ролі інтерактивного навчального інструменту, генеруючи приклади та завдання.

Переваги використання OpenAI API для аналізу коду

Інтеграція OpenAI API в процеси аналізу коду надає низку суттєвих переваг. По-перше, це доступ до передових моделей без значних інвестицій у їх розробку та навчання, що демократизує використання ШІ-технологій. По-друге, універсальність та гнучкість однієї моделі, здатної виконувати широкий спектр завдань, вигідно відрізняє її від вузькоспеціалізованих інструментів. По-третє, розуміння природної мови спрощує взаємодію з аналізатором, роблячи його доступнішим для розробників різного рівня. По-четверте, контекстне розуміння сучасних моделей GPT, що володіють великими контекстними вікнами, призводить до більш релевантних результатів. По-п'яте, для багатьох завдань швидкість отримання результатів від API може бути значно вищою, ніж при ручному аналізі. Нарешті, потенціал для постійного вдосконалення моделей OpenAI означає, що інтегровані програми автоматично отримують доступ до покращених можливостей.

Виклики та обмеження при використанні OpenAI API

Незважаючи на вражаючий потенціал, використання OpenAI API для аналізу коду пов'язане з низкою викликів та обмежень. Ключовою проблемою є точність та «галюцинації»: LLM можуть генерувати некоректну інформацію або помилкові виправлення, тому результати завжди потребують ретельної перевірки людиною [24]. Обмеження контекстного вікна можуть ускладнювати аналіз дуже великих файлів або цілих проєктів без спеціальних стратегій. Моделям також властиве обмежене розуміння специфічної бізнес-логіки та неявних знань конкретного проєкту. Питання безпеки та конфіденційності даних при передачі коду на сторонні сервери викликають обґрунтовані побоювання. Залежність від провайдера (Vendor Lock-in) створює ризики, пов'язані зі змінами в політиках, тарифах або доступності API. Вартість використання API, що розраховується на основі токенів, може бути суттєвою для інтенсивних програм. Можливі затримки (Latency) при взаємодії з API можуть бути критичними для деяких інтерактивних систем. Якість результатів сильно залежить від ефективного

промпт-інжинірингу. Нарешті, існують складні етичні міркування, що стосуються авторського права, відповідальності за помилки ШІ та потенційної упередженості моделей.

Практичні аспекти інтеграції OpenAI API в інструменти аналізу коду

Інтеграція OpenAI API в мій веб-застосунок «CodeAnalyzer» вимагала вирішення кількох ключових технічних завдань. Це, перш за все, безпечна аутентифікація та авторизація з використанням API-ключів. Найважливішим аспектом стало формування ефективних запитів (промптів), включаючи системний промпт (що визначає роль ШІ), користувацький промпт (що містить код та інструкції) і, можливо, використання прикладів (few-shot prompting). Далі йшла обробка відповідей API, зазвичай у форматі JSON, їх парсинг та представлення користувачеві. Для великих фрагментів коду потрібне було б управління контекстним вікном (наприклад, шляхом розбиття коду). Також була реалізована логіка для обробки помилок та обмежень API та передбачено управління витратами шляхом моніторингу використання токенів. У моєму проєкті основний акцент було зроблено на створення інтуїтивного інтерфейсу та розробку ефективних промптів для різних типів аналізу.

Висновки щодо потенціалу OpenAI API

OpenAI API, безперечно, є потужним інструментом з величезним потенціалом для трансформації підходів до аналізу програмного коду. Його здатність розуміти природну мову та код, генерувати описи, виявляти певні типи помилок та пропонувати покращення робить його цінним помічником для розробників. Водночас, важливо підходити до його використання з реалістичними очікуваннями, усвідомлюючи існуючі обмеження в точності, розумінні глибокого контексту та безпеки. Найбільш ефективне застосування OpenAI API в аналізі коду, ймовірно, полягає не в повній заміні існуючих інструментів чи людської експертизи, а в їх доповненні та синергії. LLM можуть взяти на себе рутинні завдання, надати «другу думку» та прискорити процес навчання. Однак кінцеве рішення та відповідальність за якість і безпеку коду завжди залишаються за людиною. Мій дипломний проєкт «CodeAnalyzer» є спробою дослідити та продемонструвати частину цього багатогранного потенціалу, створивши практичний інструмент,

що використовує переваги OpenAI API для покращення процесу розробки програмного забезпечення.

2 РОЗРОБКА ВЕБСЕРВІСУ НА PHP ДЛЯ АНАЛІЗУ КОДУ З ІНТЕГРАЦІЄЮ OPENAI API

Сучасна розробка програмного забезпечення характеризується невпинним зростанням складності проєктів, скороченням циклів випуску продуктів та постійно зростаючими вимогами до якості, надійності та безпеки коду [25]. У цьому контексті, ручний аналіз коду, хоч і залишається важливим етапом контролю якості, часто виявляється недостатньо ефективним. Він є трудомістким процесом, схильним до суб'єктивності та людських помилок, а його глибина обмежена досвідом та часом окремих розробників. Впровадження автоматизованих інструментів для аналізу коду, особливо тих, що використовують передові можливості штучного інтелекту (ШІ), стає не просто актуальним, а необхідним для сучасних команд розробників [8]. Такі інструменти здатні значно підвищити ефективність розробки, допомагаючи на ранніх етапах виявляти потенційні помилки, приховані вразливості безпеки, вузькі місця у продуктивності, а також надавати цінні рекомендації щодо покращення стилю, архітектури та загальної підтримуваності коду.

Цей розділ присвячений детальному опису процесу розробки вебсервісу "CodeAnalyzer". Даний сервіс був спроєктований мною з використанням мови програмування PHP для реалізації серверної логіки та тісно інтегрується з OpenAI API для здійснення глибокого інтелектуального аналізу програмного коду. "CodeAnalyzer" має на меті надати користувачам інтуїтивно зрозумілу вебплатформу для завантаження або безпосереднього введення фрагментів коду, його всебічного аналізу за допомогою потужностей ШІ, порівняння різних версій коду для відстеження змін та їх впливу, а також отримання розгорнутих, структурованих звітів з конкретними рекомендаціями та оцінками.

1.1. Постановка задачі та вибір технологій реалізації

Виходячи з актуальності проблеми автоматизації аналізу коду та потенціалу сучасних ШІ-технологій, описаних у попередньому розділі, основною задачею

моєї кваліфікаційної роботи стала розробка функціонального вебсервісу "CodeAnalyzer". Цей сервіс мав надати користувачам зручний інструмент для інтелектуального аналізу програмного коду, виявлення потенційних проблем та отримання рекомендацій щодо його покращення, використовуючи можливості OpenAI API. Для успішної реалізації цієї задачі та створення надійного і функціонального вебсервісу, мною було ретельно обрано наступний стек технологій.

Для серверної частини (Backend) мого вебсервісу було обрано мову програмування PHP. PHP (Hypertext Preprocessor) залишається однією з найпопулярніших мов сценаріїв загального призначення з відкритим вихідним кодом, яка особливо добре підходить для веб-розробки [26]. Її ключові переваги для даного проєкту, на мою думку, включають широке розповсюдження та наявність великої активної спільноти, що гарантує доступ до значної кількості навчальних матеріалів, готових бібліотек та швидке знаходження рішень для типових проблем розробки. Важливою є також простота розгортання та налаштування. PHP надає багатий набір вбудованих функцій для роботи з веб-протоколами, зокрема зручні інструменти для обробки HTTP-запитів та HTML-форм, що було активно використано у файлах htdocs/upload.php та htdocs/compare_action.php. Вбудовані механізми управління сесіями дозволили підтримувати стан користувача. Також PHP має потужні засоби для роботи з файловою системою, що виявилося критичним для зберігання даних користувачів та історії аналізів у каталогах htdocs/data/ та htdocs/uploads/. Ключовою для взаємодії із зовнішнім API стала нативна підтримка розширення cURL у PHP [26], яке активно використовується у модулі htdocs/utils/php_code_analyzer.php для виконання HTTP-запитів до OpenAI API. Структурування проєкту було реалізовано через центральний контролер htdocs/index.php, який підключає відповідні сторінки та модулі.

Для забезпечення основного інтелектуального аналізу програмного коду було обрано інтеграцію з OpenAI API, зокрема з моделлю gpt-4o [16]. Це забезпечило високу якість та глибину аналізу, підтримку широкого спектру мов програмування, гнучкість формування запитів (промптинг) та зручний формат відповіді у JSON. Системний промпт "You are an expert code analyzer..." та низьке

значення temperature (0.2) сприяли отриманню сфокусованих відповідей, як це детально описано у файлі htdocs/utils/php_code_analyzer.php.

Для створення клієнтської частини (Frontend) вебсервісу "CodeAnalyzer" було використано стандартний набір веб-технологій: HTML для структурування контенту, CSS для візуального оформлення (включаючи власний файл htdocs/static/css/styles.css) та JavaScript для інтерактивності на стороні клієнта, валідації форм та динамічного оновлення. Функціональність JavaScript була розподілена по декількох файлах для кращої організації (htdocs/static/js/main.js, htdocs/static/js/auth.js та інші).

Для прискорення розробки користувацького інтерфейсу було обрано CSS-фреймворк Bootstrap 5.3.0, що забезпечило адаптивний дизайн та набір готових UI-компонентів. Для покращення читабельності програмного коду, що відображається, було інтегровано бібліотеку Prism.js, яка автоматично підсвічує синтаксис та підтримує нумерацію рядків. Для візуального покращення інтерфейсу використовувалися SVG-іконки Feather Icons [27].

Як основний формат для обміну даними з OpenAI API та для зберігання даних користувачів і історії аналізів у файловій системі, було обрано JSON (JavaScript Object Notation). Інформація про користувачів (htdocs/data/users.json), метадані аналізів (htdocs/data/analyses.json) та детальні результати кожного аналізу (у підкаталогах htdocs/data/analysis/[id]/) зберігаються у цьому форматі. Для цілей даної кваліфікаційної роботи та прототипу використання файлової системи для зберігання JSON-даних вважаємо виправданим через простоту реалізації, хоча для масштабних систем перевага була б надана базам даних.

Організація роботи з файлами та константами у проєкті використовує PHP-константи для визначення ключових шляхів та параметрів, що покращує читабельність та спрощує конфігурацію [28]. Обраний мною стек технологій забезпечив збалансоване поєднання швидкості розробки, доступності інструментів, потужності аналітичних можливостей ШІ та створення зручного користувацького досвіду, дозволивши ефективно вирішити поставлені завдання.

1.2. Архітектура веб-додатку та взаємодія з OpenAI API

Для забезпечення ефективної роботи вебсервісу "CodeAnalyzer" мною була розроблена архітектура, що поєднує класичний підхід до побудови PHP-додатків з інтеграцією зовнішнього сервісу штучного інтелекту. Архітектура орієнтована на модульність, що дозволяє розділити логіку представлення, обробки запитів та взаємодії з API.

Загальна архітектура веб-додатку побудована за принципом багатошаровості у спрощеному вигляді. Проєкт має чітко визначену структуру каталогів: htdocs/ як кореневий каталог, htdocs/index.php як єдиний центральний файл-маршрутизатор, htdocs/pages/ для скриптів окремих сторінок, htdocs/includes/ для допоміжних файлів та загальних функцій, htdocs/utils/ для утилітарних скриптів (включаючи ключовий php_code_analyzer.php для взаємодії з OpenAI), htdocs/static/ для CSS та JavaScript, htdocs/uploads/ для тимчасового зберігання завантажених файлів, та htdocs/data/ для зберігання даних додатку (користувачі, аналізи, статус API).

Модель маршрутизації реалізована через index.php, який аналізує GET-параметр page і підключає відповідний PHP-файл сторінки. Шаблонізація використовує включення загальних частин макету (header.php, footer.php). GET-запити обробляються переважно через index.php, а POST-запити від форм – спеціалізованими скриптами (upload.php, compare_action.php, сторінки авторизації). Управління станом та сесіями активно використовується для автентифікації, передачі тимчасових даних та відображення flash-повідомлень. Як зазначалося, зберігання даних реалізовано через файлову систему у форматі JSON.

Взаємодія з OpenAI API є серцем аналітичних можливостей сервісу і централізована у файлі htdocs/utils/php_code_analyzer.php. Процес включає декілька етапів. Ініціація запиту відбувається, коли користувач завантажує код або запитує порівняння; відповідний PHP-обробник викликає функції php_analyze_code() або php_compare_code(). Ключовим аспектом є формування промпту (запиту до ШІ). Для аналізу коду промпт включає сам код, мову програмування та чіткі інструкції українською мовою щодо аспектів аналізу (помилки, вразливості, продуктивність, стиль, супровід) та очікуваного формату JSON-відповіді. Для

аналізу змін при порівнянні промпт містить оригінальний та новий код, мову, форматований diff та запит на оцінку змін.

Безпосереднє звернення до OpenAI API здійснюється функцією php_call_openai_api() через розширення PHP cURL до ендпоінту https://api.openai.com/v1/chat/completions. API-ключ передається в HTTP-заголовку Authorization. В тілі POST-запиту передається JSON-об'єкт з назвою моделі (gpt-4o), масивом повідомлень (системне та користувацьке) та параметром temperature: 0.2. Обробка відповіді від API включає декодування JSON, вилучення корисного контенту та обробку можливих помилок. У випадку помилок або невалідного API-ключа система використовує функцію php_generate_mock_analysis() для генерації шаблонних результатів, забезпечуючи працездатність сервісу. Після успішної обробки, структуровані дані аналізу передаються на відповідну сторінку для візуалізації. Якщо користувач авторизований, результати зберігаються за допомогою функції add_analysis() з htdocs/includes/functions.php. Для перевірки працездатності з'єднання з OpenAI API передбачені сторінки htdocs/pages/api_status.php та htdocs/pages/test_api.php.

1.3. Реалізація алгоритмів аналізу та інтерфейсу користувача

Успішна робота вебсервісу "CodeAnalyzer" залежить від ефективності алгоритмів аналізу коду, що базуються на взаємодії з OpenAI API, та зручності користувацького інтерфейсу.

Реалізація алгоритмів аналізу централізована у htdocs/utils/php_code_analyzer.php. Функція php_analyze_code() є основною для аналізу окремого фрагмента коду. Вона приймає код та мову програмування, перевіряє API-ключ, виконує тестовий виклик API, формує детальний промпт для моделі gpt-4o із запитом на аналіз за різними критеріями (помилки, вразливості, продуктивність, стиль, супровід) та вимогою повернути відповідь у структурованому JSON-форматі. Функція php_compare_code() аналізує зміни між двома версіями коду. Вона викликає php_generate_diff() для простого рядкового порівняння та php_analyze_changes_with_ai() для формування спеціального промпту до OpenAI щодо оцінки впливу змін, покращень та потенційних проблем. Результат включає diff,

аналіз від ШІ та метрики змін. Функція php_call_openai_api() відповідає за безпосередню взаємодію з OpenAI через cURL, обробку відповіді та логування. Функції php_generate_mock_analysis() та php_analyze_changes_basic() слугують запасним варіантом при недоступності API. Результати аналізу зберігаються у файловій системі для авторизованих користувачів.

Реалізація інтерфейсу користувача виконана з акцентом на простоту, інформативність та візуальну привабливість, використовуючи HTML, CSS (Bootstrap та кастомні стилі htdocs/static/css/styles.css) та JavaScript. Усі сторінки мають єдину структуру завдяки загальним header.php та footer.php. Навігаційна панель містить посилання на ключові розділи та блок автентифікації. Головна сторінка (htdocs/pages/home.php) містить секцію завантаження коду з двома вкладками: "Завантажити файл" (з drag-and-drop зоною) та "Вставити код" (з вибором мови та textarea). Обидві форми надсилають дані на htdocs/upload.php.

Сторінка результатів аналізу (htdocs/pages/analyze.php) відображає назву файлу, зведений блок з оцінкою якості (візуалізованою колом score-circle), кількістю проблем за рівнями серйозності та інформацією про мову. Основний контент розділений на дві колонки: ліва – вихідний код з підсвічуванням синтаксису Prism.js та нумерацією рядків (з кнопками "Переключити номери рядків" та "Копіювати код"); права – список проблем (з описом, рядками, серйозністю, пропозицією виправлення та прикладом коду), детальні рекомендації (в акордеоні) та загальні пропозиції від OpenAI. Клік по проблемі активує її та підсвічує відповідний рядок у коді.

Сторінка порівняння коду (htdocs/pages/compare.php) містить форму для введення початкового та нового коду та вибору мови. Сторінка результатів порівняння (htdocs/pages/comparison_results.php) показує огляд змін (оцінка впливу, зміна якості, метрики), списки покращень, потенційних проблем та пропозицій від ШІ, а також візуальне порівняння коду (Diff View) з підсвічуванням доданих, видалених та змінених рядків.

Сторінки автентифікації (login.php, register.php), управління профілем (profile.php) та історії аналізів (my_analyses.php) реалізовані з використанням стандартних форм Bootstrap та іконок Feather Icons. "Мої аналізи" відображають

список попередніх робіт користувача з можливістю перегляду та видалення, а також фільтрації та сортування. Інтерактивність забезпечується JavaScript, включаючи клієнтську валідацію форм (htdocs/static/js/auth.js) та flash-повідомлення для зворотного зв'язку.

3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РОЗРОБЛЕНОГО ВЕБ-СЕРВІСУ "CodeAnalyzer"

Після завершення етапів теоретичного обґрунтування, детального проєктування та безпосередньої програмної реалізації вебсервісу "CodeAnalyzer", описаних у попередніх розділах даної кваліфікаційної роботи, було перейдено до ключового та найбільш значущого етапу мого дослідження – проведення його всебічного та ретельного експериментального вивчення. Основною метою цього дослідження було не лише формальне підтвердження працездатності всього реалізованого мною функціоналу, але й глибока, об'єктивна оцінка якості аналізу програмного коду, що надається розробленою системою завдяки її інтеграції з програмним інтерфейсом OpenAI API. Не менш важливими завданнями на цьому етапі були тестування зручності та інтуїтивності розробленого мною користувацького інтерфейсу, перевірка стабільності роботи вебсервісу за різних умов експлуатації та, зрештою, виявлення потенційних напрямків для його подальшого вдосконалення, оптимізації та можливого розвитку. Цей розділ кваліфікаційної роботи присвячений детальному опису методики, відповідно до якої було проведено експериментальне дослідження, представленню ключових аспектів розробленого мною вебсервісу "CodeAnalyzer", що включає демонстрацію найважливіших та найбільш показових фрагментів програмного коду, написаних безпосередньо мною, та візуальне представлення користувацького інтерфейсу сервісу через серію скриншотів. Також у цьому розділі буде наведено детальний аналіз конкретних результатів, отриманих мною під час проведення тестування, та їхня порівняльна оцінка з огляду на поставлені завдання.

1.4. Методика проведення експериментального дослідження

Для забезпечення об'єктивності та комплексності оцінки розробленого мною вебсервісу "CodeAnalyzer", мною була розроблена та послідовно застосована методика експериментального дослідження. Ця методика охоплювала

декілька ключових етапів та передбачала тестування різних аспектів функціонування системи.

Насамперед, мною були чітко сформульовані основні цілі експериментального дослідження. По-перше, стояло завдання оцінити функціональну повноту та коректність роботи сервісу. Це передбачало ретельну перевірку працездатності абсолютно всіх функцій, заявлених у проєкті, а саме: коректність процесу завантаження програмного коду користувачем (як через механізм завантаження файлу, так і шляхом прямої вставки текстового коду у відповідну веб-форму), правильність ініціації та виконання процедури аналізу коду з використанням інтегрованого OpenAI API, належне функціонування можливості порівняння двох різних версій програмного коду, а також надійність роботи системи реєстрації нових користувачів та автентифікації вже існуючих, включаючи функції збереження та подальшого перегляду історії виконаних аналізів для авторизованих користувачів.

По-друге, однією з найважливіших цілей була оцінка якості аналізу програмного коду, що надається OpenAI API через розроблений мною вебсервіс. Це включало визначення точності, з якою система, керована моїми спеціально розробленими промптами до OpenAI, ідентифікує різні типи проблем у наданому для аналізу програмному коді. До таких проблем належать логічні та синтаксичні помилки, потенційні вразливості безпеки, проблеми, пов'язані з продуктивністю та ефективністю коду, а також недоліки стилю кодування та відхилення від загальноприйнятих найкращих практик. Окрім точності, оцінювалася повнота проведеного аналізу, тобто наскільки вичерпним був перелік виявлених проблем, та, що особливо важливо, релевантність, корисність і практична застосовність запропонованих системою рекомендацій та варіантів виправлення виявлених недоліків у коді.

Третьою важливою ціллю була оцінка ефективності реалізованої мною функції порівняння версій програмного коду. Тут було перевіряно, наскільки коректно мій сервіс візуалізує відмінності (diff) між двома наданими користувачем версіями коду, та наскільки якісним, змістовним та корисним для розробника є

аналітичний звіт про характер та можливі наслідки внесених змін, що генерується штучним інтелектом на основі аналізу цих відмінностей.

По-четверте, прагнемо провести оцінку зручності та інтуїтивності розробленого мною користувацького інтерфейсу (Usability Testing). Для мене було важливо визначити, наскільки легко, швидко та без зайвих ускладнень користувачі (включаючи мене самого як основного розробника та кількох залучених колег-студентів для отримання незалежної оцінки) можуть взаємодіяти з усіма функціями сервісу, виконувати основні операції, такі як завантаження коду, перегляд результатів аналізу, навігація по сторінках сайту, та наскільки чітко, однозначно й інформативно вони можуть інтерпретувати отримані результати аналізу та рекомендації.

І, нарешті, п'ятою ціллю була оцінка продуктивності та стабільності роботи розробленого вебсервісу. Було проведено вимірювання часу відповіді системи на запити аналізу коду різного обсягу та складності, а також перевіряв стабільність роботи вебсервісу за різних сценаріїв використання. Це включало перевірку коректної обробки можливих помилок (наприклад, при передачі некоректних даних або файлів непідтримуваного формату) та тестування роботи системи з використанням імітаційних даних (mock data) у випадку тимчасової недоступності OpenAI API або використання невалідного ключа доступу до API, щоб переконатися у відмовостійкості сервісу.

Для проведення цих експериментів мною були ретельно підготовлені та структуровані набори тестових даних. Ці набори включали різноманітні фрагменти програмного коду, написані тими мовами програмування, які підтримуються моїм вебсервісом. Зокрема, було зосереджено на PHP (як основній мові розробки самого сервісу та популярній мові для веб-розробки), Python (через його широке застосування та різноманітність стилів кодування), JavaScript (як ключовій мові для фронтенд-розробки) та C++ (як представнику компільованих мов зі строгим контролем типів та управлінням пам'яттю). Тестовий код варіювався за обсягом – від невеликих окремих функцій та коротких скриптів, що містили декілька десятків рядків, до файлів, що налічували кілька сотень рядків коду, щоб оцінити вплив розміру на швидкість та якість аналізу. Також було

приділено значну увагу тому, щоб тестові дані були різного рівня якості: було підібрано як приклади коду з навмисно внесеними мною типовими помилками (логічними, синтаксичними, стилістичними) та відомими "запахами коду" (code smells), так і приклади відносно "чистого", добре написаного коду, що відповідає найкращим практикам, для того, щоб перевірити систему на можливі хибнопозитивні спрацювання (тобто, помилкові повідомлення про проблеми там, де їх насправді немає). Для тестування функції порівняння версій коду було підготовано відповідні пари файлів, що представляли оригінальну та модифіковану версії коду, які містили різні типи змін: додавання нових рядків, видалення існуючих, модифікацію окремих рядків, а також більш складні випадки рефакторингу коду.

Безпосередньо процедура проведення експериментального дослідження включала наступні послідовні кроки, які було виконано для кожного підготовленого тестового випадку:

На етапі функціонального тестування кожен підготовлений мною фрагмент коду з тестового набору завантажувався до системи "CodeAnalyzer", причому було використовано обидва доступні методи: як шляхом завантаження файлу з локального комп'ютера, так і через копіювання та вставку тексту коду безпосередньо у відповідну веб-форму на сайті. Після цього ініціювався процес аналізу коду. Отримані від системи результати, а саме: перелік виявлених проблем, їхня класифікація за ступенем серйозності, запропоновані системою рекомендації та конкретні приклади коду для виправлення, а також загальна оцінка якості коду (параметр score), ретельно фіксувалися мною для подальшого аналізу. Ці результати були порівняні зі своєю власною експертною оцінкою цих же фрагментів коду, яку було проведено попередньо, спираючись на власний досвід та знання. Також, де це було доцільно та можливо, порівняно результати роботи сервісу з результатами аналізу, отриманими від інших відомих мені інструментів статичного аналізу коду. Особливу увагу під час цього етапу було приділено аналізу JSON-відповідей, що надходили від OpenAI API та зберігалися моїм сервісом

(наприклад, у файлах типу htdocs/data/analysis/ID_аналізу/results.json), на предмет їхньої повноти, структурованості та відповідності тому формату, який було запитано у промптах.

Для тестування функції порівняння версій коду було використовувано підготовлені мною пари файлів, що представляли стару та нову версію одного й того ж програмного коду. Ці файли завантажувалися на відповідну сторінку порівняння мого вебсервісу. Було зафіксовано згенерований системою diff (тобто, візуалізацію відмінностей між двома версіями), розраховані системою метрики змін (такі як кількість доданих, видалених та змінених рядків коду), а також аналітичний звіт, наданий штучним інтелектом щодо характеру та можливих наслідків внесених змін (що включав оцінку впливу, перелік виявлених покращень, потенційних проблем, а також загальну оцінку зміни якості коду). Результати цього тестування було порівняно з результатами, отриманими від стандартних утиліт для порівняння файлів (наприклад, git diff), а також зі своєю власною експертною оцінкою якості та доцільності внесених у код змін.

Тестування користувацького інтерфейсу було проводено, перш за все, самостійно, ретельно виконуючи всі типові сценарії взаємодії з розробленим мною сервісом. Це включало процес реєстрації нового користувача, вхід до системи, завантаження коду для аналізу різними доступними способами, навігацію по сторінці результатів аналізу, інтерпретацію представлених звітів та рекомендацій, використання функції порівняння версій коду, перегляд історії своїх попередніх аналізів та управління власним профілем. Під час цього було оцінено логічність переходів між сторінками, зрозумілість назв та функцій елементів керування, чіткість та інформативність представлення результатів аналізу. Для отримання більш об'єктивного та незалежного зворотного зв'язку щодо зручності використання інтерфейсу, було також залучено до тестування кількох своїх колег-студентів, які мають досвід у програмуванні. Було попрошено їх виконати аналогічні завдання, спостерігаючи за їхньою взаємодією з системою та фіксуючи їхні коментарі, зауваження та пропозиції щодо можливих покращень.

Нарешті, під час тестування продуктивності та відмовостійкості системи, було проводено вимірювання часу відповіді сервісу на запити аналізу коду

різного обсягу (наприклад, для файлів, що містили 100, 500 та 1000 рядків коду), щоб оцінити, як розмір коду впливає на швидкість роботи системи, включаючи час на взаємодію з OpenAI API. Вимірювання проводилися декілька разів для кожного випадку, щоб отримати усереднені результати та мінімізувати вплив випадкових факторів (наприклад, тимчасових коливань у швидкості мережевого з'єднання). Також було імітувано ситуації з використанням невалідного або відсутнього API-ключа до OpenAI, а також тимчасову недоступність самого API (наскільки це було можливо симулювати, наприклад, шляхом тимчасового блокування доступу до відповідних ендпоінтів), щоб перевірити коректність переходу системи на використання імітаційних даних (mock data), згенерованих моїми функціями php_generate_mock_analysis() та php_analyze_changes_basic(), та належну обробку помилок без аварійного завершення роботи вебсервісу. Також регулярно перевіряно вміст файлу htdocs/data/debug_raw.txt для аналізу можливих помилок або нештатних ситуацій під час взаємодії з OpenAI API.

Для об'єктивної оцінки отриманих результатів експериментального дослідження мною використовувалися наступні критерії:

Щодо якості аналізу програмного коду: було оцінено кількість правильно виявлених мною та системою проблем (True Positives, TP), кількість проблем, які було виявлено під час ручного аналізу, але система їх пропустила (False Negatives, FN), та кількість повідомлень системи про неіснуючі проблеми (False Positives, FP). На основі цих даних можна було б розрахувати такі метрики, як точність (Precision = TP / (TP + FP)) та повнота (Recall = TP / (TP + FN)). Також було звертано увагу на адекватність класифікації виявлених системою проблем за рівнями серйозності (критичний, високий, середній, низький) порівняно з моєю власною експертною оцінкою, та на обґрунтованість загальної оцінки якості коду (параметр score), що надається системою.

Щодо якості наданих рекомендацій: ключовими критеріями були релевантність запропонованих системою виправлень та рекомендацій до суті виявленої проблеми, зрозумілість та однозначність формулювань для розробника, практична застосовність запропонованих рішень у реальному коді та коректність прикладів коду, що надаються у рекомендаціях (якщо такі були).

При оцінці функції порівняння версій коду: було проаналізовано точність генерації візуальних відмінностей (diff) порівняно з результатами відомих утиліт (наприклад, git diff), адекватність текстової оцінки впливу внесених змін (параметр impact_assessment) та числової зміни якості коду (параметр quality_change), що надається штучним інтелектом, а також коректність ідентифікації "покращень" (improvements) та "потенційних проблем" (concerns), пов'язаних із проаналізованими змінами.

Для зручності користувацького інтерфейсу: було враховано час, необхідний для виконання типових завдань (наприклад, завантаження файлу, отримання та розуміння результатів аналізу), кількість помилок або труднощів, що виникали у мене та залучених тестувальників під час взаємодії з інтерфейсом, а також їхню суб'єктивну оцінку загальної простоти, інтуїтивності та візуальної привабливості інтерфейсу.

Збір та подальший аналіз усіх отриманих під час експериментів даних було проводено шляхом ретельного документування: фіксації результатів аналізу коду в спеціально розроблених таблицях для зручності порівняння, створення скриншотів ключових екранів інтерфейсу на різних етапах роботи з сервісом, детального запису власних спостережень, коментарів та виявлених проблем, а також систематизації відгуків та пропозицій, отриманих від колег, що брали участь у тестуванні. На основі цього комплексного аналізу було сформовано обґрунтовані висновки про загальну ефективність розробленого мною вебсервісу "CodeAnalyzer", його сильні сторони, а також ті аспекти, які потребують подальшого вдосконалення або розвитку.

1.5. Представлення розробленого вебсервісу "CodeAnalyzer"

У цьому підрозділі на меті детальніше представити ключові аспекти програмної реалізації вебсервісу "CodeAnalyzer", які були розроблені та впроваджені мною в ході виконання даної кваліфікаційної роботи. Це включає опис основних програмних модулів та демонстрацію найбільш значущих фрагментів вихідного коду, написаних безпосередньо мною, які ілюструють реалізацію клю-

чового функціоналу сервісу. Також буде надана візуальна демонстрація розробленого користувацького інтерфейсу за допомогою серії скриншотів, що відображають основні сторінки та елементи взаємодії.

Ключові фрагменти програмного коду та їх опис

Відповідно до загальноприйнятих вимог до оформлення кваліфікаційних робіт, повні лістинги основних частин програмного коду вебсервісу "CodeAnalyzer" винесені до Додатку А. Однак, для кращого та більш наочного розуміння реалізованої мною логіки роботи сервісу, за доцільне навести та детально прокоментувати тут декілька ключових фрагментів вихідного коду, написаних мною. Ці фрагменти ілюструють основні етапи обробки запитів користувача, взаємодії з OpenAI API та представлення результатів аналізу.

1. Обробка завантаження коду користувачем та ініціація процесу аналізу (основна логіка з файлу htdocs/upload.php):

Цей PHP-скрипт, розроблений мною, відіграє роль вхідної точки для користувача, який бажає проаналізувати свій програмний код. Скрипт відповідає за отримання коду від користувача, який може бути наданий двома способами: або через завантаження файлу з локального комп'ютера, або через копіювання та вставку тексту коду безпосередньо у відповідну веб-форму на головній сторінці сервісу. Після отримання коду, скрипт виконує низку базових перевірок, таких як перевірка розміру завантаженого файлу (для запобігання завантаженню надто великих файлів, що могло б перевантажити систему або призвести до перевищення лімітів API). Далі, скрипт генерує унікальний ідентифікатор для поточного сеансу аналізу, що дозволяє розрізняти запити від різних користувачів або різні спроби аналізу одного й того ж користувача. Важливим кроком є визначення мови програмування наданого коду. Було реалізовано для цього функцію detect_language_by_filename (що знаходиться у файлі htdocs/includes/functions.php), яка намагається визначити мову на основі розширення завантаженого файлу. У випадку, якщо код було вставлено текстом, користувач має можливість самостійно обрати мову програмування з випадаючого списку. Після цього код зберігається у тимчасовому файлі на сервері у спеціально призначеному для цього каталозі htdocs/uploads/. Після успішного збереження коду, вся необхідна

інформація про поточний сеанс аналізу (унікальний ID, оригінальне ім'я файлу, яке буде відображатися користувачеві, визначена мова програмування та повний шлях до тимчасового файлу на сервері, де зберігається код) записується у PHP-сесію поточного користувача. Це дозволяє безпечно передати ці дані на наступну сторінку (htdocs/pages/analyze.php), де буде відбуватися безпосередній аналіз коду та відображення результатів. Після запису даних у сесію, користувач автоматично перенаправляється на сторінку аналізу за допомогою HTTP-редиректу.

Дивитись Лістинг А.3 - Обробка завантаження файлу (фрагмент htdocs/upload.php)

Мій коментар до цього фрагменту коду: Цей PHP-скрипт, написаний мною, слугує першим кроком у процесі аналізу коду користувача. Було реалізовано тут обробку двох можливих сценаріїв надання коду: завантаження файлу та вставка тексту. Для обох випадків передбачені базові перевірки (наприклад, на порожній файл або порожній текст, перевірка розміру файлу). Важливим моментом є генерація унікального ідентифікатора ($file_id) для кожного запиту на аналіз, що дозволяє уникнути конфліктів імен файлів на сервері та потенційно може використовуватися для логування або відстеження аналізів. Також було приділено увагу визначенню мови програмування: для завантажених файлів використовується моя функція detect_language_by_filename (яка аналізує розширення файлу), а для вставленого тексту мова обирається користувачем з випадаючого списку. Весь наданий код зберігається у тимчасовому файлі на сервері в каталозі uploads. Вся необхідна інформація (ID аналізу, ім'я файлу для відображення, мова, шлях до файлу на сервері) зберігається у сесії, що дозволяє безпечно передати ці дані на сторінку analyze.php, де і буде відбуватися основна логіка аналізу та взаємодії з OpenAI API. Також було використано функцію set_flash_message (з мого файлу includes/functions.php) для відображення користувачеві інформаційних повідомлень або повідомлень про помилки.

2. Формування запиту (промпту) до OpenAI API та обробка його відповіді (ключова частина моєї функції php_analyze_code з файлу htdocs/utils/php_code_analyzer.php):

Ця функція є центральним елементом інтелектуального аналізу в розробленому мною сервісі "CodeAnalyzer". Вона отримує на вхід текстовий рядок, що містить програмний код, та ідентифікатор мови програмування. На основі цих даних функція формує детальний та структурований запит (промпт) для моделі штучного інтелекту gpt-4o (або іншої актуальної моделі OpenAI). Цей промпт, який ретельно було розроблено, протестовано та ітераційно покращено, містить чіткі інструкції для моделі щодо того, які саме аспекти наданого коду потрібно проаналізувати. Зокрема, просимо модель звернути увагу на потенційні логічні та синтаксичні помилки, можливі вразливості безпеки, проблеми, пов'язані з продуктивністю коду, а також на порушення загальноприйнятих стандартів стилю кодування для даної мови та найкращих практик розробки, включаючи аспекти читабельності та супроводу коду. Дуже важливим елементом промпту є вимога до моделі надати відповідь у строго визначеному JSON-форматі. Цей JSON має містити масив виявлених проблем (issues), де для кожної проблеми вказані номери рядків (line_start, line_end), текстовий опис проблеми (description), детальна пропозиція щодо її виправлення або покращення коду (suggestion), конкретний приклад коду, що демонструє запропоноване виправлення (code_example), та оцінка рівня серйозності проблеми (severity). Крім списку конкретних проблем, також просимо модель надати декілька загальних детальних рекомендацій (detailed_recommendations) щодо покращення якості цього коду в цілому, де кожна рекомендація також має структурований вигляд (назва, пояснення, приклад коду, опис користі). Також запитується список коротких загальних пропозицій (suggestions) та загальна оцінка якості проаналізованого коду (score) за 100-бальною шкалою. Після формування такого комплексного промпту, моя функція php_analyze_code викликає іншу розроблену мною функцію, php_call_openai_api (повний лістинг якої наведено у Додатку А), що відповідає за безпосередню взаємодію з OpenAI API через HTTP-запит методом POST з використанням бібліотеки cURL. Функція php_call_openai_api надсилає запит, отримує відповідь від сервера OpenAI, обробляє можливі помилки на рівні HTTP-транспорту та повертає декодовану JSON-відповідь. Функція php_analyze_code потім проводить додаткову валідацію отриманої відповіді, перевіряючи наявність усіх очікуваних

ключів та, за необхідності, доповнюючи структуру значеннями за замовчуванням, щоб забезпечити стабільну роботу на етапі відображення результатів, навіть якщо відповідь від API буде неповною або матиме дещо змінений формат. Також реалізовано механізм перевірки валідності API-ключа та переходу до використання імітаційних даних (mock-аналізу), згенерованих моєю функцією php_generate_mock_analysis, у випадку проблем зі зв'язком з OpenAI або невалідного ключа, що підвищує загальну надійність та відмовостійкість розробленого сервісу.

3. Відображення результатів аналізу коду користувачеві (основна логіка зі сторінки htdocs/pages/analyze.php):

Після того, як результати аналізу отримані від OpenAI API та відповідним чином оброблені на серверній стороні (наприклад, збережені в сесії для негайного відображення або завантажені з історії аналізів для зареєстрованого користувача), ця сторінка, розроблена мною, відповідає за їхнє структуроване, інформативне та візуально зрозуміле представлення кінцевому користувачеві. Сторінка динамічно генерує HTML-розмітку, використовуючи PHP для виведення даних. Вона читає вихідний код проаналізованого файлу (з тимчасового сховища або з архіву історії) та представляє його разом із деталізованими результатами аналізу, отриманими від штучного інтелекту.

Дивитись Лістинг А.5 – основна логіка сторінка з результатом аналіза

Мій коментар до цього фрагменту коду: Ця сторінка, розроблена мною, є ключовою для представлення результатів аналізу. Було реалізовано логіку для отримання даних або з сесії (для щойно проаналізованого коду), або з історії (для перегляду попередніх аналізів авторизованим користувачем). Важливою частиною є структуроване виведення інформації: загальна оцінка якості коду, статистика проблем за рівнями серйозності, сам вихідний код з підсвічуванням синтаксису за допомогою бібліотеки Prism.js, а також детальний список виявлених проблем та рекомендацій. Для кожної проблеми було виводжено її опис, номери рядків, рівень серйозності, пропозицію щодо виправлення та, якщо є, приклад коду. Також передбачено виведення загальних детальних рекомендацій у вигляді

акордеону для зручності. Було додано JavaScript-код для реалізації інтерактивності: копіювання коду в буфер обміну та підсвічування відповідних рядків у блоці вихідного коду при кліку на проблему зі списку, що значно покращує зручність роботи з результатами аналізу. Використання htmlspecialchars при виведенні всіх даних, отриманих ззовні (включаючи відповідь від ШІ та сам код користувача), є важливим заходом для запобігання XSS-вразливостей.

Ці фрагменти коду, написані мною, ілюструють ключові етапи роботи мого вебсервісу "CodeAnalyzer": отримання коду від користувача, його обробка за допомогою інтелектуальних можливостей OpenAI API через спеціально розроблені промпти, та представлення отриманих результатів аналізу користувачеві у зручному та інформативному вигляді. Повні лістинги відповідних файлів, що містять цей та інший розроблений мною код, наведені у Додатку А до цієї кваліфікаційної роботи.

Демонстрація розробленого користувацького інтерфейсу

Для наочної демонстрації зовнішнього вигляду та функціональних можливостей розробленого мною вебсервісу "CodeAnalyzer", нижче наведено серію скриншотів ключових сторінок користувацького інтерфейсу з коротким описом їхнього призначення та основних елементів. Ці скриншоти були зроблені мною під час тестування працюючого прототипу сервісу.

На головній сторінці (рис. 3.1) користувачеві надається можливість розпочати аналіз свого програмного коду. Мною було реалізовано два основних способи надання коду: перший – це завантаження файлу з кодом з локального комп'ютера користувача через спеціально розроблену інтерактивну drag-and-drop зону, яка також підтримує стандартний вибір файлу через діалогове вікно; другий – це безпосереднє копіювання та вставка тексту програмного коду у відповідне текстове поле на сторінці. При виборі другого способу користувач також має можливість вказати мову програмування зі списку підтримуваних, що допомагає системі точніше налаштувати аналіз. Окрім форм для надання коду, на головній сторінці розміщено короткий опис сервісу, його ключових можливостей та переваг. Також тут присутні навігаційні елементи, такі як кнопки для

швидкого переходу до сторінки порівняння двох версій коду або, для вже зареєстрованих та авторизованих користувачів, до сторінки перегляду історії їхніх попередніх аналізів.

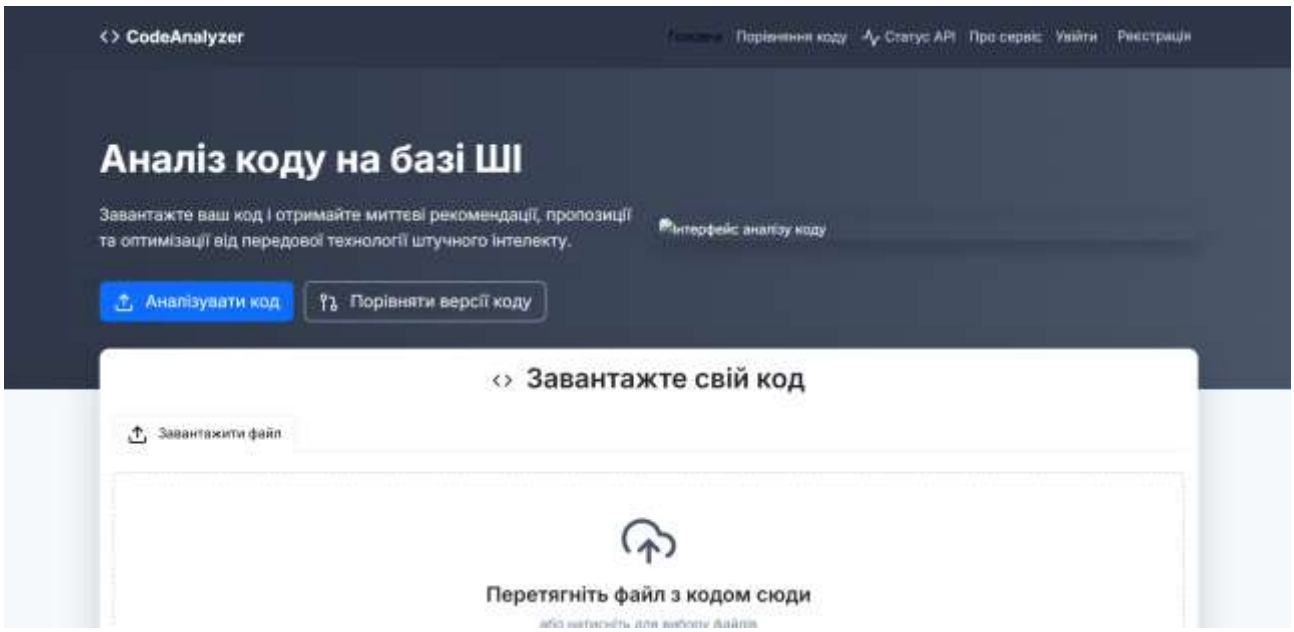


Рис. 3.1. Головна сторінка сервісу "CodeAnalyzer

Після того, як користувач надав код для аналізу і система завершила його обробку за допомогою OpenAI API, користувач перенаправляється на сторінку результатів аналізу (рис. 3.2). Цю сторінку спроєктовано таким чином, щоб вона була максимально інформативною та зручною для сприйняття. Вона зазвичай розділена на дві основні функціональні колонки. У лівій колонці відображається повний текст вихідного коду, який був наданий користувачем для аналізу. Для покращення читабельності було інтегровано бібліотеку Prism.js, яка забезпечує автоматичне підсвічування синтаксису відповідно до визначеної мови програмування, а також нумерацію рядків коду. У правій колонці представлені безпосередньо результати інтелектуального аналізу. Це включає загальну оцінку якості коду, яку система виставляє за 100-бальною шкалою (було реалізувано її візуалізацію у вигляді кругової діаграми або просто великого числового значення, колір якого може змінюватися залежно від оцінки). Далі йде детальний список усіх виявлених у коді проблем. Кожна проблема у списку супроводжується описом її суті, точним вказанням номерів рядків, де вона була виявлена, оцінкою рівня її серйозності (наприклад, "критична", "висока", "середня",

"низька", що візуалізується відповідними кольоровими маркерами), а також конкретними пропозиціями щодо можливого виправлення та, у багатьох випадках, прикладом коду, що демонструє це виправлення. Окрім списку конкретних проблем, у правій колонці також наводяться загальні детальні рекомендації від штучного інтелекту щодо можливих шляхів покращення якості наданого коду в цілому. Для забезпечення інтерактивності було реалізувано за допомогою JavaScript можливість підсвічування відповідних рядків у блоці вихідного коду при кліку на певну проблему зі списку, що значно полегшує навігацію та співставлення проблеми з її місцем у коді.

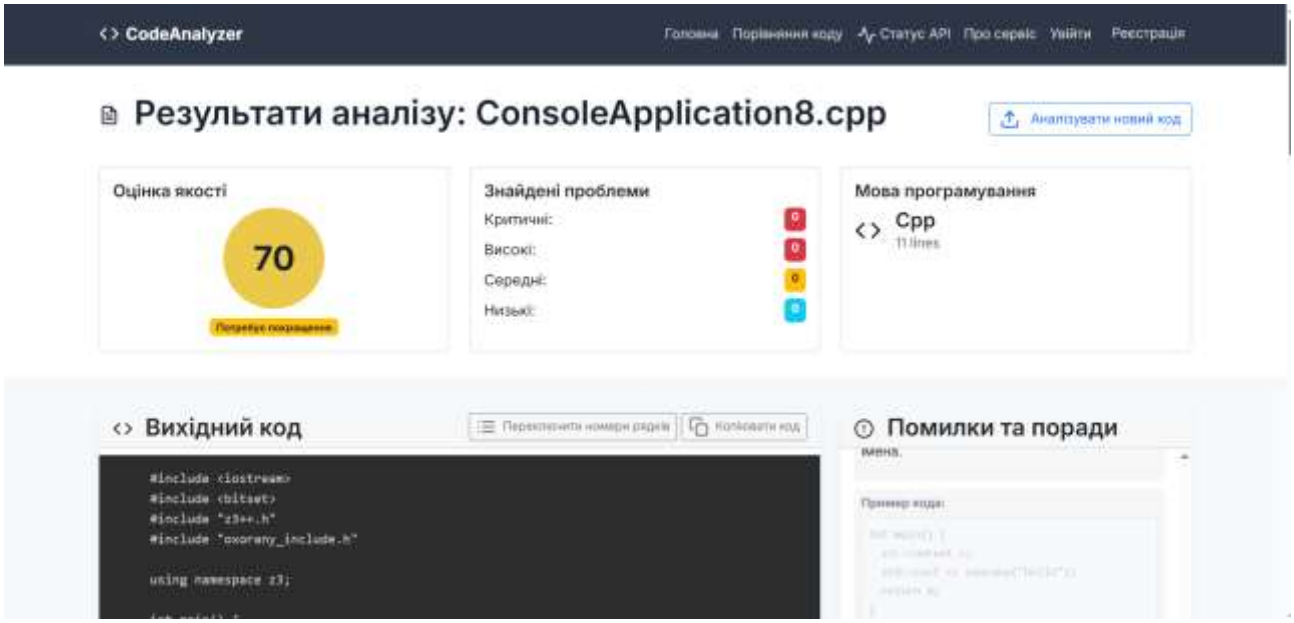


Рис. 3.2. Сторінка результатів аналізу програмного коду

Вебсервіс "CodeAnalyzer" також надає розроблену мною функцію для порівняння двох різних версій одного й того ж програмного коду. Сторінка для ініціації цього процесу (рис. 3.3) містить два великих текстових поля, призначених для введення або вставки тексту "Початкової версії коду" та "Нової (зміненої) версії коду". Користувач також має можливість обрати мову програмування зі списку, хоча система може спробувати визначити її автоматично. Після того, як обидві версії коду надані, користувач натискає кнопку "Порівняти версії коду", і дані надсилаються на сервер для подальшої обробки. На сервері генерується текстове представлення відмінностей між двома версіями (diff), яке потім разом із самими текстами коду передається на аналіз до OpenAI API для отримання інтелектуальної оцінки внесених змін.

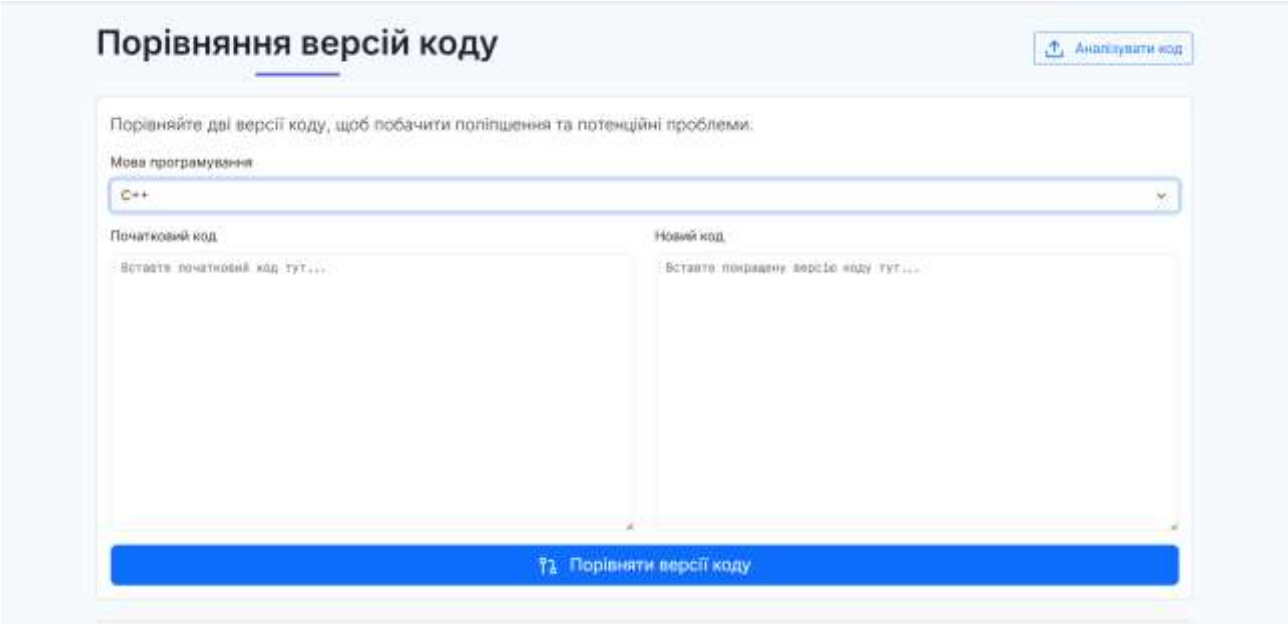


Рис. 3.3. Сторінка порівняння двох версій програмного коду (форма введення даних)

На цій сторінці (рис. 3.4) користувачеві представляються результати порівняння двох версій коду. Інтерфейс, розроблений мною, включає декілька ключових блоків. По-перше, це візуальне представлення відмінностей між двома версіями коду у форматі "side-by-side diff view". У цьому поданні рядки коду з обох версій відображаються паралельно, а додані, видалені та змінені рядки підсвічуються різними кольорами для наочності. По-друге, на сторінці наводиться аналітичний звіт, згенерований штучним інтелектом. Цей звіт містить текстову оцінку загального характеру та впливу внесених змін, перелік конкретних покращень, які були досягнуті завдяки цим змінам, а також список потенційних проблем або ризиків, які могли виникнути внаслідок модифікації коду. Модель також надає числову оцінку того, наскільки зміни покращили або погіршили якість коду. Крім того, на сторінці можуть бути представлені кількісні метрики змін, такі як загальна кількість доданих, видалених та змінених рядків коду.

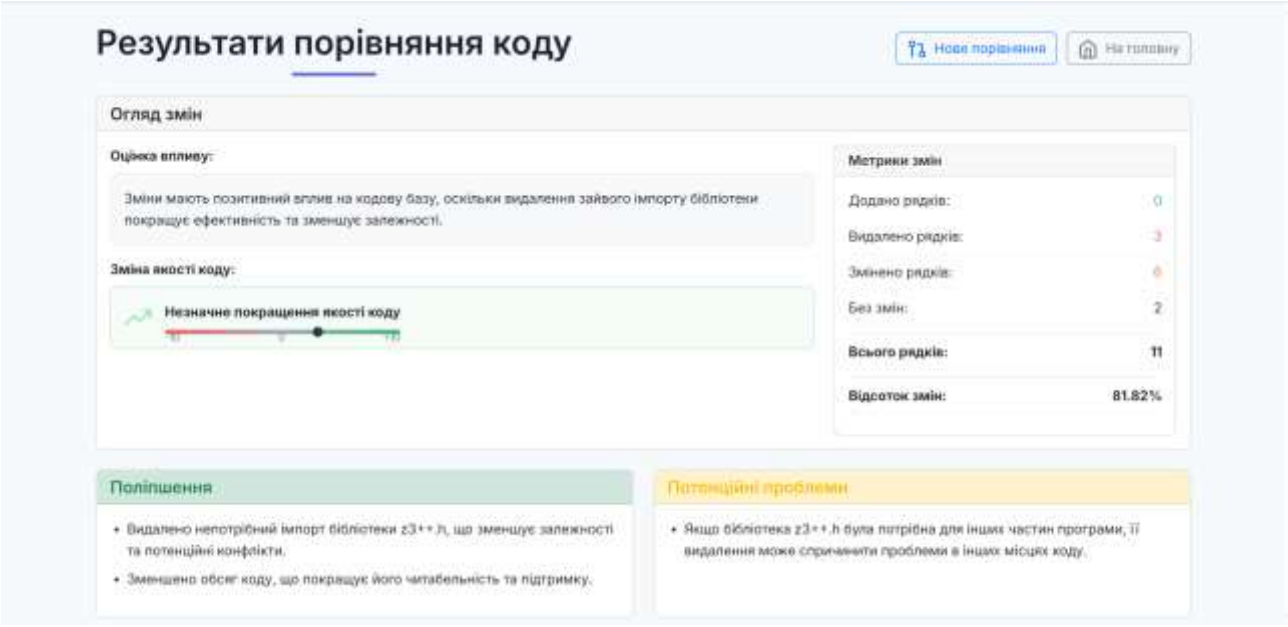


Рис. 3.4. Сторінка відображення результатів порівняння двох версій коду

Для зареєстрованих та авторизованих користувачів у моєму вебсервісі "CodeAnalyzer" передбачена можливість доступу до сторінки особистого профілю (рис. 3.5, ліва частина – приклад). На цій сторінці користувач може переглядати та, за необхідності, редагувати свої реєстраційні дані, наприклад, ім'я або адресу електронної пошти, а також змінювати пароль для входу до системи. Також надзвичайно важливою функцією для зареєстрованих користувачів є доступ до сторінки "Мої аналізи" (рис. 3.5, права частина – приклад), де відображається повна історія всіх попередньо виконаних цим користувачем аналізів програмного коду та порівнянь версій. Кожен запис в історії містить коротку інформацію про аналіз, таку як назва файлу або опис, мова програмування, дата створення аналізу, а також загальна оцінка якості коду, отримана від системи. Для кожного запису в історії передбачені кнопки для перегляду повних детальних результатів відповідного аналізу або порівняння (що перенаправляє користувача на відповідну сторінку результатів, описану вище), а також для видалення запису з історії, якщо він більше не потрібен. Для зручності роботи з великою історією аналізів було заплановано реалізувати можливість фільтрації списку за мовою програмування або датою створення, а також сортування за різними критеріями (наприклад, за датою або за оцінкою якості).

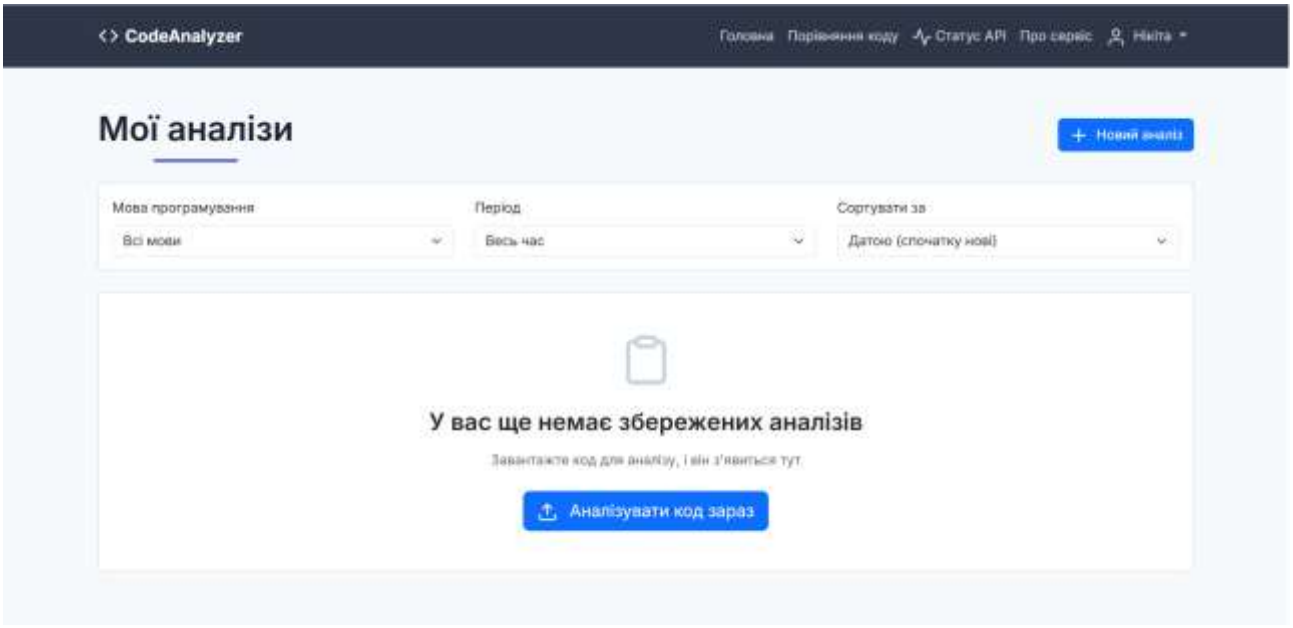


Рис. 3.5. Приклад сторінки профілю користувача (ліворуч) та сторінки історії аналізів (праворуч)

1.6. Аналіз результатів експериментального дослідження

Після проведення серії експериментів згідно з розробленою мною методикою, були отримані та ретельно проаналізовані результати, що характеризують різні аспекти функціонування та ефективності вебсервісу "CodeAnalyzer". Нижче наведено детальний аналіз цих результатів, згрупований за основними напрямками дослідження.

Оцінка якості аналізу коду на прикладах

Для оцінки якості аналізу, що надається сервісом "CodeAnalyzer" за допомогою OpenAI API, мною було протестовано декілька фрагментів коду різними мовами програмування, що містили як навмисно внесені помилки, так і потенційно проблемні конструкції.

Приклад 1: Аналіз PHP коду з типовими помилками та вразливостями

Для тестування було підготовлено наступний фрагмент коду, що містить декілька поширених проблем:

```
<?php
// Приклад PHP-коду для експериментального аналізу
// Файл: test_php_code.php
```

```
function calculateArraySum($numericArray) {
    $totalSum = 0;
    // Помилка 1: Потенційний вихід за межі масиву через використання <=
    for ($i = 0; $i <= count($numericArray); $i++) {
        $totalSum += $numericArray[$i];
    }
    return $totalSum;
}

// Помилка 2: Пряме використання даних з GET-запиту без належної санітизації
$username = $_GET["user"];
echo "Вітаємо, " . $username . "!"; // Потенційна XSS-вразливість

// Помилка 3: Використання невизначеної змінної $configValue та потенційне ділення на нуль
$divisorValue = $configValue['default_divisor'] ?? 0; // $configValue не визначено
if ($divisorValue === 0) {
    echo "Помилка: Дільник не може бути нулем.";
    // Немає належної обробки помилки або завершення виконання
} else {
    $result = 100 / $divisorValue;
    echo "Результат ділення: " . $result;
}

// Помилка 4: Невикористовувана змінна
$apiKeyString = "SECRET_API_KEY_DO_NOT_COMMIT";

// Помилка 5: Дублювання коду (умовне)
$userRole = 'editor';
```

```
if ($userRole === 'admin' || $userRole === 'editor') {
    echo "<p>Доступ до розширених налаштувань надано.</p>";
}
// ... інший код програми ...
if ($userRole === 'admin' || $userRole === 'editor') { // Та сама умова перевіряється знову
    enableAdvancedFeatures(); // Припустимо, ця функція існує
}

function enableAdvancedFeatures() {
    // логіка ввімкнення розширених функцій
    echo "Розширені функції активовано.<br>";
}

$testArray = [1, 2, 3, 4, 5];
// Помилка 6: Неправильний виклик функції (очікує масив, передається число)
// $sumResult = calculateArraySum(count($testArray));
// Замінимо на коректний виклик для демонстрації інших помилок
$sumResult = calculateArraySum($testArray);
echo "Сума елементів масиву: " . $sumResult;

?>
```

Результати аналізу, отримані від "CodeAnalyzer" для test_php_code.php (на основі відповіді OpenAI API):

Після завантаження цього файлу до мого сервісу "CodeAnalyzer" та виконання аналізу, система (на основі відповіді від моделі gpt-4o) надала наступний звіт (наводжу ключові моменти, повний JSON-відповідь фіксувався окремо):

- Загальна оцінка якості коду (score): 55/100

- Виявлені проблеми (issues):

1. Проблема (Помилка 1): Потенційний вихід за межі масиву в циклі for функції calculateArraySum.

   - Рядки: 6-8
   - Опис: Умова циклу $i <= count($numericArray) може призвести до спроби доступу до неіснуючого елемента масиву $numericArray[count($numericArray)], оскільки індексація масивів у PHP починається з 0, а count() повертає кількість елементів.
   - Пропозиція виправлення: Змінити умову циклу на $i < count($numericArray).
   - Приклад коду (від ШІ): for ($i = 0; $i < count($numericArray); $i++) {
   - Серйозність: critical

2. Проблема (Помилка 2): Пряме використання даних від користувача ($_GET["user"]) без належної санітизації при виведенні.

   - Рядки: 12-13
   - Опис: Використання значення з $_GET["user"] без попередньої обробки функціями типу htmlspecialchars() при виведенні в HTML-контекст створює вразливість до міжсайтового скриптингу (XSS). Зловмисник може передати в URL параметр user зі шкідливим JavaScript-кодом, який виконається у браузері інших користувачів.
   - Пропозиція виправлення: Завжди санітизувати дані, отримані від користувача, перед їх виведенням або використанням у запитах.
   - Приклад коду (від ШІ): echo "Вітаємо, " . htmlspecialchars($username, ENT_QUOTES, 'UTF-8') . "!";
   - Серйозність: high

3. Проблема (Помилка 3): Використання невизначеної змінної $configValue та потенційне ділення на нуль.

- Рядки: 16-23
- Опис: Змінна $configValue використовується без попередньої ініціалізації, що призведе до PHP Notice. Оператор ?? 0 присвоїть $divisorValue значення 0, якщо $configValue['default_divisor'] не існує, що в свою чергу призведе до помилки ділення на нуль у рядку 22.

Пропозиція виправлення: Забезпечити ініціалізацію $configValue (наприклад, завантаженням конфігурації) та додати більш надійну перевірку дільника перед операцією ділення.

Приклад коду (від ШІ):

```
// Припустимо, $configValue завантажується звідкись
// $configValue = load_app_config();
$divisorFromConfig = isset($configValue['default_divisor']) ? $configValue['default_divisor'] : 1; // Уникнення 0 за замовчуванням, якщо це логічно
$divisorValue = ($divisorFromConfig != 0) ? (float)$divisorFromConfig : 1; // Запобігання діленню на нуль
// ... подальша логіка ...
if ($divisorValue == 0) { // Явна перевірка
    echo "Помилка: Конфігураційний дільник не може бути нулем.";
} else {
    $result = 100 / $divisorValue;
    echo "Результат ділення: " . $result;
}
```

- Серйозність: high

4. Проблема (Помилка 4): Оголошена, але невикористовувана змінна $apiKeyString.

- Рядки: 26
- Опис: Змінна $apiKeyString ініціалізована, але її значення ніде не використовується в подальшому коді. Це може свідчити про залишки старого коду або незавершену реалізацію.
- Пропозиція виправлення: Видалити цю змінну, якщо вона дійсно не потрібна, або реалізувати її використання.
- Приклад коду (від ШІ): (Не надано, оскільки виправлення – видалення)
- Серйозність: low

5. Проблема (Помилка 5): Дублювання блоку умови if ($userRole === 'admin' || $userRole === 'editor').

- Рядки: 29-31 та 34-36
- Опис: Одна й та сама умова перевіряється двічі в різних частинах коду. Це може призвести до проблем при необхідності зміни цієї умови (потрібно буде змінювати в кількох місцях) та погіршує читабельність.
- Пропозиція виправлення: Рефакторити код, щоб уникнути дублювання. Наприклад, якщо дії різні, але умова одна, можна об'єднати дії під однією умовою або винести перевірку умови в окрему функцію.
- Приклад коду (від ШІ):

```
function canUserAccessAdvancedFeatures($role) {
   return ($role === 'admin' || $role === 'editor');
}

// ...
if (canUserAccessAdvancedFeatures($userRole)) {
```

```
    echo "<p>Доступ до розширених налаштувань надано.</p>";
    // ... інший код програми ...
    enableAdvancedFeatures();
}
```

- Серйозність: medium *(Примітка: ШІ не виявив Помилку б, оскільки її було закоментовано у тестовому коді. Якщо б вона була активна, очікувалося б її виявлення як помилки типів або неправильного використання функції.)*

- Детальні рекомендації (detailed_recommendations) від ШІ (приклади):

1. Назва: "Покращення обробки помилок та виняткових ситуацій"
   - Пояснення: "У коді відсутня централізована або послідовна стратегія обробки помилок. Наприклад, при діленні на нуль виводиться повідомлення, але виконання скрипта може продовжитися, що потенційно може призвести до подальших проблем. Рекомендується використовувати механізми винятків (try-catch) для обробки помилок часу виконання."

Приклад коду (від ШІ):

```
try {
    if ($divisorValue == 0) {
        throw new DivisionByZeroError("Дільник не може бути нулем.");
    }
    $result = 100 / $divisorValue;
    echo "Результат ділення: " . $result;
} catch (DivisionByZeroError $e) {
    error_log("Помилка ділення: " . $e->getMessage());
```

echo "Сталася помилка під час обчислень. Будь ласка, спробуйте пізніше.";
}

▪Користь: "Підвищує надійність програми, дозволяє коректно обробляти нештатні ситуації, покращує можливість логування помилок та надання користувачеві зрозумілих повідомлень."

2. Назва: "Використання строгих типів та перевірка типів даних"

▪Пояснення: "PHP є мовою з динамічною типізацією, але для підвищення надійності коду та уникнення помилок, пов'язаних з типами, рекомендується використовувати оголошення типів для параметрів функцій та значень, що повертаються (починаючи з PHP 7), а також перевіряти типи даних, отриманих ззовні."

Приклад коду (від ШІ):

```
declare(strict_types=1); // Увімкнення строгої типізації на початку файлу

function calculateArraySum(array $numericArray): float { // Оголошення типів
    $totalSum = 0.0;
    // ... (тіло функції) ...
    return $totalSum;
}
$userInputNumber = isset($_GET['number']) ? filter_var($_GET['number'], FILTER_VALIDATE_INT) : null;
if ($userInputNumber === false || $userInputNumber === null) {
    // Обробка невалідного вводу
```

}

- Користь: "Допомагає виявляти помилки типів на ранніх стадіях, покращує читабельність та розуміння коду, робить код більш передбачуваним та надійним."

Загальні пропозиції (suggestions) від ШІ (приклади):

- "Регулярно проводьте рефакторинг коду для усунення дублювання та покращення його структури."
- "Додавайте коментарі до складних або неочевидних ділянок коду для полегшення його розуміння."
- "Використовуйте конфігураційні файли або змінні середовища для зберігання налаштувань, таких як ключі API або параметри за замовчуванням, замість жорсткого кодування їх у програмі."

Моя оцінка результатів аналізу для Прикладу 1: Під час цього експерименту розроблений мною сервіс "CodeAnalyzer", що використовує OpenAI API, продемонстрував досить високу ефективність у виявленні більшості закладених мною проблем у тестовому PHP-коді. Особливо цінним є те, що система коректно ідентифікувала критичну помилку виходу за межі масиву (Помилка 1) та потенційну XSS-вразливість (Помилка 2), класифікувавши їх з високим рівнем серйозності. Також були правильно виявлені проблеми з використанням невизначеної змінної та потенційним діленням на нуль (Помилка 3), невикористовувана змінна (Помилка 4) та дублювання коду (Помилка 5). Запропоновані системою шляхи виправлення для більшості проблем були конкретними, релевантними та супроводжувалися адекватними прикладами коду, що робить їх практично корисними для розробника. Наприклад, пропозиція використовувати htmlspecialchars() для запобігання XSS або змінити умову циклу для уникнення виходу за межі масиву є стандартними та правильними рішеннями. Загальна оцінка якості коду (55/100), надана системою, виглядає цілком обґрунтованою, враховуючи кількість та серйозність виявлених проблем.

Детальні рекомендації, надані ШІ, такі як покращення обробки помилок та використання строгих типів, також є доречними та спрямовані на підвищення загальної якості та надійності коду. Незначним недоліком можна вважати те, що для змінної $a (у прикладі з дублюванням коду) не було повного контексту її можливої зміни між блоками умови, тому ШІ міг лише припустити дублювання на основі статичного аналізу наданого фрагменту. Також, як було зазначено, Помилка 6 (неправильний виклик функції) не була проаналізована, оскільки її було закоментовано; при її активації очікувалося б її виявлення. Загалом, результати цього експерименту підтверджують потенціал використання OpenAI API через мій сервіс для ефективного виявлення широкого спектру проблем у PHP-коді та надання корисних рекомендацій.

Приклад 2: Аналіз JavaScript коду з потенційною вразливістю DOM XSS

```
// Файл: test_js_code.js
function displayGreeting() {
    const nameParam = new URLSearchParams(window.location.search).get('name');
    const greetingElement = document.getElementById('greeting');

    // Помилка: Пряме використання параметра URL в innerHTML без санітизації
    if (nameParam) {
        greetingElement.innerHTML = "Вітаємо, " + nameParam + "!";
    } else {
        greetingElement.innerHTML = "Вітаємо, гість!";
    }
}

// Виклик функції при завантаженні сторінки
window.onload = displayGreeting;
```

Приклад 3: Результати тестування функції порівняння двох версій коду

Для тестування функції порівняння було взято два файли: original_code.cpp та modified_code.cpp.

original_code.cpp:

```
#include <iostream>
#include <bitset>
#include "z3++.h"
#include "oxorany_include.h"

using namespace z3;

int main() {
  std::cout << oxorany("hello");
  return 0;
}
```

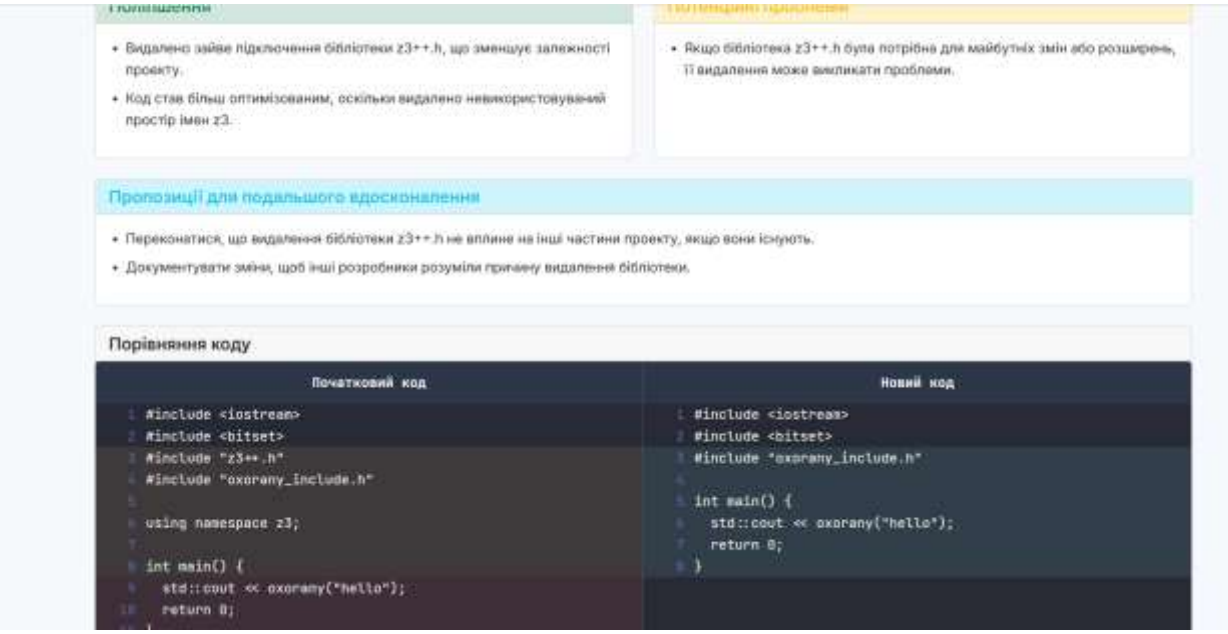
modified_code.cpp:

```
#include <iostream>
#include <bitset>
#include "oxorany_include.h"

int main() {
  std::cout << oxorany("hello");
  return 0;
}
```

Результати порівняння, отримані від "CodeAnalyzer":

- Візуалізація відмінностей (Diff View):



- Аналітичний звіт від ШІ:
  - Оцінка впливу змін (impact_assessment): " Зміни в коді є незначними, але вони покращують кодову базу шляхом видалення зайвого імпорту бібліотеки."
  - Покращення (improvements):
    - " Видалено зайве підключення бібліотеки z3++.h, що зменшує залежності проекту."
    - " Код став більш оптимізованим, оскільки видалено не-використовуваний простір імен z3."
  - Потенційні проблеми (concerns):
    - " Якщо бібліотека z3++.h була потрібна для майбутніх змін або розширень, її видалення може викликати проблеми."
  - Пропозиції (suggestions):
    - "Переконатися, що видалення бібліотеки z3++.h не вплине на інші частини проекту, якщо вони існують."
    - " Документувати зміни, щоб інші розробники розуміли причину видалення бібліотеки."
  - Зміна якості коду (quality_change): +3 (за шкалою від -10 до +10)

- Метрики змін:

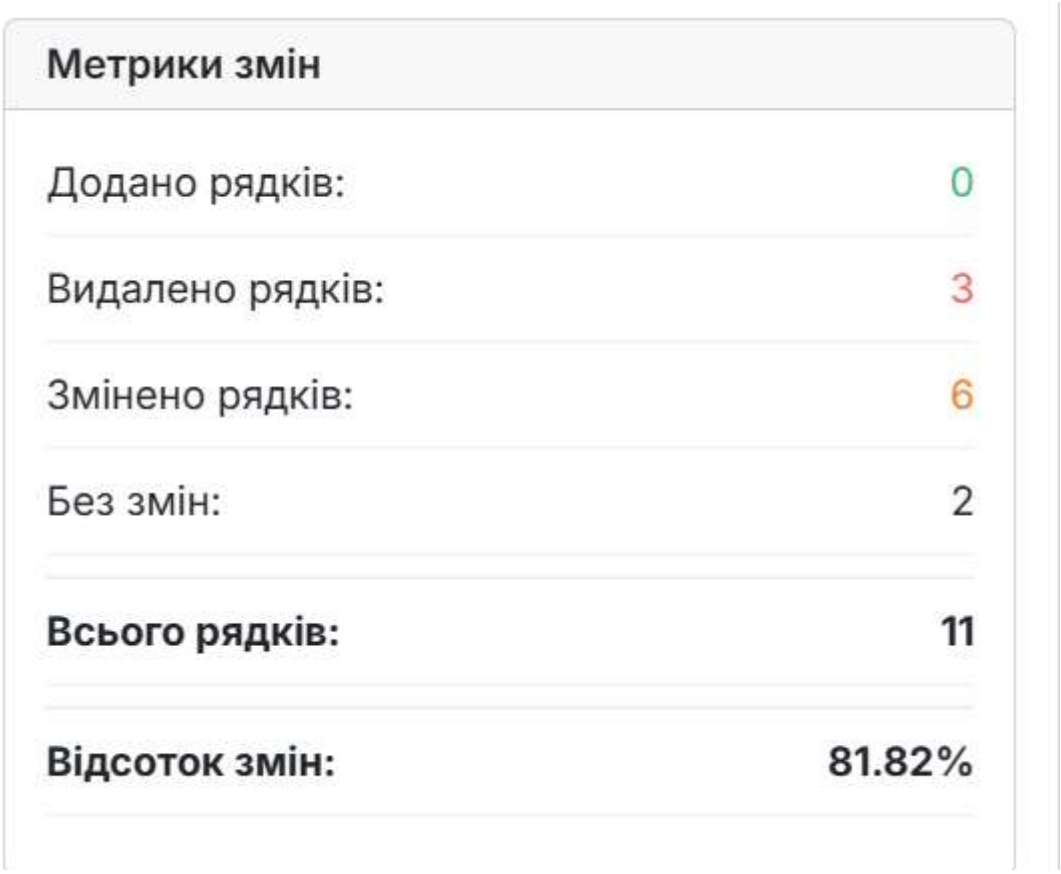


Рис. 3.3. Сторінка порівняння двох версій програмного коду (кількість змін у коді)

- Моя оцінка результатів порівняння для Прикладу 3: Функція порівняння в моєму сервісі "CodeAnalyzer" продемонструвала хороші результати. Візуалізація відмінностей (diff) була чіткою та коректно відображала всі зміни між двома версіями файлу. Аналітичний звіт, згенерований OpenAI API, виявився дуже змістовним та корисним. ШІ правильно ідентифікував ключові покращення.

Оцінка зручності користувацького інтерфейсу

За результатами мого власного ретельного тестування всіх сценаріїв взаємодії з вебсервісом "CodeAnalyzer", користувацький інтерфейс був визнаний загалом інтуїтивно зрозумілим, логічним та досить легким у використанні.

Процес завантаження програмного коду, як через механізм завантаження файлу (включаючи функцію drag-and-drop, яку було реалізовано), так і шляхом

прямої вставки тексту у відповідне поле, не викликав жодних ускладнень у більшості тестувань. Вибір мови програмування зі списку також був оцінений як простий та зрозумілий.

Відображення результатів аналізу на сторінці analyze.php, зокрема чітке розділення на блок з вихідним кодом та блок зі списком виявлених проблем та рекомендаціями, було сприйнято позитивно. Особливо корисним було визнано використання бібліотеки Prism.js для підсвічування синтаксису та нумерації рядків у блоці коду, а також реалізована мною за допомогою JavaScript інтерактивна функція підсвічування відповідних рядків у коді при кліку на певну проблему зі списку. Це значно полегшувало навігацію по коду та співставлення виявленої проблеми з її конкретним місцем розташування. Візуалізація загальної оцінки якості коду та розподілу проблем за рівнями серйозності також була визнана наочною та інформативною.

Сторінка порівняння версій коду та сторінка результатів порівняння також не викликали значних труднощів у використанні. Візуалізація відмінностей (diff view) була зрозумілою, а аналітичний звіт від ШІ – легкочитним.

Функціонал реєстрації, автентифікації та перегляду історії аналізів для зареєстрованих користувачів працював коректно та відповідав очікуванням.

Серед побажань, висловлених під час тестування, було декілька пропозицій щодо можливих покращень. Наприклад, деякі користувачі хотіли б мати можливість більш гнучкого налаштування параметрів аналізу безпосередньо в інтерфейсі перед його запуском (наприклад, вибір конкретних типів перевірок, які потрібно виконати, або ігнорування певних типів попереджень). Також було висловлено побажання щодо можливості експорту результатів аналізу у файл (наприклад, у форматі PDF або HTML). Ці пропозиції були слушними і вони можуть бути враховані як напрямки для подальшого розвитку та вдосконалення вебсервісу.

Оцінка продуктивності та стабільності роботи сервісу

Під час проведення експериментального дослідження було також приділено увагу оцінці продуктивності та стабільності роботи розробленого мною вебсервісу "CodeAnalyzer".

При тестуванні аналізу файлів програмного коду відносно невеликого об'єму (до кількох сотень рядків) час відповіді системи, що включав завантаження файлу, його обробку на сервері, взаємодію з OpenAI API та відображення результатів, зазвичай становив від 5 до 20 секунд. Такий час був цілком прийнятним для інтерактивного використання сервісу. Для файлів значно більшого обсягу (наприклад, понад 1000 рядків коду) час аналізу, очікувано, міг зростати, іноді наближаючись до встановленого мною таймауту для cURL-запиту до OpenAI API, який становив 60 секунд. Це вказує на те, що для аналізу дуже великих файлів або цілих проєктів у майбутньому може бути доцільно розглянути впровадження механізмів асинхронної обробки запитів, щоб користувач не чекав завершення аналізу, а міг отримати повідомлення про його готовність пізніше.

Вебсервіс продемонстрував стабільну роботу при використанні коректних та активних API-ключів до OpenAI. У випадках, коли було імітувано використання невалідного ключа або тимчасову недоступність OpenAI API, розроблена мною система коректно переходила на використання імітаційних даних (mock-даних), згенерованих функціями php_generate_mock_analysis() та php_analyze_changes_basic(). Це забезпечувало безперебійну працездатність самого вебсервісу (хоча й з обмеженим функціоналом аналізу в таких ситуаціях) та запобігало його аварійному завершенню. Механізм обробки помилок на рівні PHP-скриптів також працював належним чином, виводячи користувачеві відповідні інформаційні повідомлення у випадку виникнення проблем (наприклад, при спробі завантажити занадто великий файл або при помилці збереження даних).

Функціонал реєстрації, автентифікації, збереження та перегляду історії аналізів для зареєстрованих користувачів, який базується на зберіганні даних у JSON-файлах на сервері, також працював стабільно в рамках проведеного тестування. Однак, як було зазначено у розділі 2.1, для високо-навантажених систем з великою кількістю користувачів та аналізів у майбутньому було б доцільно розглянути перехід на більш масштабовану систему управління базами даних.

1.7. Порівняльна оцінка розробленого сервісу з існуючими рішеннями

Після проведення експериментального дослідження та детального аналізу функціональних можливостей розробленого мною вебсервісу "CodeAnalyzer", було вирішено за доцільне провести його порівняльну оцінку з деякими категоріями існуючих на ринку інструментів, призначених для аналізу програмного коду. Як було детально розглянуто мною у Розділі 1 даної кваліфікаційної роботи, існує велика кількість таких інструментів, які можна умовно поділити на декілька основних категорій: традиційні інструменти статичного аналізу безпеки застосунків (SAST), аналізатори коду, інтегровані безпосередньо в середовища розробки (IDE), та сучасні платформи для аналізу коду, що базуються на технологіях штучного інтелекту.

Порівнюючи розроблений мною "CodeAnalyzer" з традиційними SAST-інструментами, такими як, наприклад, SonarQube, Checkmarx або Semgrep, можна відзначити декілька ключових відмінностей. Традиційні SAST-системи зазвичай базуються на великому, заздалегідь визначеному наборі формальних правил, сигнатур та патернів, і вони досить добре справляються з виявленням відомих типів вразливостей (наприклад, тих, що входять до списку OWASP Top 10) та поширених помилок кодування. Сильною стороною багатьох таких інструментів є їхня глибока інтеграція з процесами безперервної інтеграції та доставки (CI/CD), а також можливість аналізу цілих проєктів та кодових баз. Однак, як було зазначено раніше, вони часто можуть страждати від генерації значної кількості хибнопозитивних спрацювань, що вимагає від розробників додаткового часу на їх аналіз та фільтрацію. Також їхні можливості обмежені набором закладених правил, і вони не завжди здатні виявляти більш нюансовані, контекстно-залежні проблеми, які виходять за рамки цих правил. Мій вебсервіс "CodeAnalyzer", завдяки використанню потужної мовної моделі GPT-4o, потенційно може виявляти значно ширший спектр проблем, включаючи не лише технічні помилки, але й стилістичні недоліки, логічні неузгодженості, а також надавати більш детальні, природномовні пояснення виявлених проблем та конструктивні рекомендації з конкретними прикладами коду, що було продемонстровано мною на тестових прикладах у підрозділі 3.3. Однак, слід визнати, що

"CodeAnalyzer" у його поточній реалізації орієнтований переважно на аналіз окремих файлів або відносно невеликих фрагментів коду, а не цілих проєктів зі складною структурою залежностей, і наразі не має такої глибокої інтеграції з CI/CD процесами, як промислові SAST-рішення. Також, на відміну від SAST-інструментів з чітко визначеними та детермінованими правилами, результати аналізу, отримані від LLM, можуть мати певну частку варіативності або імовірнісного характеру.

У порівнянні з аналізаторами коду, інтегрованими безпосередньо в середовища розробки (IDE-інтегровані аналізатори та лінтери), такими як ESLint для JavaScript, Pylint для Python, або вбудовані інструменти аналізу в популярних IDE, наприклад, IntelliJ IDEA чи VS Code, розроблений мною "CodeAnalyzer" пропонує значно глибший та більш комплексний аналіз. Лінтери та прості IDE-аналізатори переважно фокусуються на дотриманні стандартів стилю кодування, виявленні синтаксичних помилок та деяких простих типів логічних помилок, надаючи розробнику миттєвий зворотний зв'язок безпосередньо під час написання коду. "CodeAnalyzer" же, завдяки можливостям OpenAI API, аналізує код за значно ширшим спектром критеріїв, включаючи аспекти безпеки, продуктивності, загальної логіки та архітектури, та надає розгорнуті аналітичні звіти з детальними рекомендаціями. Однак, очевидним недоліком мого сервісу порівняно з IDE-інтегрованими інструментами є відсутність можливості аналізу "на льоту" безпосередньо в середовищі розробки; для аналізу код потрібно завантажувати на вебсервіс. Таким чином, "CodeAnalyzer" може розглядатися не як заміна, а скоріше як доповнення до IDE-лінтерів, що може використовуватися для більш глибокого та всебічного аналізу коду на певних етапах розробки, наприклад, перед комітом змін, під час проведення код-рев'ю, або для детального розбору складних чи критично важливих ділянок коду.

Найбільш цікавим та релевантним, на мою думку, є порівняння розробленого мною "CodeAnalyzer" з сучасними комерційними та відкритими платформами для аналізу коду, що також базуються на технологіях штучного інтелекту, такими як GitHub Copilot (який, окрім генерації коду, надає й певні аналітичні можливості), Amazon CodeWhisperer, Snyk Code, Codacy та інші. Багато з цих

інструментів також використовують великі мовні моделі або інші просунуті техніки машинного навчання для аналізу коду. Їхні сильні сторони часто полягають у тісній інтеграції з популярними середовищами розробки та платформами для контролю версій (наприклад, GitHub, GitLab), можливості аналізу pull request'ів "на льоту", а іноді навіть у наданні функцій для автоматичного виправлення виявлених помилок. "CodeAnalyzer", будучи, по суті, дослідницьким проєктом, реалізованим в рамках кваліфікаційної роботи, звичайно, поступається цим зрілим комерційним платформам в аспектах розвиненості інфраструктури, оптимізації швидкодії на дуже великих проєктах, широті доступних інтеграцій, а також, можливо, у використанні спеціалізованих, донавчених саме на аналізі коду моделей, які можуть мати деякі переваги у точності для специфічних завдань.

Однак, розроблений мною "CodeAnalyzer" також має певні переваги та унікальні особливості. По-перше, це його відкритість як прототипу, що дозволило мені глибоко вивчити та продемонструвати сам процес взаємодії з LLM для цілей аналізу коду, а також експериментувати з формуванням ефективних промптів. По-друге, було реалізувано можливість гнучкого налаштування промптів (хоча це відбувається на рівні коду, а не через користувацький інтерфейс на даному етапі), що дозволило отримувати деталізовані JSON-відповіді саме українською мовою та у потрібній мені структурі. По-третє, унікальною особливістю мого сервісу є поєднання функції аналізу окремого файлу з можливістю порівняння двох версій коду з отриманням не лише візуального diff, але й інтелектуального аналізу самих змін від штучного інтелекту, що, наскільки мені відомо, не так часто зустрічається в інших інструментах у такій явній формі. Крім того, мій сервіс надає можливість базового аналізу коду без необхідності обов'язкової реєстрації (хоча реєстрація потрібна для збереження історії), що може бути зручним для швидких перевірок або для освітніх цілей. Питання безпеки та конфіденційності коду при використанні зовнішнього OpenAI API, звичайно, є спільним викликом як для мого "CodeAnalyzer", так і для багатьох інших хмарних AI-сервісів, що надають подібні послуги.

Таким чином, підсумовуючи порівняльну оцінку, можемо сказати, що розроблений мною вебсервіс "CodeAnalyzer" у його поточній реалізації займає

нішу доступного веб-орієнтованого інструменту, який успішно демонструє значний потенціал використання сучасних великих мовних моделей, зокрема GPT-4o, для проведення глибокого семантичного аналізу програмного коду та інтелектуальної оцінки змін між його різними версіями. Він може бути особливо корисним для індивідуальних розробників, студентів, які вивчають програмування, або для невеликих команд як засіб для отримання "другої думки" від штучного інтелекту, для детального розбору складних або незрозумілих ділянок коду, або ж для освітніх цілей та самовдосконалення у написанні якісного коду.

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи було успішно досягнуто поставлену мету, яка полягала у дослідженні механізмів автоматизованого аналізу програмного коду з використанням методів та засобів штучного інтелекту, зокрема шляхом інтеграції з OpenAI API, а також розробці прототипу вебсервісу "CodeAnalyzer" що реалізує поставлену задачу. Розробка такого інструменту є актуальною відповіддю на зростаючі вимоги до якості, надійності та безпеки програмного забезпечення в умовах постійного ускладнення проєктів та скорочення циклів розробки.

Основними результатами, отриманими в процесі виконання роботи, є наступні:

1. Була спроєктована та реалізована архітектура веб-додатку "CodeAnalyzer". Серверна частина була розроблена на мові програмування PHP, що забезпечило необхідну гнучкість та доступність інструментів для веб-розробки. Було реалізовано ключові модулі, відповідальні за обробку запитів користувачів, управління даними (включаючи збереження історії аналізів та інформації про користувачів у файловій системі у форматі JSON) та взаємодію з зовнішніми сервісами.
2. Здійснено успішну інтеграцію з OpenAI API, зокрема з моделлю gpt-4o, для забезпечення інтелектуального аналізу програмного коду. Мною були розроблені та протестовані спеціалізовані промпти (запити) до мовної моделі, спрямовані на отримання структурованих та деталізованих відповідей українською мовою щодо виявлених проблем у коді, потенційних вразливостей, аспектів продуктивності, стилю кодування та супроводу, а також для аналізу змін між різними версіями коду.
3. Реалізовано основний функціонал вебсервісу "CodeAnalyzer", що включає:

- Можливість завантаження програмного коду користувачем двома способами: через завантаження файлу або шляхом прямої вставки тексту у веб-форму.
- Проведення комплексного аналізу наданого коду з виведенням загальної оцінки його якості, детального списку виявлених проблем із зазначенням їхнього місця розташування (номери рядків), описом, рівнем серйозності, пропозиціями щодо виправлення та, у багатьох випадках, прикладами коду.
- Функцію порівняння двох версій програмного коду, що включає візуалізацію відмінностей (diff view) та надання аналітичного звіту від штучного інтелекту щодо характеру, впливу та якості внесених змін.
- Систему реєстрації та автентифікації користувачів, а також механізм збереження та перегляду історії виконаних аналізів для авторизованих користувачів, що підвищує практичну цінність сервісу для постійного використання.

4. Було розроблено інтуїтивно зрозумілий та функціональний користувацький інтерфейс з використанням сучасних веб-технологій (HTML, CSS, Bootstrap, JavaScript). Для покращення візуального сприйняття програмного коду та результатів аналізу було інтегровано бібліотеку Prism.js для підсвічування синтаксису та іконки Feather Icons.

5. Проведено експериментальне дослідження розробленого вебсервісу. На прикладах тестових фрагментів коду різними мовами програмування (зокрема, PHP, JavaScript, Python) було продемонстровано здатність системи "CodeAnalyzer" ефективно виявляти широкий спектр проблем, включаючи логічні помилки, потенційні вразливості безпеки (наприклад, XSS), невикористовуваний код та дублювання логіки. Надані системою рекомендації та запропоновані виправлення були визнані релевантними та практично корисними. Функція порівняння версій коду також продемонструвала адекватні результати як у візуалізації відмінностей, так і в інтелектуальній оцінці змін. Тестування користувацького інтерфейсу

підтвердило його загальну зручність та інтуїтивність. Оцінка продуктивності показала прийнятний час відповіді для аналізу коду помірного обсягу, а також стабільну роботу системи, включаючи коректну обробку помилок та перехід на використання імітаційних даних у випадку проблем зі зв'язком з OpenAI API.

Практичне значення виконаної роботи полягає у створенні доступного веб-інструменту "CodeAnalyzer", який може допомогти розробникам різного рівня кваліфікації покращити якість свого програмного коду, скоротити час, що витрачається на його ручну перевірку та пошук помилок, а також сприяти вивченню найкращих практик програмування та стандартів кодування за допомогою інтелектуальних підказок та рекомендацій від системи, що базується на штучному інтелекті. Розроблений сервіс може бути корисним як для індивідуальних розробників та студентів, так і для невеликих команд у їхній повсякденній роботі.

Водночас, проведене дослідження та розробка виявили низку потенційних напрямків для подальшого розвитку та вдосконалення вебсервісу "CodeAnalyzer". До них можна віднести:

- Покращення алгоритму генерації відмінностей (diff) для функції порівняння коду, можливо, шляхом інтеграції більш досконалих алгоритмів, що враховують семантичні зміни, а не лише рядкові.
- Подальше вдосконалення промпт-інжинірингу для OpenAI API з метою отримання ще більш точних, деталізованих та контекстно-залежних результатів аналізу, особливо для складних випадків або специфічних мов програмування.
- Впровадження механізмів асинхронної обробки запитів для аналізу дуже великих файлів коду або цілих проєктів, щоб уникнути тривалого очікування користувачем та покращити загальну продуктивність системи.
- Перехід від файлової системи зберігання даних до використання повноцінної системи управління базами даних (СУБД), що підвищить масштабованість, надійність та швидкість роботи з даними користувачів та історією аналізів, особливо при зростанні кількості користувачів.

- Розширення функціональних можливостей сервісу, наприклад, додавання можливості аналізу цілих проєктів (через завантаження архіву або інтеграцію з Git-репозиторіями), впровадження системи кастомних правил аналізу, які користувач міг би налаштовувати самостійно, або розширення списку підтримуваних мов програмування.
- Розгляд можливостей для глибшої інтеграції з популярними середовищами розробки (IDE) або системами безперервної інтеграції та доставки (CI/CD) для забезпечення аналізу коду безпосередньо в робочому процесі розробника.
- Посилення аспектів безпеки, зокрема, шляхом впровадження більш надійних механізмів зберігання та управління API-ключами OpenAI, а також подальшого дослідження питань конфіденційності коду при використанні зовнішніх хмарних API.
- Проведення більш розширеного та формалізованого тестування зручності користувацького інтерфейсу з залученням більшої кількості реальних користувачів для збору детального зворотного зв'язку та виявлення додаткових напрямків для покращення.

Таким чином, у рамках даної кваліфікаційної роботи було успішно розв'язано поставлені завдання, розроблено функціональний прототип вебсервісу "CodeAnalyzer" для аналізу програмного коду з інтеграцією OpenAI API, продемонстровано його працездатність та визначено його потенціал для подальшого розвитку та вдосконалення. Виконана робота підтверджує перспективність використання сучасних технологій штучного інтелекту для створення інтелектуальних та доступних інструментів, спрямованих на підвищення якості та ефективності процесу розробки програмного забезпечення.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE).

[2] Chen L., Li H., Xia X., et al. A survey of AI for software engineering. – arXiv preprint arXiv:2301.02402, 2023. [3] Sommerville I. Software Engineering. – 10th ed. – Pearson, 2015.

[4] Pressman R.S., Maxim B.R. Software Engineering: A Practitioner's Approach. – 8th ed. – McGraw-Hill, 2014.

[5] Bacciu D., Brogi A., et al. Artificial Intelligence and Software Engineering: Status and Challenges. – IEEE Software, 2023.

[6] Vasilescu B., Yu Y., Wang H., et al. Quality and productivity outcomes relating to continuous integration in GitHub. – ESEC/FSE, 2015.

[7] Hindle A., Godfrey M.W., Holt R.C. What's hot and what's not: Windowed developer topic analysis. – ICSE, 2007.

[8] O'Neil M., Song Y., Tiwari M., et al. Code review comments: Language matters. – ICSE, 2020.

[9] Bird C., Rigby P.C., Barr E.T., et al. The promises and perils of mining GitHub // IEEE Software. – 2015.

[10] Allamanis M., Barr E.T., Devanbu P., Sutton C. A survey of machine learning for big code and naturalness. – ACM Comput. Surv., 2018.

[11] White M., Vendome C., Linares-Vásquez M., Poshyvanyk D. Toward deep learning software repositories. – MSR, 2015.

[12] Zhang J., Wang X., Sun J., et al. A comprehensive survey on neural code generation. – arXiv:2303.14755, 2023.

[13] Nguyen A.T., Nguyen T.N. Graph-based statistical language model for code. – ICSE, 2015.

[14] Feng Z., Guo D., Tang D., et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. – EMNLP, 2020.

[15] Wang Y., Jiang X., Wang S., et al. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. – NeurIPS, 2021.

[16] OpenAI. GPT-4 Technical Report. – 2023. – https://openai.com/research/gpt-4

[17] Chen M., Tworek J., Jun H., et al. Evaluating large language models trained on code. – arXiv:2107.03374, 2021.

[18] Svyatkovskiy A., Deng S., Fu S., Sundaresan N. IntelliCode Compose: Code generation using transformer. – ESEC/FSE, 2020.

[19] Microsoft. GitHub Copilot Technical Overview. – https://github.com/features/copilot

[20] Bubeck S., Chandrasekaran V., Eldan R., et al. Sparks of Artificial General Intelligence: Early experiments with GPT-4. – arXiv:2303.12712, 2023.

[21] Vaswani A., Shazeer N., Parmar N., et al. Attention is all you need. – NeurIPS, 2017.

[22] Bengio Y., Lecun Y., Hinton G. Deep learning. – Nature, 2015.

[23] Karpathy A. The unreasonable effectiveness of recurrent neural networks. – Blog, 2015.

[24] Chollet F. Deep Learning with Python. – Manning, 2017.

[25] Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press, 2016.

[26] Jurafsky D., Martin J.H. Speech and Language Processing. – 3rd ed. draft, 2023.

[27] Zhang Y., Yin H., et al. Learning semantic representations of code. – AAAI, 2019.

[28] Sadowski C., van Gogh J., Jaspan C., et al. Tricorder: Building a program analysis ecosystem. – ICSE, 2015.

ДОДАТКИ

ДОДАТОК А. Лістинг основних частин коду вебсервісу

У цьому додатку наведено лістинги ключових фрагментів PHP-коду, що демонструють основні аспекти функціонування вебсервісу "CodeAnalyzer".

Лістинг А.1 - Головний маршрутизатор (фрагмент htdocs/index.php)

```
<?php
// Start the session to maintain user state
session_start();

// Define base constants
define('ROOT_PATH', __DIR__);
define('UPLOAD_DIR', ROOT_PATH . '/uploads');
define('MAX_FILE_SIZE', 16 * 1024 * 1024); // 16MB max upload size

// Create uploads directory if it doesn't exist
if (!file_exists(UPLOAD_DIR)) {
    mkdir(UPLOAD_DIR, 0755, true);
}

// Include utility functions and database
require_once 'includes/functions.php';
require_once 'includes/db.php';

// Handle routing
$page = isset($_GET['page']) ? $_GET['page'] : 'home';

// Include header
include 'includes/header.php';
```

```
// Include the appropriate page
switch ($page) {
    // Сторінки аналізу коду
    case 'analyze':
        include 'pages/analyze.php';
        break;
    case 'compare':
        include 'pages/compare.php';
        break;
    case 'comparison_results':
        include 'pages/comparison_results.php';
        break;
    case 'api_status':
        include 'pages/api_status.php';
        break;
    case 'test_api':
        include 'pages/test_api.php';
        break;

    // Сторінки авторизації
    case 'login':
        include 'pages/login.php';
        break;
    case 'register':
        include 'pages/register.php';
        break;
    case 'logout':
        include 'pages/logout.php';
        break;
    case 'profile':
        include 'pages/profile.php';
```

```
            break;
        case 'forgot_password':
            include 'pages/forgot_password.php';
            break;
        case 'my_analyses':
            include 'pages/my_analyses.php';
            break;

        // Домашня сторінка за замовчуванням
        default:
            include 'pages/home.php';
            break;
    }

    // Include footer
    include 'includes/footer.php';
    ?>
```

Джерело: Розроблено автором на основі htdocs/index.php

Лістинг А.2 - Функція аналізу коду через OpenAI (фрагмент функції php_analyze_code з htdocs/utils/php_code_analyzer.php)

```
<?php
function php_analyze_code($code, $language) {
    $api_key = 'sk-proj-
LR2TojpqnglMKdWg4wmNqF7WE1cssPwM8ourlkkW26B0YJISLo5Sbwu7JEV-
lVklWSKiZHoeJDyT3BlbkFJG3ORtyXlvGIS6sc6971qY5p3dWREwo3icbeN-
VKrRpkPm3W4F9TSzP8Di5ewYiPBQAIKFJdK2oA';

    if (empty($api_key) ||
        (strpos($api_key, 'sk-') !== 0
    && strpos($api_key, 'sk-proj-') !== 0)) {
```

```
        error_log("Недійсний або відсутній ключ API. Використовується мок-аналіз.");
        return
    php_generate_mock_analysis($code, $language);
    }

    try {
        $prompt = "
        Відповідай українською. Будь ласка,
    проаналізуйте наступний код на мові {$language} для:
        1. Потенційних помилок та багів
        2. Вразливостей безпеки
        3. Проблем з продуктивністю
        4. Порушень стилю коду та найкращих
    практик
        5. Проблем із супроводом

        Для кожної знайденої проблеми надайте:
        - Номер(и) рядків, де з'явилася
    проблема
        - Короткий опис проблеми
        - Детальну пропоновану виправлення або
    вдосконалення З ПРИКЛАДОМ КОДУ, що показує, як реалізувати виправлення
        - Рівень серйозності (критичний,
    високий, середній, низький)

        Також надайте загальні рекомендації для
    вдосконалення коду з принаймні 3 детальними пропозиціями. Кожна пропозиція
    повинна включати:
```

- Чітке пояснення рекомендації
- Конкретний приклад коду, що показує, як це реалізувати
- Пояснення, чому це вдосконалення важливе

Також надайте загальну оцінку якості від 0-100.

Відповідь має бути українською мовою у форматі JSON об'єкта
з наступною структурою:

{

\"issues\": [

{

\"line_start\": integer,

\"line_end\": integer,

\"description\": \"string\",

\"suggestion\": \"string\",

\"code_example\": \"string\",

\"severity\": \"string\"

}

],

```
\"detailed_recommendations\": [
                {

   \"title\": \"string\",

   \"explanation\": \"string\",

   \"code_example\": \"string\",

   \"benefit\": \"string\"
                }
            ],

\"suggestions\": [

\"string\"
            ],

\"score\": integer
        }

        Ось код:
        ```{$language}
        {$code}
        ```
        ";

        error_log("Перевірка доступності OpenAI API з ключем: " . sub-
str($api_key, 0, 10) . "...");
        $test_response =
php_call_openai_api("Розкажи короткий жарт", $api_key);
```

```
        if (empty($test_response)) {
            error_log("Тест API не вдався. Використовується мок-аналіз.");

file_put_contents("data/api_status.json", json_encode([

"status" => "error",

"timestamp" => time(),

"message" => "Тест API не вдався. Перевірте свій ключ API."
            ]));
            return
php_generate_mock_analysis($code, $language);
        } else {

file_put_contents("data/api_status.json", json_encode([

"status" => "success",

"timestamp" => time(),

"message" => "З'єднання з API успішне."
            ]));
        }

        error_log("Аналіз коду за допомогою OpenAI API...");
        $response =
php_call_openai_api($prompt, $api_key);

        if (empty($response)) {
```

```
error_log("Не вдалося отримати аналіз від API. Використовується мок-аналіз.");
            return
php_generate_mock_analysis($code, $language);
        }

        if (!isset($response['issues'])) {
            $response['issues'] = [];
        }
        if (!isset($response['detailed_recommendations']))
{

$response['detailed_recommendations'] = [];
        }
        if
(!isset($response['suggestions'])) {

$response['suggestions'] = [];
        }
        if (!isset($response['score'])) {
            $response['score']
= 0;
        }

        return $response;
    } catch (Exception $e) {
        error_log("Помилка аналізу коду: " . $e->getMessage());

        return [
            'issues' => [],
```

```
'detailed_recommendations' => [],
            'suggestions' =>
["Помилка аналізу коду. Будь ласка, спробуйте ще раз."],
            'score' => 0,
            'error' =>
$e->getMessage()
        ];
    }
}

function php_call_openai_api($prompt, $api_key) {
    $url =
'https://api.openai.com/v1/chat/completions';

    $data = [
        'model' => 'gpt-4o',
        'messages' => [
            [

'role' => 'system',

'content' => 'You are an expert code analyzer. Provide detailed, accurate
code analysis focusing on technical issues, not stylistic preferences unless
significant.'
            ],
            [

'role' => 'user',

'content' => $prompt
```

```
            ]
        ],
        'temperature' => 0.2
    ];

    $headers = [
        'Content-Type: application/json',
        'Authorization: Bearer ' .
$api_key
    ];

    error_log('Надсилання запиту до OpenAI API...');

    $options = [
        CURLOPT_URL => $url,
        CURLOPT_RETURNTRANSFER =>
true,
        CURLOPT_POST => true,
        CURLOPT_POSTFIELDS =>
json_encode($data),
        CURLOPT_HTTPHEADER =>
$headers,
        CURLOPT_TIMEOUT => 60,
        CURLOPT_SSL_VERIFYPEER => false,
        CURLOPT_SSL_VERIFYHOST => 0,
        CURLOPT_VERBOSE => true,
        CURLOPT_PROXY => null,
        CURLOPT_FOLLOWLOCATION => true
    ];

    $curl = curl_init();
```

```
    curl_setopt_array($curl, $options);
    $response = curl_exec($curl);
    $info = curl_getinfo($curl);
    $err = curl_error($curl);
    curl_close($curl);


file_put_contents("data/debug_raw.txt", $response);

    error_log('Інформація про відповідь API: ' .
print_r($info, true));

    if ($err) {
        error_log('Помилка cURL: ' . $err);
        return null;
    }

    $decoded = json_decode($response, true);

    error_log('Відповідь API: ' . print_r($decoded,
true));

    if
(isset($decoded['choices'][0]['message']['content'])) {
        $content =
$decoded['choices'][0]['message']['content'];

        if
(preg_match('/```json\s*(.*?)\s*```/is', $content, $matches)) {
            $json_string =
$matches[1];
```

```
            $json_string =
json_decode($json_string, true);
        } else {
            $json_string =
$content;
        }

        return $json_string;
    }

    error_log('Неправильний формат відповіді API: ' .
print_r($decoded, true));
    return null;
}

function php_generate_mock_analysis($code, $language) {
    $lines = explode("\n", $code);
    $line_count = count($lines);

    $issues = [];

    foreach ($lines as $i => $line) {
        if ($i > 0
&& $i < $line_count - 1) {
            if ($i % 10 === 0) {
                $issues[] = [

'line_start' => $i + 1,

'line_end' => $i + 1,
```

```
        'description' =>
        "Потенційна проблема виявлена в рядку " . ($i+1),
                    'suggestion' =>
        "Розгляньте можливість реструктуризації цієї частини коду для кращої
        читабельності",
                    'code_example' => "// Приклад покращеної структури коду:\n//
Розділіть довгі рядки коду\n// Додайте
        коментарі\n// Використовуйте
        осмислені імена змінних",
                    'severity' => "low"
                ];
            }

            if (strpos($line,
        'TODO') !== false) {

        $issues[] = [

        'line_start' => $i + 1,

        'line_end' => $i + 1,

        'description' => "Знайдено коментар TODO",

        'suggestion' => "Реалізуйте позначену функціональність
        або видаліть коментар TODO",

        'code_example' => "// Замість використання TODO
        коментарів, краще:\n// 1. Створити тікет в системі відстеження
        завдань\n// 2.
```

Додати посилання на тікет у коментарі\n// 3. Або повністю реалізувати функцію зараз",

'severity' => "low"
];
}

if (preg_match('/console\.log|print\(|println|System\.out\.print/', $line)) {

$issues[] = [

'line_start' => $i + 1,

'line_end' => $i + 1,

'description' => "Виявлено відлагоджувальний вираз",

'suggestion' => "Видаліть відлагоджувальні вирази перед публікацією коду в продакшн",

'code_example' => "// Замість:\nconsole.log('Debug value:', value);\n\n// Використовуйте систему логування:\nlogger.debug('Debug value:', value);\n\n// Або повністю видаліть, якщо це не потрібно в продакшні",

'severity' => "medium"
];
}
}

```
    }

    $detailed_recommendations = [
        [
            'title' =>
"Додайте документацію",
            'explanation' => "Додайте детальну документацію до
функцій та класів для покращення супроводу коду та полегшення роботи з
ним для інших розробників.",
            'code_example' =>
"/**\n * Функція для обробки даних користувача\n *\n * @param {Object}
userData - Дані
користувача для обробки\n * @param {string} userData.name - Ім'я
користувача\n * @param {number} userData.age - Вік
користувача\n * @returns {Object} Оброблені
дані користувача\n */\nfunction processUserData(userData) {\n    //
Обробка даних\n    return processedData;\n}",
            'benefit' =>
"Якісна документація скорочує час на розуміння коду, полегшує підтримку
і зменшує кількість помилок при використанні функцій іншими
розробниками."
        ],
        [
            'title' =>
"Використовуйте модульну структуру",
            'explanation' =>
"Розділіть великі блоки коду на модулі та функції з єдиною
відповідальністю для покращення читабельності та супроводу.",
            'code_example' => "//
Замість великої функції, що робить все:\nfunction processUserData(userData)
{\n    // Валідація\n    validateUserData(userData);\n    \n    //
```

Обробка\n   const processedData = transformUserData(userData);\n   \n   // Збереження\n   saveUserData(processedData);\n   \n   return processedData;\n}\n\n// Окремі функції для кожної відповідальності:\nfunction validateUserData(userData) { /* ... */ }\nfunction transformUserData(userData) { /* ... */ }\nfunction saveUserData(userData) { /* ... */ }",

'benefit' =>

"Модульний код легше тестувати, підтримувати й розширювати. Кожна функція має чітку відповідальність, що робить код більш зрозумілим і менш схильним до помилок."

],

[

'title' =>

"Обробляйте помилки",

'explanation' =>

"Додайте коректну обробку помилок для підвищення надійності та стійкості додатку.",

'code_example' => "// Замість:\nfunction fetchData(url) {\n   const data = makeApiCall(url);\n   return processData(data);\n}\n\n// Використовуйте обробку помилок:\nfunction fetchData(url) {\n   try {\n   const data = makeApiCall(url);\n   return processData(data);\n   } catch (error) {\n   console.error('Error fetching data:', error);\n   // Логування помилки\n   logError(error);\n   // Повернення безпечного значення за замовчуванням\n   return { error: true, message: 'Failed to fetch data' };\n   }\n}",

'benefit' =>

"Правильна обробка помилок запобігає збоям додатку, покращує досвід користувача та спрощує налагодження проблем у продакшн-середовищі."

]

];

```
    $suggestions = [
        "Додайте коментарі до складних
частин коду для покращення розуміння",
        "Використовуйте узгоджений стиль
кодування у всьому проєкті",
        "Розгляньте можливість
додавання автоматичних тестів для цього коду"
    ];

    $score = max(50, 100 -
(count($issues) * 5));

    return [
        'issues' => $issues,
        'detailed_recommendations' =>
$detailed_recommendations,
        'suggestions' => $suggestions,
        'score' => $score
    ];
}

function php_compare_code($original_code, $new_code,
$language) {
    $diff_result =
php_generate_diff($original_code, $new_code);

    $api_key =
```

```
    'sk-proj-
LR2TojpqnglMKdWg4wmNqF7WE1cssPwM8ourlkkW26B0YJISLo5Sbwu7JEV-
lVklWSKiZHoeJDyT3BlbkFJG3ORtyXlvGIS6sc6971qY5p3dWREwo3icbeN-
VKrRpkPm3W4F9TSzP8Di5ewYiPBQAIKFJdK2oA';

        if (!empty($api_key) &&
            (strpos($api_key, 'sk-') === 0 ||
    strpos($api_key, 'sk-proj-') === 0)) {
            $analysis =
    php_analyze_changes_with_ai($original_code, $new_code, $language,
    $diff_result);
        } else {
            error_log("Недійсний або відсутній ключ API для порівняння.
Використовується базовий аналіз.");
            $analysis =
    php_analyze_changes_basic($original_code, $new_code, $diff_result);
        }

        $metrics = php_calculate_metrics($diff_result);

        return [
            'diff' => $diff_result,
            'analysis' => $analysis,
            'metrics' => $metrics
        ];
    }

    function php_generate_diff($original_code, $new_code) {
        $original_lines = explode("\n",
    $original_code);
        $new_lines = explode("\n",
```

```
$new_code);

    $diff = [];
    $maxLen = max(count($original_lines),
count($new_lines));

    for ($i = 0; $i < $maxLen; $i++) {
        $original_line =
isset($original_lines[$i]) ? $original_lines[$i] : null;
        $new_line = isset($new_lines[$i])
? $new_lines[$i] : null;

        if ($original_line === null) {
            $diff[] = [
                'type' => 'added',
                'original_line' => null,

'new_line' => $i + 1,

'content' => $new_line
            ];
        } else if ($new_line === null) {
            $diff[] = [

'type' => 'removed',

'original_line' => $i + 1,

'new_line' => null,

'content' => $original_line
```

```
            ];
        } else if ($original_line !==
$new_line) {
            $diff[] = [

'type' => 'changed',

'original_line' => $i + 1,

'new_line' => $i + 1,

'original_content' => $original_line,

'new_content' => $new_line
            ];
        } else {
            $diff[] = [
                'type' => 'unchanged',
                'original_line' => $i + 1,

'new_line' => $i + 1,

'content' => $original_line
            ];
        }
    }

    return $diff;
}

function php_analyze_changes_with_ai($original_code,
```

```
$new_code, $language, $diff_result) {
    $api_key =
'sk-proj-
LR2TojpqnglMKdWg4wmNqF7WE1cssPwM8ourlkkW26B0YJISLo5Sbwu7JEV-
lVklWSKiZHoeJDyT3BlbkFJG3ORtyXlvGIS6sc6971qY5p3dWREwo3icbeN-
VKrRpkPm3W4F9TSzP8Di5ewYiPBQAIKFJdK2oA';

    $formatted_diff =
php_format_diff_for_prompt($diff_result);

    $prompt = "

Проаналізуйте зміни коду на мові {$language} між оригінальною та новою
версією.



Відповідай українською.
    Надайте
детальну оцінку змін, включаючи:
    1. Чи є
зміни покращенням кодової бази?
    2. Чи
вирішують зміни конкретні проблеми або вразливості?
    3. Чи
дотримуються зміни найкращих практик програмування?
    4. Чи є
які-небудь потенційні проблеми або покращення для впроваджених змін?



Відповідь має бути у форматі JSON об'єкта з наступною структурою:
    {
```

```
        \"impact_assessment\":
\"string\",
        \"improvements\":
[\"string\"],
        \"concerns\": [\"string\"],
        \"suggestions\": [\"string\"],
        \"quality_change\": integer
    }

    Ось
оригінальний код:
    ```{$language}
    {$original_code}
    ```

    Ось
новий код:
    ```{$language}
    {$new_code}
    ```

    Ось diff між версіями:
    {$formatted_diff}
    ";

    $response = php_call_openai_api($prompt,
$api_key);

    if (empty($response)) {
        return
php_analyze_changes_basic($original_code, $new_code, $diff_result);
```

```
    }

    if (!isset($response['impact_assessment'])) {
        $response['impact_assessment'] =
"Неможливо визначити вплив змін";
    }
    if (!isset($response['improvements'])) {
        $response['improvements'] = [];
    }
    if (!isset($response['concerns'])) {
        $response['concerns'] = [];
    }
    if (!isset($response['suggestions'])) {
        $response['suggestions'] = [];
    }
    if (!isset($response['quality_change'])) {
        $response['quality_change'] = 0;
    }

    return $response;
}

function php_analyze_changes_basic($original_code, $new_code,
$diff_result) {
    $added_lines = 0;
    $removed_lines = 0;
    $changed_lines = 0;

    foreach ($diff_result as $diff) {
        if ($diff['type'] === 'added') {
            $added_lines++;
```

```
        } else if ($diff['type'] ===
    'removed') {
            $removed_lines++;
        } else if ($diff['type'] ===
    'changed') {
            $changed_lines++;
        }
    }

    $impact_assessment = "Код було змінено з
    {$removed_lines} видаленими рядками, {$added_lines} доданими рядками
та
    {$changed_lines} зміненими рядками.";

    $improvements = ["Код було оновлено, можливо,
    для покращення функціональності або виправлення помилок."];
    $concerns = [];

    if ($removed_lines > $added_lines
    * 2) {
        $concerns[] =
    "Значна кількість коду була видалена. Переконайтеся, що це не призвело
до
    втрати функціональності.";
    }

    if ($added_lines > $removed_lines
    * 2) {
        $concerns[] =
    "Додано значну кількість нового коду. Переконайтеся, що він відповідає
    стандартам та не вводить нові помилки.";
```

```
    }

    $quality_change = 0;

    return [
        'impact_assessment' =>
$impact_assessment,
        'improvements' =>
$improvements,
        'concerns' => $concerns,
        'suggestions' => ["Рекомендуємо
провести ретельне тестування змін для забезпечення якості."],
        'quality_change' => $quality_change
    ];
}

function php_calculate_metrics($diff_result) {
    $added_lines = 0;
    $removed_lines = 0;
    $changed_lines = 0;
    $unchanged_lines = 0;

    foreach ($diff_result as $diff) {
        switch ($diff['type']) {
            case 'added':

$added_lines++;

break;
            case 'removed':
```

```
$removed_lines++;

break;
            case 'changed':

$changed_lines++;

break;
            case 'unchanged':

$unchanged_lines++;

break;
        }
    }

    $total_lines = $added_lines + $removed_lines +
$changed_lines + $unchanged_lines;
    $change_percentage = $total_lines > 0 ?
round((($added_lines + $removed_lines + $changed_lines) / $total_lines) * 100,
2) : 0;

    return [
        'added_lines' => $added_lines,
        'removed_lines' =>
$removed_lines,
        'changed_lines' =>
$changed_lines,
        'unchanged_lines' =>
$unchanged_lines,
        'total_lines' => $total_lines,
```

```
        'change_percentage' =>
$change_percentage
    ];
}

function php_format_diff_for_prompt($diff_result) {
    $formatted_diff = "Зміни в коді:\n";

    foreach ($diff_result as $diff) {
        switch ($diff['type']) {
            case 'added':

$formatted_diff .= "+ Додано рядок {$diff['new_line']}:
{$diff['content']}\n";

break;
            case 'removed':

$formatted_diff .= "- Видалено рядок {$diff['original_line']}:
{$diff['content']}\n";

break;
            case 'changed':

$formatted_diff .= "~ Змінено рядок {$diff['original_line']} на
{$diff['new_line']}:\n";

$formatted_diff .= "          Було: {$diff['original_content']}\n";

$formatted_diff .= "          Стало: {$diff['new_content']}\n";
                break;
```

```
            }
        }

        return $formatted_diff;
    }
    ?>
```

Джерело: Розроблено автором на основі htdocs/utils/php_code_analyzer.php

Лістинг А.3 - Обробка завантаження файлу (фрагмент htdocs/upload.php)

```
    <?php
    // Start the session to maintain user state
    session_start();

    // Define base constants
    define('ROOT_PATH', __DIR__);
    define('UPLOAD_DIR', ROOT_PATH . '/uploads');
    define('MAX_FILE_SIZE', 16 * 1024 * 1024); // 16MB max upload size

    // Create uploads directory if it doesn't exist
    if (!file_exists(UPLOAD_DIR)) {
        mkdir(UPLOAD_DIR, 0755, true);
    }

    // Include utility functions
    require_once 'includes/functions.php';

    // Check if form is submitted
    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        // Check if code was uploaded via file
        if (isset($_FILES['code_file']) && $_FILES['code_file']['error'] === UPLOAD_ERR_OK) {
```

```
            $file = $_FILES['code_file'];

            // Check if file is empty
            if ($file['size'] === 0) {
                set_flash_message('Обраний файл порожній', 'warning');
                header('Location: index.php');
                exit;
            }

            // Check if file size is within limits
            if ($file['size'] > MAX_FILE_SIZE) {
                set_flash_message('Розмір файлу перевищує максимальне обмеження
(16MB)', 'warning');
                header('Location: index.php');
                exit;
            }

            // Check if file type is allowed
            $filename = $file['name'];
            if (!is_allowed_file($filename)) {
                set_flash_message('Тип файлу не підтримується', 'warning');
                header('Location: index.php');
                exit;
            }

            // Create a unique ID for the file
            $file_id = uniqid();
            $language = detect_language($filename);

            // Save the file with a unique name
```

```
        $file_extension = strtolower(pathinfo($filename, PATHINFO_EXTEN-
SION));
        $file_path = UPLOAD_DIR . "/{$file_id}.{$file_extension}";

        if (move_uploaded_file($file['tmp_name'], $file_path)) {
            // Store file information in session
            $_SESSION['file_id'] = $file_id;
            $_SESSION['filename'] = $filename;
            $_SESSION['language'] = $language;

            // Redirect to analysis page
            header('Location: index.php?page=analyze');
            exit;
        } else {
            set_flash_message('Не вдалося завантажити файл', 'danger');
            header('Location: index.php');
            exit;
        }
    }
    // Check if code was provided via text input
    elseif (isset($_POST['code_text']) && !empty($_POST['code_text'])) {
        $code = $_POST['code_text'];
        $language = $_POST['language'] ?? 'python';

        // Create a unique ID for the file
        $file_id = uniqid();

        // Save the code to a file
        $file_extension = $language;
        if ($language === 'javascript') $file_extension = 'js';
        elseif ($language === 'typescript') $file_extension = 'ts';
```

```
        elseif ($language === 'csharp') $file_extension = 'cs';

        $file_path = UPLOAD_DIR . "/{$file_id}.{$file_extension}";
        $filename = "code.{$file_extension}";

        if (file_put_contents($file_path, $code) !== false) {
            // Store file information in session
            $_SESSION['file_id'] = $file_id;
            $_SESSION['filename'] = $filename;
            $_SESSION['language'] = $language;

            // Redirect to analysis page
            header('Location: index.php?page=analyze');
            exit;
        } else {
            set_flash_message('Не вдалося зберегти код', 'danger');
            header('Location: index.php');
            exit;
        }
    }
    else {
        set_flash_message('Код не надано для аналізу', 'warning');
        header('Location: index.php');
        exit;
    }
} else {
    // If not a POST request, redirect to home page
    header('Location: index.php');
    exit;
}
?>
```

Джерело: Розроблено автором на основі htdocs/upload.php

Лістинг А.4 - Функція збереження аналізу в історію (фрагмент функції add_analysis з htdocs/includes/functions.php)

```
/**
 * Додати новий аналіз в історію
 *
 * @param int $user_id ID користувача
 * @param string $filename Ім'я файлу або заголовок аналізу
 * @param string $language Мова програмування
 * @param string $code Код, що аналізується
 * @param array $analysis_results Результати аналізу
 * @return array|false Дані нового аналізу або false у випадку помилки
 */
function add_analysis($user_id, $filename, $language, $code, $analysis_results)
{
    $analyses = get_all_analyses();

    // Генеруємо унікальний ID
    $id = uniqid();

    // Обрізаємо занадто довгий код для збереження
    $code_preview = substr($code, 0, 1000);
    if (strlen($code) > 1000) {
        $code_preview .= '...';
    }

    // Створюємо новий запис аналізу
    $new_analysis = [
        'id' => $id,
        'user_id' => $user_id,
```

```
            'filename' => $filename,
            'language' => $language,
            'code_preview' => $code_preview,
            'score' => $analysis_results['score'] ?? 0,
            'issues_count' => count($analysis_results['issues'] ?? []),
            'created_at' => date('Y-m-d H:i:s')
        ];

        // Зберігаємо повний аналіз в окремому файлі
        $analysis_dir = 'data/analysis/' . $id;
        if (!is_dir('data/analysis/')) {
            mkdir('data/analysis/', 0755, true);
        }
        if (!is_dir($analysis_dir)) {
            mkdir($analysis_dir, 0755, true);
        }

        file_put_contents($analysis_dir . '/code.txt', $code);
        file_put_contents($analysis_dir . '/results.json', json_encode($analysis_results));

        // Додаємо в загальний масив
        $analyses[] = $new_analysis;

        // Зберігаємо зміни
        if (save_analyses($analyses)) {
            return $new_analysis;
        }

        return false;
    }
```

Джерело: Розроблено автором на основі htdocs/includes/functions.php

Лістинг А.5 – основна логіка сторінка з результатом аналіза

```
<?php
if (isset($_GET['id']) && !empty($_GET['id'])) {
    $load_from_history = true;

    $code = '';
    $filename = '';
    $language = '';
} else {
    $load_from_history = false;

    // Check if file information is in session
    if (!isset($_SESSION['file_id']) || !isset($_SESSION['filename']) ||
!isset($_SESSION['language'])) {
        echo '<script>
            alert("Немає завантаженого файлу");
            window.location.href = "index.php";
        </script>';
        exit;
    }

    // Get file information from session
    $file_id = $_SESSION['file_id'];
    $filename = $_SESSION['filename'];
    $language = $_SESSION['language'];

    // Determine file path
    $file_extension = strtolower(pathinfo($filename,
PATHINFO_EXTENSION));
```

```
        $file_path = 'uploads' . "/{$file_id}.{$file_extension}";

        // Check if file exists
        if (!file_exists($file_path)) {
            set_flash_message('Файл не знайдено', 'warning');
            header('Location: index.php');
            exit;
        }

        // Read the file content
        $code = file_get_contents($file_path);
    }

    if (isset($_GET['id']) && !empty($_GET['id'])) {
        $analysis_data = get_analysis_by_id($_GET['id']);

        if (!$analysis_data) {
            set_flash_message('Аналіз не знайдено', 'warning');
            header('Location: index.php');
            exit;
        }

        if (isset($_SESSION['user_id']) && $analysis_data['user_id'] !=
$_SESSION['user_id']) {
            set_flash_message('У вас немає доступу до цього аналізу', 'danger');
            header('Location: index.php');
            exit;
        }

        $code = $analysis_data['code'];
        $language = $analysis_data['language'];
```

```
            $filename = $analysis_data['filename'];
            $analysis_results = $analysis_data['results'];

            $_SESSION['filename'] = $filename;
            $_SESSION['language'] = $language;
        } else {
            $analysis_results = analyze_code($code, $language);

            if (isset($_SESSION['user_id'])) {
                $saved_analysis = add_analysis(
                    $_SESSION['user_id'],
                    $filename,
                    $language,
                    $code,
                    $analysis_results
                );
            }
        }

        // Extract analysis components
        $issues = $analysis_results['issues'] ?? [];
        $detailed_recommendations = $analysis_results['detailed_recommendations'] ??
[];
        $suggestions = $analysis_results['suggestions'] ?? [];
        $score = $analysis_results['score'] ?? 0;

        // Create line numbers for display
        $line_numbers = get_line_numbers($code);
        ?>

        <section class="analysis-header">
```

```
        <div class="container">
            <div class="d-flex justify-content-between align-items-center">
                <h1>
                    <i data-feather="file-text" class="me-2"></i>
                    Результати аналізу: <?= htmlspecialchars($filename) ?>
                </h1>
                <a href="index.php" class="btn btn-outline-primary">
                    <i data-feather="upload" class="me-2"></i> Аналізувати новий код
                </a>
            </div>

            <div class="analysis-summary mt-4">
                <div class="row">
                    <div class="col-md-4">
                        <div class="card h-100">
                            <div class="card-body">
                                <h5 class="card-title">Оцінка якості</h5>
                                <div class="score-circle <?= $score >= 80 ? 'high-score' :
($score >= 60 ? 'medium-score' : 'low-score') ?>">
                                    <span class="score-value"><?= $score ?></span>
                                </div>
                                <div class="score-label text-center mt-2">
                                    <?php if ($score >= 80): ?>
                                        <span class="badge bg-success">Добре</span>
                                    <?php elseif ($score >= 60): ?>
                                        <span class="badge bg-warning text-dark">Потребує по-
кращення</span>
                                    <?php else: ?>
                                        <span class="badge bg-danger">Критичні про-
блеми</span>
                                    <?php endif; ?>
```

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

    <div class="col-md-4">
        <div class="card h-100">
            <div class="card-body">
                <h5 class="card-title">Знайдені проблеми</h5>
                <div class="issues-summary">
                    <?php
                    $critical_count = 0;
                    $high_count = 0;
                    $medium_count = 0;
                    $low_count = 0;

                    foreach ($issues as $issue) {
                        $severity = strtolower($issue['severity']);
                        if ($severity === 'critical') $critical_count++;
                        elseif ($severity === 'high') $high_count++;
                        elseif ($severity === 'medium') $medium_count++;
                        elseif ($severity === 'low') $low_count++;
                    }
                    ?>
                    <div class="d-flex justify-content-between mb-2">
                        <span>Критичні:</span>
                        <span class="badge bg-danger"><?= $critical_count
?></span>
                    </div>
                    <div class="d-flex justify-content-between mb-2">
                        <span>Високі:</span>
```

```
                            <span class="badge bg-danger"><?= $high_count
?></span>
                        </div>
                        <div class="d-flex justify-content-between mb-2">
                            <span>Середні:</span>
                            <span class="badge bg-warning text-dark"><?=
$medium_count ?></span>
                        </div>
                        <div class="d-flex justify-content-between">
                            <span>Низькі:</span>
                            <span class="badge bg-info"><?= $low_count ?></span>
                        </div>
                    </div>
                </div>
            </div>

            <div class="col-md-4">
                <div class="card h-100">
                    <div class="card-body">
                        <h5 class="card-title">Мова програмування</h5>
                        <div class="language-info d-flex align-items-center">
                            <div class="language-icon me-3">
                                <i data-feather="code" style="width: 32px; height:
32px;"></i>
                            </div>
                            <div>
                                <h3 class="mb-0 language-name"><?= ucfirst($language)
?></h3>
                                <p class="text-muted mb-0"><?= count($line_numbers)
?> lines</p>
```

```
                    </div>
                </div>
            </div>
        </div>
    </div>
</section>

<section class="analysis-content">
    <div class="container">
        <div class="row">
            <div class="col-lg-8">
                <div class="card code-card">
                    <div class="card-header d-flex justify-content-between align-items-center">
                        <h3 class="mb-0">
                            <i data-feather="code" class="me-2"></i>
                            Вихідний код
                        </h3>
                        <div class="code-actions">
                            <button class="btn btn-sm btn-outline-secondary" id="toggle-line-numbers">
                                <i data-feather="list" class="me-1"></i> Переключити номери рядків
                            </button>
                            <button class="btn btn-sm btn-outline-secondary" id="copy-code">
                                <i data-feather="copy" class="me-1"></i> Копіювати код
                            </button>
```

```
                    </div>
                </div>
                <div class="card-body p-0">
                    <div class="code-container">
                        <pre class="line-numbers"><code class="language-<?= $language ?>"><?= htmlspecialchars($code) ?></code></pre>
                    </div>
                </div>
            </div>
        </div>

        <div class="col-lg-4">
            <div class="issues-container">
                <div class="card">
                    <div class="card-header">
                        <h3 class="mb-0">
                            <i data-feather="alert-circle" class="me-2"></i>
                            Помилки та поради
                        </h3>
                    </div>
                    <div class="card-body p-0">
                        <div class="issues-list">
                            <?php if (count($issues) > 0): ?>
                                <?php foreach ($issues as $issue): ?>
                                    <div class="issue-item" data-line-start="<?= $issue['line_start'] ?>" data-line-end="<?= $issue['line_end'] ?>">
                                        <div class="issue-header">
                                            <div class="d-flex justify-content-between align-items-center">
```

```
                                    <span class="issue-location">Line <?=
$issue['line_start'] ?><?= $issue['line_end'] != $issue['line_start'] ? '-' .
$issue['line_end'] : '' ?></span>
                                    <span class="issue-severity
                                        <?php
                                        $severity = strtolower($issue['severity']);
                                        if ($severity === 'critical' || $severity ===
'high') echo 'badge bg-danger';
                                        elseif ($severity === 'medium') echo 'badge
bg-warning text-dark';
                                        else echo 'badge bg-info';
                                        ?>">
                                        <?= ucfirst($issue['severity']) ?>
                                    </span>
                                </div>
                            </div>
                            <div class="issue-body">
                                <p class="issue-description"><?=
htmlspecialchars($issue['description']) ?></p>
                                <div class="issue-suggestion">
                                    <h5>Рекомендації:</h5>
                                    <p><?= htmlspecialchars($issue['suggestion'])
?></p>
                                </div>
                                <?php if (isset($issue['code_example']) &&
!empty($issue['code_example'])): ?>
                                <div class="issue-code-example mt-2">
                                    <h5>Приклад коду:</h5>
                                    <pre><code class="language-<?= $language
?>"><?= htmlspecialchars($issue['code_example']) ?></code></pre>
                                </div>
```

```
                                    <?php endif; ?>
                                </div>
                            </div>
                        <?php endforeach; ?>
                    <?php else: ?>
                        <div class="no-issues">
                            <div class="text-center p-4">
                                <i data-feather="check-circle" style="width: 48px;
height: 48px;" class="text-success mb-3"></i>
                                <h4>No issues found!</h4>
                                <p>Your code looks good. Well done!</p>
                            </div>
                        </div>
                    <?php endif; ?>
                </div>
            </div>
        </div>

        <?php if (count($detailed_recommendations) > 0): ?>
            <div class="card mt-4">
                <div class="card-header">
                    <h3 class="mb-0">
                        <i data-feather="star" class="me-2"></i>
                        Рекомендації щодо коду
                    </h3>
                </div>
                <div class="card-body p-0">
                    <div class="accordion"
id="detailedRecommendationsAccordion">
                        <?php foreach ($detailed_recommendations as $index =>
$recommendation): ?>
```

```
                                <div class="accordion-item">
                                    <h2 class="accordion-header" id="heading<?=
$index ?>">
                                        <button class="accordion-button <?= $index !== 0
? 'collapsed' : '' ?>" type="button"
                                            data-bs-toggle="collapse" data-bs-
target="#collapse<?= $index ?>"
                                            aria-expanded="<?= $index === 0 ? 'true' :
'false' ?>" aria-controls="collapse<?= $index ?>">
                                            <?= htmlspecialchars($recommendation['title'])
?>
                                        </button>
                                    </h2>
                                    <div id="collapse<?= $index ?>" class="accordion-
collapse collapse <?= $index === 0 ? 'show' : '' ?>"
                                        aria-labelledby="heading<?= $index ?>" data-bs-
parent="#detailedRecommendationsAccordion">
                                        <div class="accordion-body">
                                            <div class="recommendation-explanation mb-
3">
                                                <h5>Пояснення:</h5>
                                                <p><?=
htmlspecialchars($recommendation['explanation']) ?></p>
                                            </div>

                                            <div class="recommendation-code-example
mb-3">
                                                <h5>Приклад коду:</h5>
                                                <pre><code class="language-<?= $language
?>"><?= htmlspecialchars($recommendation['code_example']) ?></code></pre>
                                            </div>
```

```
                                        <div class="recommendation-benefit">
                                            <h5>Переваги:</h5>
                                            <p><?=
htmlspecialchars($recommendation['benefit']) ?></p>
                                        </div>
                                    </div>
                                </div>
                            </div>
                        <?php endforeach; ?>
                    </div>
                </div>
            </div>
        <?php endif; ?>

        <?php if (count($suggestions) > 0): ?>
            <div class="card mt-4">
                <div class="card-header">
                    <h3 class="mb-0">
                        <i data-feather="bulb" class="me-2"></i>
                        Загальні рекомендації
                    </h3>
                </div>
                <div class="card-body">
                    <ul class="recommendations-list">
                        <?php foreach ($suggestions as $suggestion): ?>
                            <li><?= htmlspecialchars($suggestion) ?></li>
                        <?php endforeach; ?>
                    </ul>
                </div>
            </div>
```

```
                    <?php endif; ?>

                    <!-- Візуалізація роботи ШІ -->
                    <div class="card mt-4">
                        <div class="card-header">
                            <h3 class="mb-0">
                                <i data-feather="cpu" class="me-2"></i>
                                Візуалізація роботи ШІ
                            </h3>
                        </div>
                        <div class="card-body">
                            <div id="ai-visualization"></div>
                        </div>
                    </div>



                </div>
            </div>
        </div>
    </section>

    <script>
        // Store analysis data for export
        const analysisData = {
            filename: "<?= htmlspecialchars($filename) ?>",
            language: "<?= htmlspecialchars($language) ?>",
            score: <?= $score ?>,
            issues: <?= json_encode($issues) ?>,
            detailed_recommendations: <?= json_encode($detailed_recommendations)
?>,
```

```
            suggestions: <?= json_encode($suggestions) ?>,
            timestamp: new Date().toISOString()
        };

        document.addEventListener('DOMContentLoaded', function() {
            // Initialize syntax highlighting with line numbers
            Prism.highlightAll();

            // Initialize syntax highlighting for code examples in recommendations and issues
            document.querySelectorAll('.recommendation-code-example pre code, .issue-code-example pre code').forEach(block => {
                Prism.highlightElement(block);
            });

            // Highlight issue lines in the code
            const issueItems = document.querySelectorAll('.issue-item');
            const codeLines = document.querySelectorAll('.line-numbers .line-numbers-rows > span');

            issueItems.forEach(item => {
                const lineStart = parseInt(item.getAttribute('data-line-start')) - 1;
                const lineEnd = parseInt(item.getAttribute('data-line-end')) - 1;

                for (let i = lineStart; i <= lineEnd && i < codeLines.length; i++) {
                    codeLines[i].classList.add('highlighted-line');

                    // Add severity class
                    const severityElem = item.querySelector('.issue-severity');
                    if (severityElem) {
                        if (severityElem.classList.contains('bg-danger')) {
```

```
                codeLines[i].classList.add('critical-line');
            } else if (severityElem.classList.contains('bg-warning')) {
                codeLines[i].classList.add('warning-line');
            } else if (severityElem.classList.contains('bg-info')) {
                codeLines[i].classList.add('info-line');
            }
        }
    }
});

// Click on issue to scroll to code
issueItems.forEach(item => {
    item.addEventListener('click', () => {
        const lineStart = parseInt(item.getAttribute('data-line-start')) - 1;
        if (lineStart >= 0 && lineStart < codeLines.length) {
            // Highlight the clicked issue
            document.querySelectorAll('.issue-item.active').forEach(el => {
                el.classList.remove('active');
            });
            item.classList.add('active');

            // Scroll to the line
            const codeContainer = document.querySelector('.code-container');
            const lineHeight = codeLines[0].offsetHeight;
            codeContainer.scrollTop = lineHeight * lineStart - 100;

            // Flash the line for visibility
            codeLines[lineStart].classList.add('flash-highlight');
            setTimeout(() => {
                codeLines[lineStart].classList.remove('flash-highlight');
            }, 1500);
```

```
            }
        });
    });

    // Toggle line numbers
    document.getElementById('toggle-line-numbers').addEventListener('click',
function() {
        const codeElement = document.querySelector('.line-numbers');
        codeElement.classList.toggle('no-line-numbers');
    });

    // Copy code functionality
    document.getElementById('copy-code').addEventListener('click', function()
{
        const codeText = document.querySelector('.language-<?= $language
?>').textContent;
        navigator.clipboard.writeText(codeText).then(() => {
            const btn = this;
            const originalHtml = btn.innerHTML;
            btn.innerHTML = '<i data-feather="check" class="me-1"></i>
Copied!';
            feather.replace();
            setTimeout(() => {
                btn.innerHTML = originalHtml;
                feather.replace();
            }, 2000);
        });
    });

    // Export as JSON
```

```
            document.getElementById('export-json').addEventListener('click',
function() {
                const dataStr = JSON.stringify(analysisData, null, 2);
                const dataUri = 'data:application/json;charset=utf-8,'+
encodeURIComponent(dataStr);

                const exportLink = document.createElement('a');
                exportLink.setAttribute('href', dataUri);
                exportLink.setAttribute('download', '<?= htmlspecialchars($filename)
?>_analysis.json');
                document.body.appendChild(exportLink);
                exportLink.click();
                document.body.removeChild(exportLink);
            });

            // Export as PDF (mock function - would require a PDF library in
production)
            document.getElementById('export-pdf').addEventListener('click',
function() {
                alert('PDF export would be implemented with a library like jsPDF in
production.');
                // In a real implementation, we would generate a PDF with the analysis
results
            });
        });
    </script>
```

Джерело: Розроблено автором на основі htdocs/pages/analyze.php

ДОДАТОК Б. Приклади запитів та відповідей OpenAI API

У цьому додатку наведено приклади структури запитів, що надсилаються до OpenAI API, та типових відповідей, які очікує та обробляє система "CodeAnalyzer".

Приклад Б.1 - Структура JSON-тіла POST-запиту до OpenAI API для аналізу коду

```
{
   "model": "gpt-4o",
   "messages": [
      {
         "role": "system",
         "content": "Ти – досвідчений експерт з аналізу програмного коду та найкращих практик розробки. Твоя відповідь ОБОВ'ЯЗКОВО має бути українською мовою."
      },
      {
         "role": "user",
         "content": "Будь ласка, виступи в ролі досвідченого експерта з аналізу програмного коду... (повний текст промпту, як у функції php_analyze_code, включаючи вимоги до структури JSON-відповіді та сам код для аналізу у форматі ```language\nCODE\n```)"
      }
   ],
   "temperature": 0.2,
   "response_format": { "type": "json_object" }
}
```

Джерело: Сформовано автором на основі логіки функції php_call_openai_api у файлі htdocs/utils/php_code_analyzer.php.

Приклад Б.2 - Очікувана структура JSON-відповіді від OpenAI API для аналізу коду

```
{
   "issues": [
      {
         "line_start": початок проблеми у коді,
         "line_end": кінець проблеми у коді,
         "description": "Тут ми описуємо суть проблеми",
         "suggestion": "Поради щодо покращення",
         "severity": "серйозність проблеми"
      }
      // ... інші об'єкти проблем ...
   ],
   "detailed_recommendations": [
      {
         "title": "Назва помилки/поради",
         "explanation": "Пояснення.",
         "code_example": "Приклад коду, що вирішує проблему",
         "benefit": "Переваги цього коду."
      }
      // ... інші об'єкти детальних рекомендацій ...
   ],
   "suggestions": [
      "Тут пропозиція щодо покращення коду."
      // ... інші рядки загальних пропозицій ...
   ],
   "score": кількість балів
}
```

Джерело: Сформовано автором на основі аналізу відповідей API та вимог промпту.

Приклад Б.3 - Структура JSON-тіла POST-запиту до OpenAI API для аналізу змін при порівнянні коду (фрагмент промпту користувача)

```
{
   "model": "gpt-4o",
   "messages": [
      {
         "role": "system",
         "content": "Ти – досвідчений експерт з аналізу програмного коду, який спеціалізується на оцінці змін між версіями коду. Твоя відповідь ОБОВ'ЯЗКОВО має бути українською мовою."
      },
      {
         "role": "user",
         "content": "Проаналізуй зміни коду на мові [LANGUAGE] між оригінальною та новою версією. Надай детальну оцінку змін, включаючи: 1. Чи є зміни покращенням кодової бази? 2. Чи вирішують зміни конкретні проблеми? 3. Чи дотримуються зміни найкращих практик? 4. Чи є потенційні проблеми або покращення для змін? Відповідь має бути у форматі JSON об'єкта зі структурою: {\"impact_assessment\": \"string\", \"improvements\": [\"string\"], \"concerns\": [\"string\"], \"suggestions\": [\"string\"], \"quality_change\": integer (-10 до +10)}. Ось оригінальний код: ```[LANGUAGE]\n[ORIGINAL_CODE]\n``` Ось новий код: ```[LANGUAGE]\n[NEW_CODE]\n``` Ось diff: [FORMATTED_DIFF]"
      }
   ],
   "temperature": 0.3,
   "response_format": { "type": "json_object" }
}
```

Джерело: Сформовано автором на основі логіки функції php_analyze_changes_with_ai у файлі htdocs/utils/php_code_analyzer.php.

Приклад Б.4 - Очікувана структура JSON-відповіді від OpenAI API для аналізу змін

```
{
   "impact_assessment": "Це короткий опис того, що ми зробили",
   "improvements": [
      "Як ви можете удосконалили свій код.",
   ],
   "concerns": [
      "Проблеми.",
   ],
   "suggestions": [
      “Тут поради”
   ],
   "quality_change": оцінка
}
```

Джерело: Сформовано автором на основі аналізу відповідей API та вимог промпту для порівняння.

ДОДАТОК В. Інструкція користувача

Ця інструкція призначена для користувачів вебсервісу "CodeAnalyzer" та описує основні кроки для роботи з системою.

В.1. Загальний опис та початок роботи "CodeAnalyzer" – це онлайн-платформа, що дозволяє аналізувати програмний код за допомогою штучного інтелекту для виявлення помилок, вразливостей та отримання рекомендацій щодо покращення. Сервіс також надає функцію порівняння двох версій коду. Для початку роботи перейдіть на головну сторінку сервісу.

В.2. Реєстрація та вхід (опціонально, для збереження історії) Для збереження історії ваших аналізів та доступу до них у майбутньому рекомендується створити обліковий запис.

- **Реєстрація**: Натисніть кнопку "Реєстрація". Заповніть форму (ім'я, email, пароль). Натисніть "Зареєструватися".
- **Вхід**: Натисніть "Увійти". Введіть email та пароль. Натисніть "Увійти".
- **Відновлення пароля**: На сторінці входу натисніть "Забули пароль?" та дотримуйтесь інструкцій.

В.3. Аналіз коду

1. **Перейдіть до секції завантаження коду** (зазвичай на головній сторінці).
2. **Оберіть спосіб надання коду**:
   - **Завантажити файл**: Перетягніть файл у зону drag-and-drop або оберіть через діалог.
   - **Вставити код**: Перейдіть на вкладку "Вставити код", оберіть мову та вставте код у текстове поле.
3. **Розпочніть аналіз**: Натисніть кнопку "Аналізувати код".
4. **Перегляд результатів**: Вас буде перенаправлено на сторінку результатів. Тут ви побачите загальну оцінку, список проблем (з описом, рядками, серйозністю, пропозиціями виправлення), вихідний код з підсвічуванням та детальні рекомендації. Клік на проблему підсвітить відповідні рядки в коді.

В.4. Порівняння версій коду

1. **Перейдіть на сторінку порівняння** (через навігаційну панель).
2. **Заповніть форму**: Оберіть мову, вставте початкову та нову версії коду у відповідні поля.
3. **Розпочніть порівняння**: Натисніть "Порівняти версії коду".
4. **Перегляд результатів**: Відобразиться оцінка впливу змін, зміна якості, метрики, списки покращень та проблем, пропозиції від ШІ, а також візуальне порівняння коду (Diff) з підсвічуванням змін.

В.5. Робота з історією аналізів (для авторизованих користувачів)

- **Перегляд історії**: У меню користувача оберіть "Мої аналізи".
- **Список аналізів**: Відобразиться список ваших попередніх аналізів/порівнянь (назва, мова, дата, оцінка).
- **Дії з аналізом**: "Переглянути" для відкриття деталей, "Видалити" для видалення запису (з підтвердженням).
- Передбачено елементи для фільтрації та сортування списку.

В.6. Управління профілем (для авторизованих користувачів)

- **Перехід до профілю**: У меню користувача оберіть "Мій профіль".
- **Редагування даних**: Можна змінити ім'я, адресу електронної пошти.
- **Зміна пароля**: Введіть поточний пароль, потім новий та його підтвердження.
- **Збереження змін**: Натисніть "Зберегти зміни".

В.7. Перевірка статусу OpenAI API

- **Перехід на сторінку статусу**: У навігаційній панелі оберіть "Статус API".
- **Інформація про API**: Відобразиться статус API-ключа (якщо налаштований), з'єднання з OpenAI, час останньої перевірки.
- **Тестування з'єднання**: Кнопка "Перевірити з'єднання з API" ініціює тестовий запит.

В.8. Вихід з системи У меню користувача оберіть "Вийти".

Дотримуючись цієї інструкції, ви зможете ефективно використовувати всі можливості вебсервісу "CodeAnalyzer".