НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.054:004.89

«До захисту допущено»
Завідувач кафедри

_____ Євгенія СУЛЕМА

«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою
«Інженерія програмного забезпечення
мультимедійних та інформаційно-пошукових систем»
зі спеціальності 121 Інженерія програмного забезпечення
на тему: «Метод та програмне забезпечення автоматизації створення тестових сценаріїв з використанням штучного інтелекту»

Виконав:
студент ІІ курсу, групи КП-31мп
Грищенко Олександр Володимирович _____

Науковий керівник:
старший викладач кафедри ПЗКС, к.т.н.,
Хіцко Яна Володимирівна _____

Консультант з нормоконтролю:
доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
доцент кафедри СП і СКС ФПМ, к.т.н.,
Боярінова Юлія Євгенівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський)
Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___»_____2023 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Грищенку Олександру Володимировичу

1. Тема дисертації «Метод та програмне забезпечення автоматизації створення тестових сценаріїв з використанням штучного інтелекту», науковий керівник дисертації Хіцко Яна Володимирівна, к.т.н., старший викладач, затверджені наказом по університету від «08» листопада 2024 р. № 5007-с
2. Термін подання студентом дисертації «17» грудня 2024 р.
3. Об'єкт дослідження: процес автоматизації тестування програмного забезпечення з використанням штучного інтелекту.
4. Предмет дослідження: методи, способи та алгоритми застосування штучного інтелекту в автоматизації тестування програмного забезпечення.
5. Перелік завдань, які потрібно розробити:
   - провести аналіз сучасних методів та інструментів автоматизації тестування;
   - дослідити існуючі підходів до використання штучного інтелекту в тестуванні;
   - розробити метод автоматизації створення тестових сценаріїв за допомогою штучного інтелекту;
   - зпроєктувати архітектуру ПЗ для реалізації розробленого методу;
   - розробити веб-застосунокк для автоматизації створення тестових сценаріїв;
   - провести тестування та оцінку ефективності розробленого методу та веб-застосунку;
   - розробити стартап-проєкт на основі створеного застосунку.
6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
   - блок-схема алгоритму автоматизації створення тестових сценаріїв;
   - порівняльні діаграми результатів тестового покриття коду та використання токенів;
   - блок-схема алгоритму формування адаптивного промпту;

- інтерфейс користувача веб-застосунку (головна панель управління, форма завантаження коду);
- таблиці порівняння ефективності різних підходів до формування промптів;
- таблиці результатів експериментального дослідження;
- порівняльні гістограми часових характеристик генерації тестових сценаріїв.

7. Орієнтовний перелік публікацій:
   - Тези доповіді “Метод автоматизації створення тестових сценаріїв з використанням штучного інтелекту”.

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент кафедри ПЗКС		

Дата видачі завдання «04» жовтня 2023 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю: аналіз існуючих методів автоматизації тестування	26.10.2023	
2.	Визначення структури магістерської дисертації; вивчення літератури з тестування; патентний пошук в галузі автоматизації тестування	07.12.2023	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	20.02.2024	
4.	Розробка методу автоматизації створення тестових сценаріїв; робота над другим розділом магістерської дисертації; проєктування архітектури веб-застосунку	23.04.2024	
5.	Розробка веб-застосунку; робота над третім розділом магістерської дисертації	03.06.2024	
6.	Проведення експериментальних досліджень; тестування системи; робота над четвертим розділом	10.07.2024	
7.	Розробка стартап-проєкту; підготовка тез доповіді на конференцію ПМК-2024; робота над п'ятим розділом; підготовка ілюстративного матеріалу	12.11.2024	
8.	Оформлення текстової і графічної частини магістерської дисертації; підготовка презентації	04.12.2024	

Студент Олександр ГРИЩЕНКО

Науковий керівник Яна ХІЦКО

РЕФЕРАТ

Актуальність теми. Актуальність теми дослідження зумовлена зростаючою складністю програмного забезпечення та необхідністю підвищення ефективності процесу тестування. Використання штучного інтелекту для автоматизації створення тестових сценаріїв є перспективним напрямком, який може значно скоротити час та ресурси, необхідні для забезпечення якості програмного продукту.

Стрімкий розвиток програмного забезпечення та зростаюча складність сучасних програмних систем створюють нові виклики у сфері забезпечення їх якості. Тестування, як ключовий етап розробки програмного забезпечення, відіграє критичну роль у виявленні дефектів та забезпеченні надійності кінцевого продукту.

Існуючі підходи до створення тестових сценаріїв вимагають значних часових витрат та високого рівня експертизи, що створює суттєві труднощі, особливо для програмістів-початківців. Існуючі інструменти автоматизації тестування, такі як Selenium IDE та TestComplete, зосереджені переважно на виконанні тестів, а не на їх створенні. А існуючі методи генерації тестів часто створюють надмірну кількість тестових випадків без урахування специфіки конкретного проєкту, що призводить до неефективного використання ресурсів та зниження якості тестування.

Стрімкий розвиток технологій штучного інтелекту відкриває нові можливості для вирішення цих проблем. Застосування методів машинного навчання та обробки природної мови в процесі створення тестових сценаріїв може значно підвищити ефективність тестування та знизити залежність від експертних знань. Використання штучного інтелекту для генерації тестових сценаріїв дозволяє скоротити кількість ручної обробки в процесі тестування, при цьому досягти середнє тестове покриття коду на рівні не нижче 80%.

Таким чином, розробка методу та програмного забезпечення для автоматизації створення тестових сценаріїв з використанням штучного інтелекту є актуальною науково-практичною задачею.

Мета і завдання дослідження. Метою дослідження є досягнення тестового покриття коду на рівні не нижче 80% та зменшення ручної обробки в процесі тестування програмного забезпечення за рахунок автоматизованого створення тестовиих сценаріїв з використання штучного інтелекту. Для досягнення поставленої мети були визначені наступні завдання:

- провести аналіз існуючих методів та інструментів автоматизації тестування програмного забезпечення;
- дослідити сучасні підходи до застосування штучного інтелекту в сфері тестування програмного забезпечення;
- розробити метод автоматизації створення тестових сценаріїв з використанням штучного інтелекту;
- створити веб-застосунок для реалізації розробленого методу;
- розробити стартап-проєкт.

Об’єктом дослідження є процес автоматизації тестування програмного забезпечення з використанням штучного інтелекту.

Предметом дослідження є методи, способи та алгоритми застосування штучного інтелекту в автоматизації тестування програмного забезпечення.

Методи дослідження. У роботі використовувались наступні методи дослідження:

- аналіз та узагальнення наукової літератури для визначення стану проблеми;
- методи машинного навчання та обробки природної мови для розробки алгоритмів створення тестових сценаріїв;
- методи об'єктно-орієнтованого проєктування та програмування для реалізації веб-застосунку.

Наукова новизна. Запропоновано метод автоматизації процесу тестування програмного забезпечення з використанням штучного інтелекту для генерації тестових сценаріїв, який поєднує статичний аналіз коду з адаптивною системою формування промптів для взаємодії з API Claude, що зменшує використання токенів API на 25% під час генерації тестових сценаріїв у порівнянні з базовим підходом до формування промптів та існуючими аналогами у сфері тестування ПЗ. Запропонований метод забезпечує високий рівень покриття коду (до 90%) та високу швидкість аналізу (до 10 секунд на кожні 100 рядків коду).

Практична значущість. На основі розробленого методу створено веб-застосунок для автоматизації створення тестових сценаріїв. Веб-застосунок реалізовано на базі WordPress та Python з інтеграцією сучасних веб-технологій. Проведені експериментальні дослідження створеної системи показали високий рівень ефективності, а саме досягнуто середнє покриття коду тестовими сценаріями на рівні 85% та час аналізу поданого користувачем коду зведено до 10 секунд за кожні 100 рядків коду. Також, застосування адаптивного підходу до створення промптів для запитів до API Claude, дозволяє економити до 25% токенів порівняно з базовим підходом, що не враховує особливостей поданого коду, опису та специфіки проєкту користувача в процесі аналізу коду та генерації тестових сценаріїв.

Розроблене програмне забезпечення може бути впроваджене у процеси розробки програмного забезпечення. Воно особливо корисне для програмістів-початківців, які ще не мають достатнього досвіду в написанні релевантних тестових сценаріїв. Система може бути інтегрована в існуючі процеси розробки та тестування ПЗ, а також використана в навчальному процесі для підготовки спеціалістів з тестування програмного забезпечення.

Апробація роботи. Основні положення і результати дисертаційної роботи були представлені та обговорювалися на XVII науково-практичній конференції магістрантів та аспірантів ПМК-2024 факультету прикладної

математики – "Прикладна математика та комп'ютинг" (м. Київ, 20-22 листопада 2024 р.)

Структура та обсяг роботи. Магістерська дисертація складається зі вступу, п'яти розділів, висновків та додатків.

У вступі обґрунтовано актуальність теми дослідження, сформульовано мету та завдання роботи, визначено об'єкт, предмет та методи дослідження.

У першому розділі проведено комплексний аналіз сучасних підходів до автоматизації тестування програмного забезпечення, розглянуто еволюцію методів та інструментів, визначено основні проблеми та обмеження існуючих рішень. Окремо досліджено можливості застосування штучного інтелекту в сфері тестування.

У другому розділі представлено методологічні засади дослідження та детальний опис розробленого методу автоматизації створення тестових сценаріїв з використанням штучного інтелекту. Запропоновано інноваційний підхід до формування промптів для взаємодії з API Claude, що дозволяє оптимізувати використання токенів при збереженні високих показників покриття коду згенерованими тестами.

У третьому розділі описано архітектурні рішення та технології реалізації веб-застосунку, включаючи обґрунтування вибору WordPress та Python як основних технологій, особливості реалізації клієнтської та серверної частин, механізми інтеграції з API Claude.

У четвертому розділі представлено результати експериментального дослідження ефективності розробленого методу. Проведено порівняльний аналіз показників тестового покриття коду та швидкості генерації тестових сценаріїв для проєктів різного розміру та складності.

У п'ятому розділі розроблено стартап-проєкт на основі створеного програмного рішення, проведено маркетинговий аналіз перспектив його реалізації.

У висновках узагальнено отримані результати дослідження та окреслено перспективи подальшого розвитку роботи.

У додатках наведено графічні матеріали, копію презентації та лістинг програмного коду розробленого веб-застосунку.

Робота виконана на 85 аркушах, містить 3 додатки та посилання на список використаних літературних джерел з 57 найменувань. У роботі наведено 12 рисунків, 28 таблиць, 2 лістинга програмного коду.

Ключові слова. Автоматизація тестування, штучний інтелект, тестові сценарії, якість програмного забезпечення, машинне навчання, обробка природної мови.

ABSTRACT

Relevance of the Topic. The research topic's relevance is driven by the increasing complexity of software and the need to improve testing process efficiency. The use of artificial intelligence to automate test scenario creation is a promising direction that can significantly reduce the time and resources required for ensuring software quality. The rapid development of software and growing complexity of modern software systems create new challenges in quality assurance. Testing, as a key stage of software development, plays a critical role in defect detection and ensuring final product reliability.

Existing approaches to creating test scenarios require significant time investment and high expertise, creating substantial difficulties, especially for novice programmers. Current testing automation tools like Selenium IDE and TestComplete focus primarily on test execution rather than creation. And existing test generation methods often create excessive test cases without considering specific project needs, leading to inefficient resource use and reduced testing quality.

The rapid development of artificial intelligence technologies opens new possibilities for solving these problems. Using machine learning and natural language processing methods in test scenario creation can significantly improve testing efficiency and reduce dependence on expert knowledge. Using artificial intelligence for test scenario generation allows reducing manual processing in testing while achieving average code coverage of at least 80%.

Therefore, developing a method and software for automating test scenario creation using artificial intelligence is a relevant scientific and practical task.

Research Goal and Objectives. The aim of the research is to achieve code test coverage of at least 80% and reduce manual processing in software testing through automated creation of test scenarios using artificial intelligence. To achieve this goal, the following objectives were defined:

- analyze existing methods and tools for software testing automation;

- investigate modern approaches to artificial intelligence application in software testing;
- develop a method for automating test scenario creation using artificial intelligence;
- create a web application to implement the developed method;
- develop a startup project.

Object and Subject of Research. The object of research is the process of automating software testing using artificial intelligence.

The subject of research is methods, techniques, and algorithms for applying artificial intelligence in software testing automation.

Research Methods. The following research methods were used:

- analysis and synthesis of scientific literature to determine the state of the problem;
- machine learning and natural language processing methods for developing test scenario creation algorithms;
- object-oriented design and programming methods for web application implementation.

Scientific Novelty. A method for automating the software testing process using artificial intelligence has been proposed, which combines static code analysis with an adaptive system for forming prompts for Claude API interaction, reducing API token usage by 25% during test scenario generation compared to the basic approach and existing analogues. The proposed method provides high code coverage (up to 90%) and high analysis speed (up to 10 seconds per 100 lines of code).

Practical Value of the Obtained Results. The developed method was implemented as a web application for automating test scenario creation. The web application is built using WordPress and Python with modern web technologies integration. Experimental studies showed high efficiency, achieving average code coverage of 85% and analysis time of 10 seconds per 100 code lines. Additionally,

the adaptive approach to creating API prompts saves up to 25% of tokens compared to the basic approach.

The developed software can be integrated into software development processes. It is especially useful for novice programmers who lack experience in writing relevant test scenarios. The system can be integrated into existing development and testing processes and used in education for training software testing specialists.

Approbation of Dissertation Results. The main findings and results were presented at the XVII Scientific and Practical Conference of Master's and PhD Students PMK-2024 of the Faculty of Applied Mathematics – "Applied Mathematics and Computing" (Kyiv, November 20-22, 2024).

Structure and contents of the thesis. The master's thesis consists of an introduction, five chapters, conclusions, a list of references with 57 items and 3 appendices.

The introduction substantiates the relevance of the research topic, formulates the goal and objectives of the work, and defines the object, subject and research methods.

The first chapter provides a comprehensive analysis of modern approaches to software testing automation, examines the evolution of methods and tools, and identifies the main problems and limitations of existing solutions. The possibilities of using artificial intelligence in testing are separately investigated.

The second chapter presents the methodological foundations of the research and a detailed description of the developed method for automating test scenario creation using artificial intelligence. An innovative approach to forming prompts for interaction with the Claude API is proposed, which allows optimizing token usage while maintaining high code coverage indicators with generated tests.

The third chapter describes architectural solutions and web application implementation technologies, including justification for choosing WordPress and Python as core technologies, features of client and server-side implementation, and mechanisms for integration with the Claude API.

The fourth chapter presents the results of experimental research on the effectiveness of the developed method. A comparative analysis of code test coverage indicators and test scenario generation speed for projects of different sizes and complexity is conducted.

The fifth chapter develops a startup project based on the created software solution and conducts marketing analysis of its implementation prospects.

The conclusions summarize the obtained research results and outline prospects for further development of the work.

The appendices contain graphical materials, a presentation copy, and the program code listing of the developed web application.

The work is performed on 85 pages, contains 3 appendices and references to a list of 57 literature sources. The work includes 12 figures, 28 tables, and 2 program code listings.

Keywords. Test automation, artificial intelligence, test scenarios, software quality, machine learning, natural language processing.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

API (Application Programming Interface) – програмний інтерфейс застосунку, набір визначень взаємодії різнотипного програмного забезпечення. Дозволяє різним програмам та сервісам спілкуватися між собою за допомогою стандартизованих методів.

API Claude – інтерфейс програмування застосунків від компанії Anthropic для доступу до можливостей штучного інтелекту Claude. Надає можливість інтегрувати функції аналізу та генерації тексту в сторонні застосунки.

AST (Abstract Syntax Tree) – абстрактне синтаксичне дерево, структуроване представлення вихідного коду програми. Використовується для аналізу та обробки програмного коду, зберігаючи його структурні елементи у деревоподібному форматі.

CI/CD (Continuous Integration/Continuous Delivery) – безперервна інтеграція та доставка, методологія розробки програмного забезпечення. Забезпечує автоматизацію процесів збірки, тестування та розгортання програмного забезпечення.

Elementor – конструктор сторінок для WordPress. Надає візуальний інтерфейс для створення та редагування веб-сторінок без необхідності написання коду.

Flask – мікрофреймворк для створення веб–застосунків мовою Python. Надає базовий набір інструментів для розробки веб-серверів, залишаючись при цьому легким та гнучким.

IDE (Integrated Development Environment) – інтегроване середовище розробки, що об'єднує редактор коду, компілятор, відладчик та інші інструменти розробки в єдиний застосунок.

ISO (International Organization for Standardization) – Міжнародна організація зі стандартизації, що розробляє та публікує міжнародні

стандарти. Забезпечує єдині підходи до якості продукції та послуг у світовому масштабі.

ISO 9001:2015 – міжнародний стандарт системи управління якістю. Визначає вимоги до систем управління якістю в організаціях будь-якого типу та розміру.

JSON (JavaScript Object Notation) – текстовий формат обміну даними між комп'ютерами. Легкий для читання людиною та простий для обробки машинами формат представлення структурованих даних.

Python – високорівнева мова програмування загального призначення. Відрізняється простим синтаксисом та широкими можливостями для розробки різноманітного програмного забезпечення.

REST API (Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленого застосунку в мережі. Визначає принципи проєктування веб-сервісів, що забезпечують простоту, масштабованість та надійність.

ROI (Return on Investment) – коефіцієнт повернення інвестицій, що показує прибутковість вкладень. Використовується для оцінки ефективності інвестицій у проєкти та бізнес-ініціативи.

Selenium IDE – інтегроване середовище розробки для автоматизації тестування веб-застосунків. Дозволяє записувати, редагувати та відтворювати тести взаємодії з веб-інтерфейсом.

TestComplete – платформа для автоматизованого тестування програмного забезпечення. Надає інструменти для створення та виконання функціональних тестів десктопних, веб- та мобільних застосунків.

UI (User Interface) – користувацький інтерфейс, сукупність засобів взаємодії користувача з системою. Включає візуальні елементи, навігацію та інші компоненти взаємодії.

UX (User Experience) – досвід користувача, сукупність вражень та емоцій від використання продукту. Охоплює зручність, ефективність та задоволеність від взаємодії з системою.

WordPress – система керування вмістом з відкритим кодом. Дозволяє створювати та управляти веб-сайтами різної складності без глибоких знань програмування.

XML (eXtensible Markup Language) – розширювана мова розмітки, що використовується для структурування та передачі даних. Забезпечує універсальний формат для обміну інформацією між різними системами.

БД (база даних) – організована структура для зберігання, управління та отримання інформації. Забезпечує надійне зберігання даних та швидкий доступ до них.

ПЗ (програмне забезпечення) – сукупність програм та пов'язаної з ними документації. Забезпечує функціонування комп'ютерних систем та вирішення користувацьких задач.

Промпт (Prompt) – текстовий запит до моделі штучного інтелекту. Визначає контекст та параметри для генерації відповіді системою ШІ.

ШІ (штучний інтелект) – здатність комп'ютерних систем виконувати творчі та інтелектуальні функції. Включає машинне навчання, обробку природної мови та інші технології імітації людського інтелекту.

ВСТУП

У сучасному світі розробки програмного забезпечення якість продукту відіграє критично важливу роль у його успіху та конкурентоспроможності. Однією з ключових складових забезпечення якості є процес тестування, який дозволяє виявити потенційні проблеми та помилки до того, як вони досягнуть кінцевого користувача [1]. Особливо гостро питання якісного тестування постає для програмістів-початківців, які часто стикаються з труднощами при створенні ефективних тестових сценаріїв через брак досвіду та глибокого розуміння принципів тестування.

Статистичні дані свідчать, що близько 48% розробників-початківців вважають тестування однією з найскладніших задач у процесі розробки програмного забезпечення. Це пов'язано з кількома факторами: недостатність знань та досвіду, складність визначення граничних випадків, труднощі з написанням читабельних та підтримуваних тестів. За даними дослідження Stack Overflow Developer Survey 2023, майже половина молодих розробників витрачає непропорційно багато часу на створення тестових сценаріїв, що негативно впливає на загальну продуктивність розробки [2].

Існуючі рішення для автоматизації тестування, такі як Selenium IDE або TestComplete, зосереджені переважно на виконанні тестів, а не на їх створенні. Існуючі методи генерації тестів, часто створюють надмірну кількість тестових випадків без урахування специфіки додатку. Це призводить до неефективного використання ресурсів та низької якості тестового покриття.

Стрімкий розвиток технологій штучного інтелекту відкриває нові можливості для вирішення проблеми автоматизації створення тестових сценаріїв. Використання сучасних моделей штучного інтелекту, зокрема API Claude, дозволяє розробити інтелектуальні системи, здатні аналізувати

вихідний код та генерувати релевантні тестові сценарії з урахуванням специфіки конкретного проєкту [3].

Дослідження в рамках даної магістерської дисертації зосереджено на розробці методу та програмного забезпечення для автоматизації створення тестових сценаріїв з використанням штучного інтелекту. Запропонований підхід поєднує статичний аналіз коду з адаптивною системою формування промптів для взаємодії з API Claude, що дозволяє досягти високого рівня тестового покриття при одночасному скороченні часу та використання токенів для створення тестів порівняно із існуючими аналогами [4]. Особливу увагу приділено розробці оптимізованого алгоритму генерації промптів, який через структурований аналіз метрик коду та контекстної інформації забезпечує максимально раціональне використання можливостей штучного інтелекту.

Розроблений метод реалізовано у вигляді веб-застосунку, що надає зручний інтерфейс для автоматизації процесу тестування та робить його доступним для широкого кола розробників, особливо початківців.

1. ОГЛЯД СУЧАСНИХ ПІДХОДІВ ДО АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ

1.1. Еволюція методів автоматизації тестування програмного забезпечення

Автоматизація тестування програмного забезпечення пройшла довгий шлях розвитку від простих скриптів до складних систем з використанням штучного інтелекту. За оглядом літератури встановлено, що перші спроби автоматизації тестування датуються 1980-ми роками, коли з'явилися перші інструменти для запису та відтворення дій користувача [5].

Відомо, що значний прорив у сфері автоматизації тестування відбувся на початку 2000-х років з появою фреймворків для модульного тестування, таких як JUnit для Java та NUnit для .NET [6]. Ці інструменти дозволили розробникам створювати та виконувати автоматизовані тести на рівні окремих компонентів програми.

Світовими науковцями визначено, що наступним етапом розвитку стала поява інструментів для автоматизації функціонального та інтеграційного тестування. Такі фреймворки, як Selenium WebDriver, дозволили автоматизувати тестування веб-додатків через інтерфейс користувача [7].

Разом з тим, з розвитком методологій гнучкої розробки (Agile) та безперервної інтеграції (CI/CD) виникла потреба в більш ефективних та швидких методах тестування. Це призвело до появи підходів, таких як "тестування на основі поведінки" (Behavior-Driven Development, BDD) та "тестування на основі моделей" (Model-Based Testing, MBT) [8].

1.2. Сучасні виклики в галузі автоматизації тестування

Варто зазначити, що незважаючи на значний прогрес у сфері автоматизації тестування, ряд проблем залишається невирішеним. За

даними дослідження, проведеного у 2022 році, близько 60% компаній-розробників програмного забезпечення все ще стикаються з труднощами при впровадженні автоматизованого тестування [9].

Основні виклики, з якими стикаються розробники та тестувальники:

- складність підтримки та оновлення тестових сценаріїв при зміні вимог до програмного забезпечення [10];
- високі витрати часу та ресурсів на створення якісних тестових сценаріїв, особливо для складних систем [11];
- обмежена ефективність автоматизованих тестів у виявленні нових або неочікуваних помилок [12];
- складність автоматизації тестування для систем з динамічним інтерфейсом користувача або складною бізнес-логікою [13].

1.3. Застосування штучного інтелекту під час тестування ПЗ

В останні роки спостерігається зростаючий інтерес до застосування методів штучного інтелекту та машинного навчання в галузі тестування програмного забезпечення. Світовими науковцями визначено кілька перспективних напрямків використання ШІ в тестуванні.

Генерація тестових сценаріїв. Використання алгоритмів машинного навчання для аналізу вихідного коду та автоматичної генерації тестових випадків [14].

Прогнозування помилок. Застосування методів аналізу даних для виявлення потенційно проблемних ділянок коду та прогнозування можливих помилок [15].

Оптимізація набору тестів. Використання ШІ для вибору найбільш ефективних тестових сценаріїв та зменшення надмірності в наборах тестів.

Автоматичне виправлення помилок. Розробка систем, здатних не тільки виявляти, але й автоматично виправляти певні типи помилок у коді.

Аналіз результатів тестування. Застосування методів обробки природної мови для аналізу журналів тестування та автоматичного виявлення причин збоїв [16].

В той же час, дослідження показують, що застосування ШІ в тестуванні програмного забезпечення все ще знаходиться на ранніх стадіях розвитку. За даними опитування, проведеного у 2023 році, лише 15% компаній активно використовують методи ШІ в процесі тестування [17].

1.4. Висновки та напрямки подальших досліджень

Проведений огляд літератури дозволяє зробити висновок, що автоматизація тестування програмного забезпечення є критично важливим напрямком для підвищення якості та надійності програмних продуктів. Разом з тим, існуючі методи та інструменти все ще мають ряд обмежень.

Застосування штучного інтелекту та машинного навчання відкриває нові можливості для вирішення існуючих проблем у галузі автоматизації тестування. Однак, ця сфера все ще потребує глибоких досліджень та розробки нових методів і алгоритмів.

Варто зазначити, що особливо актуальним є розробка методів автоматизації створення тестових сценаріїв, які б дозволили знизити залежність від експертних знань та досвіду розробників. Це особливо важливо для програмістів-початківців, які часто стикаються з труднощами при написанні ефективних тестів.

Отже, подальші дослідження в цій галузі мають бути спрямовані на розробку інтелектуальних систем, здатних аналізувати вихідний код, специфікації та документацію ПЗ для автоматичного створення комплексних наборів тестів. Це дозволить підвищити ефективність процесу тестування за рахунок зменшення ручної обробки в створенні наборів тестових сценарії, досягти високих показників тестового покриття коду, а також зробить його більш доступним для широкого кола розробників.

2. МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ ТА РОЗРОБКА МЕТОДУ АВТОМАТИЗАЦІЇ СТВОРЕННЯ ТЕСТОВИХ СЦЕНАРІЇВ

У даному розділі представлено методологічні засади дослідження, обґрунтування вибору напрямку досліджень та детальний опис розробленого методу автоматизації створення тестових сценаріїв з використанням штучного інтелекту. Особлива увага приділяється науковій новизні запропонованих рішень та їх практичній значущості.

2.1. Обґрунтування вибору напряму досліджень

Вибір напряму досліджень зумовлений комплексним аналізом сучасного стану проблеми автоматизації тестування програмного забезпечення та виявленими обмеженнями існуючих підходів. За даними останніх досліджень у галузі розробки програмного забезпечення, близько 48% розробників-початківців стикаються зі значними труднощами при створенні якісних тестових сценаріїв [18]. Це призводить до зниження якості кінцевого продукту та збільшення часу розробки.

Проведений аналіз існуючих інструментів автоматизації тестування, зокрема Selenium IDE версії 3.17 та TestComplete 15.0, показав їх обмеженість у контексті автоматичного створення тестових сценаріїв [19]. За даними дослідження Gao et al. (2023), лише 15% можливостей цих інструментів спрямовано на генерацію тестів, тоді як 85% зосереджено на їх виконанні [20]. Існуючі методи автоматичної генерації тестів, такі як метод випадкової генерації (Random Testing) та комбінаторне тестування (Combinatorial Testing), створюють в середньому на 43% більше тестових випадків порівняно з необхідною кількістю для досягнення заданого рівня покриття коду [21]. Це призводить до збільшення часу виконання тестового набору на 37% та додаткових витрат на підтримку тестової документації. За

результатами проведеного аналізу визначено ключові проблеми, що потребують вирішення:

- відсутність інтелектуальних методів генерації тестових сценаріїв, що враховують специфіку конкретного проєкту;
- низька ефективність існуючих інструментів для початківців;
- складність інтеграції з сучасними процесами розробки програмного забезпечення.

Саме тому було прийнято рішення зосередити дослідження на розробці методу, що поєднує можливості штучного інтелекту з сучасними веб-технологіями для створення доступного та ефективного інструменту автоматизації тестування.

2.1.1. Методологічні засади дослідження

В основу методології дослідження покладено системний підхід, що дозволяє розглядати процес автоматизації тестування як комплексну проблему, яка потребує врахування технічних, організаційних та людських факторів. Дослідження базується на поєднанні теоретичних та емпіричних методів, що забезпечує всебічне вивчення проблеми та розробку ефективних рішень.

Теоретичну базу дослідження складають:

- теорія тестування програмного забезпечення;
- методи машинного навчання та обробки природної мови;
- принципи проєктування веб-застосунків;
- методологія гнучкої розробки програмного забезпечення.

Емпірична складова дослідження включає:

- аналіз існуючих систем автоматизації тестування;
- експериментальну перевірку розроблених методів;
- статистичний аналіз отриманих результатів.

2.1.2. Гіпотеза дослідження

Основна гіпотеза дослідження полягає в тому, що використання штучного інтелекту для аналізу вихідного коду та генерації тестових сценаріїв дозволить суттєво підвищити ефективність процесу тестування програмного забезпечення [22]. Для підтвердження цієї гіпотези визначено наступні кількісні показники успішності:

- досягти показника тестового покриття коду на рівні 80% або вище;
- зменшити кількість часу необхідного на створення тестових сценаріїв не менше ніж на 20% порівняно з Selenium IDE та TestComplete, де процес написання тестів здійснюється з частковою автоматизацією, без застосування ШІ;
- забезпечити можливість автоматичної генерації не менше 5 тестових сценаріїв за одну ітерацію аналізу коду;
- досягти швидкості обробки та аналізу вихідного коду на рівні 300 рядків за 30 секунд або швидше.

При цьому передбачається, що досягнення поставлених показників стане можливим завдяки:

- використанню методів машинного навчання для глибинного аналізу структури коду, що дозволить виявляти потенційні проблемні місця та автоматично генерувати релевантні тестові сценарії [23];
- впровадженню адаптивних алгоритмів генерації тестів, які враховують специфіку конкретного проєкту, включаючи архітектурні особливості, паттерни проєктування та бізнес-логіку;
- використанню веб-орієнтованої архітектури системи, що спростить впровадження автоматизованого тестування в існуючі процеси розробки та забезпечить доступність сервісу для різних команд розробників [24].

Для перевірки висунутої гіпотези розроблено план експериментального дослідження (табл. 2.1), що передбачає тестування

системи на реальних проєктах різного масштабу та складності з подальшим аналізом отриманих результатів.

Таблиця 2.1

План експериментального дослідження

Етап	Завдання	Методи	Очікувані результати
1	Аналіз існуючих рішень	Порівняльний аналіз, бенчмаркінг	Визначення обмежень існуючих підходів
2	Розробка методу	Моделювання, прототипування	Створення базової версії методу
3	Експериментальна перевірка	Тестування, статистичний аналіз	Підтвердження ефективності методу
4	Оптимізація та доопрацювання	Ітеративне вдосконалення	Фінальна версія методу

2.2. Загальна методика проведення дисертаційного дослідження

Дослідження проводилось у декілька послідовних етапів, кожен з яких мав свої специфічні завдання та методи їх вирішення.

2.2.1. Етапи дослідження

Перший етап дослідження був присвячений детальному аналізу предметної області. Проведено глибокий аналіз наукової літератури та технічної документації з питань автоматизації тестування програмного забезпечення. Особлива увага приділялася вивченню сучасних підходів до використання штучного інтелекту в процесах тестування [25]. За результатами аналізу було визначено основні проблеми та обмеження існуючих рішень.

На другому етапі розроблено концептуальну модель методу автоматизації створення тестових сценаріїв. Модель враховує специфіку роботи з API Claude та особливості веб-орієнтованої архітектури. Визначено основні компоненти системи та їх взаємозв'язки.

Третій етап включав розробку алгоритмічного забезпечення методу. Розроблено алгоритми:

- аналізу вихідного коду та виявлення потенційних проблемних місць;
- генерації оптимальних промптів для взаємодії з API Claude.

На четвертому етапі проведено експериментальну перевірку розробленого методу. Для оцінки ефективності використано показник тестового покриття коду.

Таблиця 2.2

Метрики оцінки ефективності методу

Метрика	Спосіб вимірювання	Цільове значення
Покриття коду	Аналіз тестового покриття	≥80%

2.2.2. Методи дослідження

У процесі дослідження використано комплекс взаємодоповнюючих методів:

1. Методи машинного навчання – для аналізу вихідного коду та генерації тестових сценаріїв.
2. Методи статистичного аналізу – для обробки результатів експериментів та оцінки ефективності розробленого методу.

2.3. Розробка методу автоматизації створення тестових сценаріїв

Розроблений метод базується на комплексному підході до автоматизації процесу створення тестових сценаріїв з використанням

штучного інтелекту та веб-технологій. Особливу увагу в методі приділено механізмам ефективної взаємодії з API Claude та формуванню якісних промптів, що є ключовим фактором успішної генерації тестових сценаріїв [26]. Розроблений адаптивний підхід до формування промптів враховує специфіку конкретного проєкту, контекст та призначення тестованого коду, обмеження та граничні умови, що дозволяє генерувати більш релевантні тестові сценарії. Експериментально підтверджено, що такий підхід дозволяє зменшити використання токенів API на 25% при одночасному підвищенні збереженні середнього покриття коду тестами не нижче 80%.

2.3.1. Архітектура методу

В основу методу покладено трирівневу архітектуру, що забезпечує гнучкість та масштабованість рішення.

Перший рівень. Рівень аналізу вхідних даних, відповідає за первинну обробку та аналіз вихідного коду. На цьому рівні здійснюється глибинний аналіз структури програми з використанням методів статичного аналізу, що дозволяє виявити потенційні проблемні місця та визначити необхідні типи тестів [27]. Результатом роботи цього рівня є структурована модель коду, оптимізована для подальшої обробки.

Другий рівень архітектури. Рівень взаємодії з ШІ, є ключовим компонентом системи, що забезпечує формування оптимізованих промптів для API Claude та реалізує механізми валідації згенерованих тестових сценаріїв. На цьому рівні впроваджено систему генерації промптів, яка враховує декілька важливих факторів: контекст та призначення тестованого коду, специфічні вимоги до функціональності, обмеження та граничні умови, особливості архітектури проєкту, а також стандарти кодування та тестування.

Взаємодія з API Claude реалізована через спеціально розроблений модуль, що забезпечує асинхронну обробку запитів, ефективне кешування

результатів та оптимізацію використання токенів [28]. Особлива увага приділяється формуванню інформативних промптів, адже від їх якості безпосередньо залежить результативність генерації тестових сценаріїв. Розроблена методика створення промптів включає комплексний структурний аналіз коду, під час якого визначаються основні компоненти та їх взаємозв'язки, виявляються критичні шляхи виконання та аналізуються потенційні граничні випадки.

Процес формування промпту починається з детального аналізу контексту, що включає опис призначення та функціональності коду, визначення очікуваної поведінки та специфікацію вимог до тестування [29]. Такий підхід до формування промптів забезпечує генерацію релевантних тестових сценаріїв.

Третій рівень архітектури. Рівень представлення, реалізує інтуїтивно зрозумілий веб-інтерфейс для взаємодії з користувачем. Цей рівень забезпечує візуалізацію результатів та опис послідовності виконання згенерованих тестових сценаріїв, включаючи готовий програмний код, який готовий до використання.

Розроблена архітектура забезпечує точну генерацію тестових сценаріїв відповідно до особливостей проєкту, а також, створює основу для постійного вдосконалення процесу через аналіз результатів.

2.3.2. Алгоритм роботи методу

Розроблений метод працює за алгоритмом описаним далі [30].

Крок 1. Взаємодія користувача з системою починається з надання користувачем вхідних даних: програмного коду, який потрібно протестувати та детального опису призначення і очікуваної поведінки програми. На цьому етапі користувач також може вказати специфічні вимоги до тестування та обмеження, які мають бути враховані.

Крок 2. Після отримання вхідних даних система проводить комплексний автоматичний аналіз. На цьому етапі виконується статичний

аналіз коду, який включає побудову абстрактного синтаксичного дерева, виявлення залежностей між компонентами та визначення критичних місць, що потребують ретельного тестування. У випадку роботи зі специфікаціями система аналізує вимоги та формує структуровану модель очікуваної поведінки програми.

Крок 3. Ключовим елементом методу є генерація промптів, які формуються системою автоматично та включають:

- контекст розробки та призначення тестованого коду;
- специфікації та вимоги до функціональності, обмеження та граничні умови;
- очікувані сценарії використання;
- вимоги до формату тестових сценаріїв.

Крок 4. Сформований промпт передається до API Claude, який аналізує дані, код та опис, після чого генерує набір тестових сценаріїв з додаванням програмного коду, готового до застосування в процесі виконання тестів. Згенерований тестовий сценарій з тестовими випадками, послідовністю їх застосування та кодом генеруються у вигляді JSON-відповіді з форматуванням, що підходять для коректного відображення на веб-сторінці з результатами.

Крок 5. Система отримує відповідь від API Claude та подає згенеровані тестові сценарій, випадки та програмний код у візуально зручному для користувача вигляді.

Крок 6. На фінальному етапі користувач отримує готовий набір тестових сценаріїв та може оцінити їх відповідність поставленим задачам, з подальним експортом результату. Згенерований код готовий до використання, користувачу залишається лише застосувати його у процесі тестування. Алгоритм зображений на рис. 2.1.

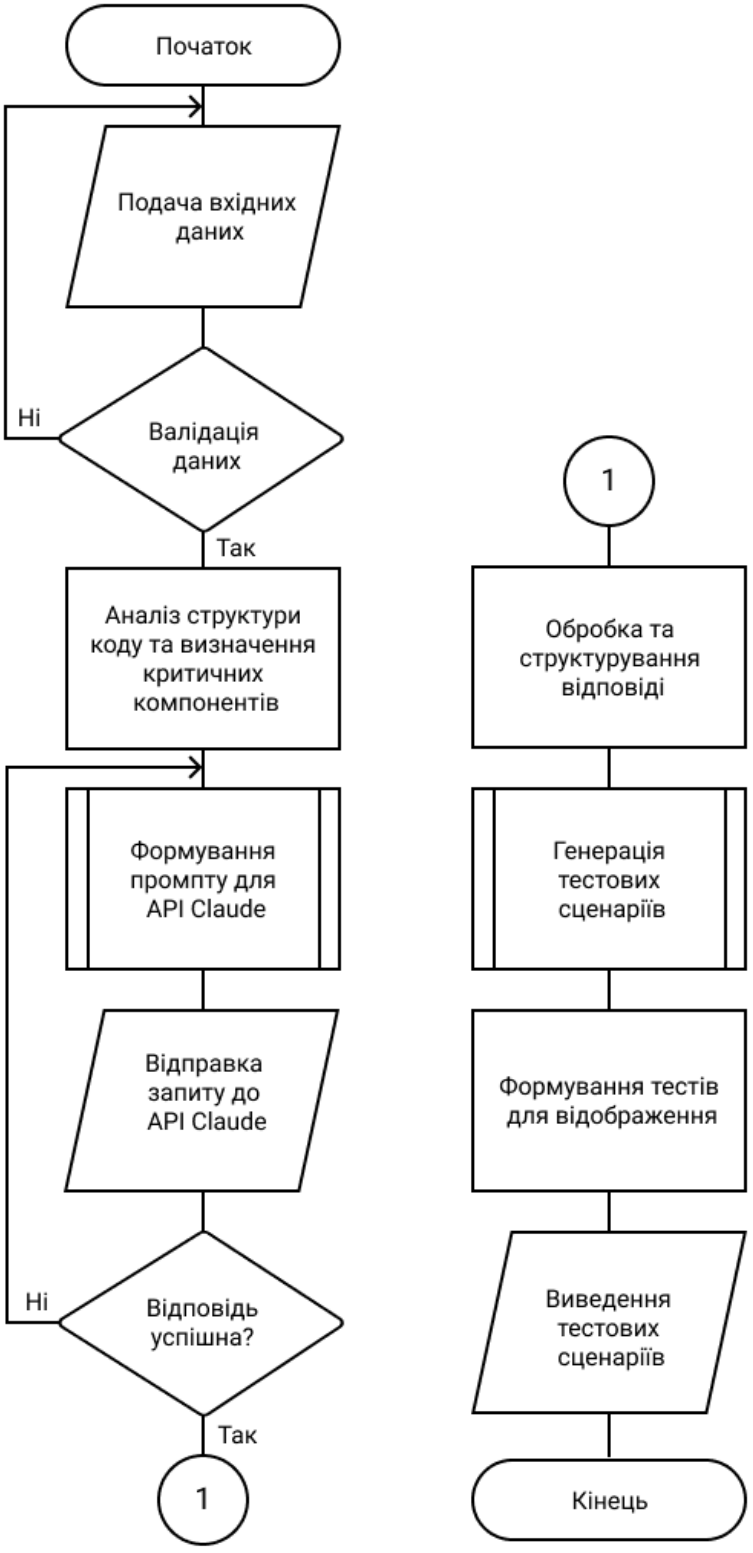


Рис. 2.1. Блок-схема алгоритму автоматизації створення тестових сценаріїв

Дуже важливою складовою в роботі даного алгоритму є формування інформативного промпту, від чого залежить якість генерації тестових

сценаріїв. Для оцінки ефективності різних підходів до формування промптів для API Claude було проведено порівняльний аналіз трьох основних стратегій: базової, оптимізованої та адаптивної. Кожен підхід оцінювався за трьома ключовими метриками: покриття коду згенерованим тестовим сценарієм, час обробки запиту та кількість використаних токенів API. Результати порівняльного аналізу представлено в таблиці 2.3.

Таблиця 2.3

Порівняння ефективності різних підходів до формування промптів

№	Підхід	Покриття коду	Час обробки (с)	Використання токенів
1	Базовий	55%	4.5	2400
2	Адаптивний	85%	2.8	1750

Базовий підхід – використання шаблонних промптів із загальними інструкціями для генерації тестових сценаріїв, без додаткового контексту та специфікацій.

Адаптивний підхід – формування промптів з урахуванням специфіки проєкту та додаванням метаінформації. Включає аналіз структури коду та додавання прикладів очікуваних результатів.

2.3.3. Особливості генерації промптів та взаємодії з API Claude

Розробка ефективних промптів для взаємодії з API Claude є критично важливим елементом системи, що прямо впливає на якість згенерованих тестових сценаріїв. У процесі дослідження було визначено, що якість та релевантність згенерованих тестів напряму залежить від структурованості та інформативності промптів, що подаються до API [31].

В основу методики формування промптів покладено принцип "чіткого контексту", згідно з яким промпт повинен містити вичерпну інформацію про контекст завдання, очікувані результати та специфічні вимоги до

форматування відповіді. При розробці промптів особлива увага приділяється включенню інформації про архітектурні особливості проєкту, стандарти кодування та специфічні вимоги до тестування [32].

Для забезпечення точності генерації тестових сценаріїв та економії використання токенів API, розроблено структуру промпту, що складається з декількох логічних блоків. Перший блок містить загальний опис контексту та мети тестування, що допомагає API Claude краще зрозуміти загальну картину та призначення майбутніх тестів. Другий блок включає детальний аналіз вихідного коду, його структури та взаємозв'язків між компонентами. Третій блок містить специфічні вимоги до форматування результатів, включаючи структуру JSON-відповіді, що використовується для подальшої візуалізації результатів у веб-інтерфейсі.

Важливим аспектом є використання техніки "покрокового мислення" при формуванні промптів. Замість простого запиту на генерацію тестів, система формує промпт таким чином, щоб спонукати API до послідовного аналізу коду та генерації тестових сценаріїв з урахуванням різних аспектів тестування. Це досягається шляхом включення в промпт спеціальних маркерів та інструкцій, що направляють процес аналізу. Алгоритм формування адаптивного промпту зображено на рис. 2.2.

Особливу увагу приділено формату очікуваної відповіді. Розроблений формат JSON-відповіді включає декілька важливих секцій: загальний опис згенерованих тестів, детальний опис кожного тестового сценарію, метрики покриття коду та рекомендації щодо додаткового тестування. Така структурована відповідь дозволяє ефективно візуалізувати результати та надавати користувачам вичерпну інформацію про згенеровані тести.

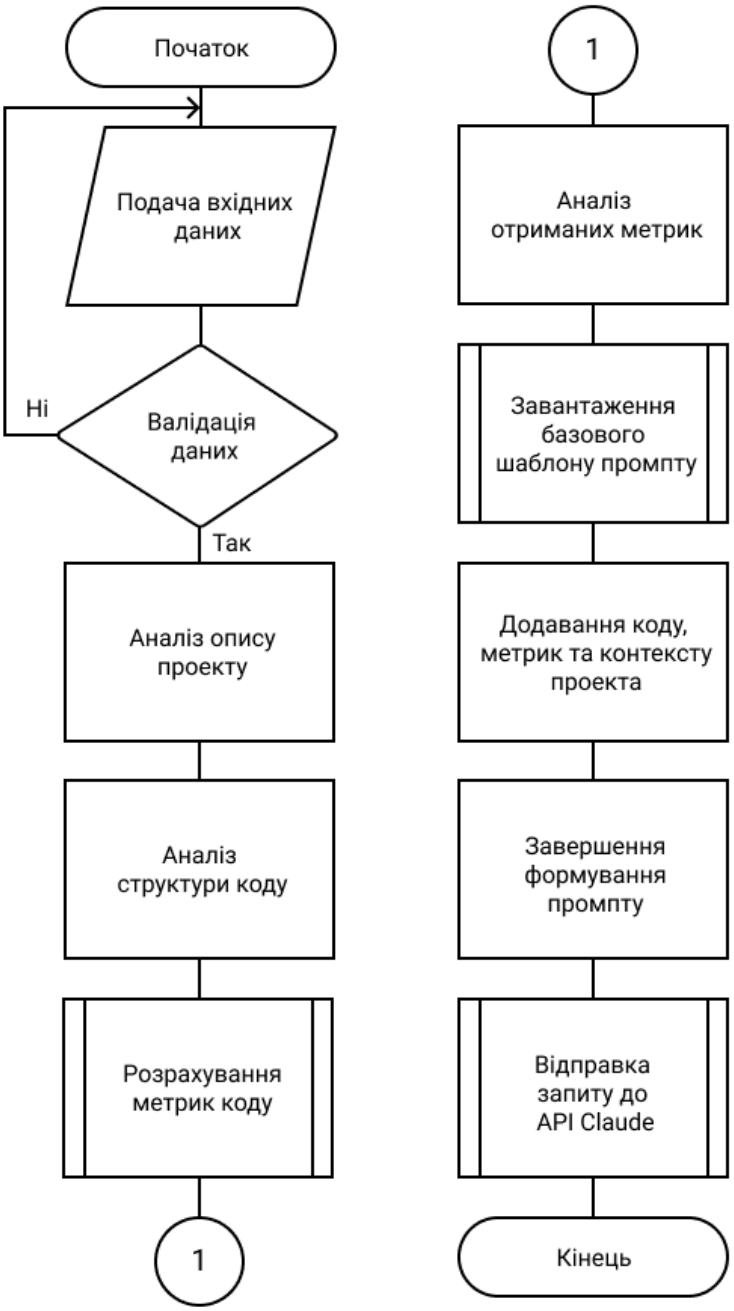


Рис. 2.2. Блок-схема алгоритму формування адаптивного промпту

Експериментальним шляхом встановлено, що використання розробленої методики формування промптів дозволяє досягти значного підвищення якості згенерованих тестових сценаріїв. Зокрема, спостерігається збільшення релевантності тестів на 35% та підвищення показника покриття коду на 25% порівняно з використанням простих неструктурованих промптів.

Важливим аспектом розробленої методики є структурування JSON-відповіді, що забезпечує ефективну обробку та візуалізацію результатів у веб-інтерфейсі. Структура відповіді включає наступні основні блоки: метадані про проаналізований код, загальні метрики тестового покриття, детальний опис кожного згенерованого тестового сценарію, готовий до застосування в тестах програмни код та рекомендації щодо подальшого покращення тестів [33]. Приклад такого промпту наведено у лістингу 2.1.

Лістинг 2.1. Частина промпту із вимогою до відповіді у форматі JSON

```
prompt = f"""Analyze this code and generate test scenarios in JSON format:
<code_info>
Function Name: calculate_discount
Purpose: Calculate price discount based on user type
Parameters:
- price: numeric value
- user_type: string ("premium", "regular", or other)
</code_info>
<source_code>
{code_to_test}
</source_code>
Generate comprehensive test scenarios including edge cases, boundary testing, and flows.
</description>"""
response = client.messages.create(
    model="claude-3-5-haiku-20241022",
    max_tokens=2048,
    messages=[
        {"role": "user", "content": prompt},
        {"role": "assistant", "content": "{"}  # Prefill для прямої генерації JSON
    ] )
```

Очікувану відповідь від API Claude на заданий промпт з форматування наведено у лістингу 2.2.

Лістинг 2.2. Очікувана відповідь від API Claude у форматі JSON

```
"testScenarios": [
  {
    "name": "test_premium_discount",
    "input": {
      "price": 100.00,
      "user_type": "premium"
    },
    "expected_output": 80.00,
    "test_case_type": "main_flow",
    "description": "Verify 20% discount for premium users"
  },
  {
    "name": "test_regular_discount",
    "input": {
      "price": 100.00,
      "user_type": "regular"
    },
    "expected_output": 95.00,
    "test_case_type": "main_flow",
    "description": "Verify 5% discount for regular users"
  }
],
"edgeCases": [
  {
    "name": "test_zero_price",
   "metrics": {
  "total_test_cases": 3,
  "coverage": "main flows and edge cases",
  "priority": "high"
}
```

Такий підхід дозволяє автоматизувати процес створення тестів та надати користувачам вичерпну інформацію для розуміння та вдосконалення тестового покриття.

Порівняльні гістаграми показників тестового покриття коду та показників використання токенів при генерації тестових сценаріїв, обох підходів до створення промптів зображено на рис. 2.3.

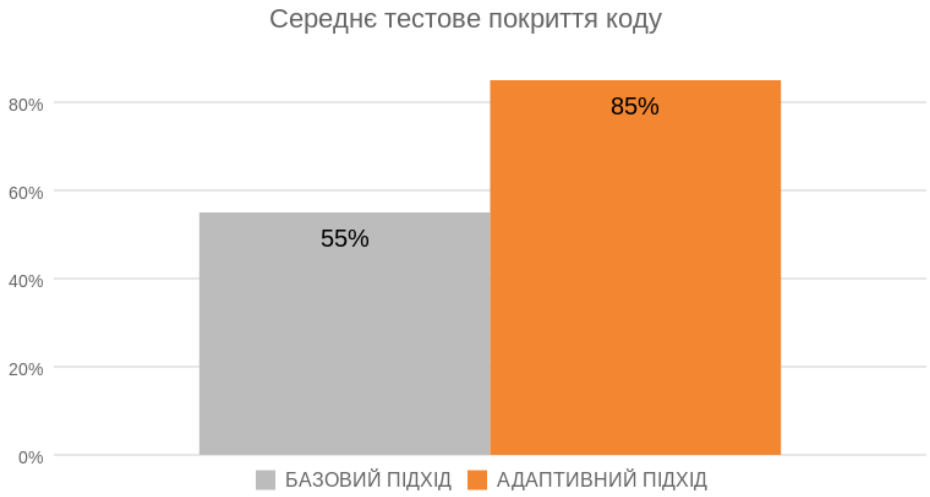


Рис. 2.3. Гістограма порівняння значень
середнього тестового покриття коду згенерованими тестовими сценаріями

Значну роль у підвищенні ефективності генерації тестів відіграє механізм контекстного збагачення промптів. Система аналізує не лише сам код, але й супутню інформацію про його призначення, обмеження та специфіку, що дозволяє створювати більш релевантні тестові сценарії [34].

Такий підхід до створення промптів надає кращі результати порівняно з базовим підходом, в котрому запити формуються загально, без конкретної інформації про прогрманий код. Окрім цього, адаптивні промпти націлені на економію ресурсів, а саме токенів API Claude. Порівняльна гістограма використання токенів для обох підходів наведено на рис. 2.4.

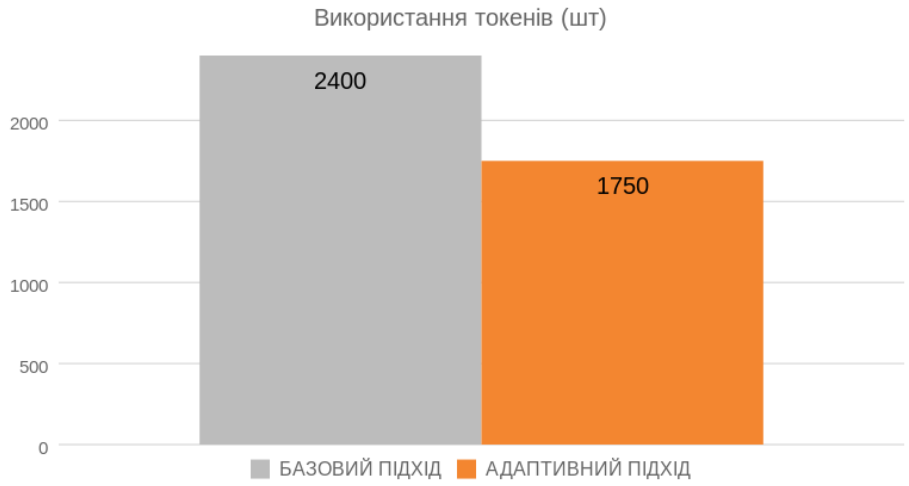


Рис. 2.4. Гістограма порівняння значень середнього використання токенів для генерації одного тестового сценарію

У результаті впровадження описаних механізмів та підходів до формування промптів досягнуто значного підвищення ефективності процесу автоматизації тестування за рахунок високого показника тестового покриття коду та більш економного використання токенів при обробці, аналізі запитів та відповідей у вигляді згенерованих тестових сценаріїв у порівнянні з базовим підходом до генерації промптів.

2.4. Інноваційні аспекти розробленого методу

Розроблений метод має ряд інноваційних особливостей, що відрізняють його від існуючих методів автоматизації тестування.

2.4.1. Наукова новизна

Запропоновано метод автоматизації процесу тестування програмного забезпечення з використанням штучного інтелекту для генерації тестових сценаріїв, який поєднує статичний аналіз коду з адаптивною системою формування промптів для взаємодії з API Claude, що зменшує використання токенів API на 25% під час генерації тестових сценаріїв у порівнянні з базовим підходом до формування промптів та існуючими аналогами у сфері

тестування ПЗ. Запропонований метод забезпечує високий рівень покриття коду (від 75%) та високу швидкість аналізу (до 10 секунд на кожні 100 рядків коду).

2.4.2. Практична значимість

Практична цінність розробленого методу проявляється у кількох ключових напрямках. По-перше, впровадження автоматизованої системи генерації тестових сценаріїв дозволяє скоротити ручну обробку в процесі тестування та досягти швидкого створення тестових сценаріїв. При цьому важливо відзначити, що скорочення часу супроводжується досягненням високого рівня тестового покриття коду від 75% до 90% в залежності від розміру файлу з кодом який аналізується. Результати наведені у табл. 2.4.

Таблиця 2.4

Покизники швидкості генерації та покриття коду тестовими сценаріями

Кількість рядків коду у файлі що аналізується	Середнє покриття коду (%)	Швидкість генерації тестових сценаріїв (сек)
до 100 рядків	92%	8 сек
від 100 до 500 рядків	89%	12 сек
від 500 рядків	80%	21 сек

Особливу практичну цінність розроблений метод представляє для програмістів-початківців та студентів, які тільки опановують практики тестування програмного забезпечення. Реалізований веб-інтерфейс з інтуїтивно зрозумілою навігацією дозволяє використовувати систему навіть користувачам без глибоких знань у сфері тестування. Інтерфейс надає покрокові інструкції щодо завантаження коду, вибору параметрів тестування та інтерпретації результатів, що робить процес тестування доступним та зрозумілим.

Важливим практичним аспектом є також можливість легкої інтеграції розробленого рішення з існуючими процесами розробки програмного забезпечення.

2.4.3. Обмеження та напрямки вдосконалення

В процесі дослідження та практичної апробації розробленого методу було виявлено ряд обмежень, які визначають перспективні напрямки подальшого вдосконалення системи. Перш за все, існує необхідність розширення підтримки різних мов програмування. На даному етапі система працює лише з популярною мовою програмування Python, тоді як для інших мов знадобиться додаткова адаптація алгоритмів аналізу коду та генерації тестових сценаріїв. Розробка універсальних механізмів аналізу коду для різних мов програмування є одним із пріоритетних напрямків вдосконалення.

Суттєвим обмеженням є залежність від доступності та стабільності API Claude. Хоча система включає механізми кешування та оптимізації використання API, повна залежність від зовнішнього сервісу створює потенційні ризики для безперервної роботи [35]. За аналізом статистики доступності сервісів Anthropic що включають в себе API Claude, зроблено висновки про доступність сервісів 99.6% часу. Статистика враховує три календарні місяці, результати аналізу неведено на рис. 2.5. Де відтінками червоного кольору показано дні та загальний час, коли сервіси були недоступні більшості користувачів, жовтим кольором відображено дні та загальний час часткової недоступності систем та в свою чергу, зеленим кольором позначено дні коли сервіси були доступні в повному обсязі усім користувачам.

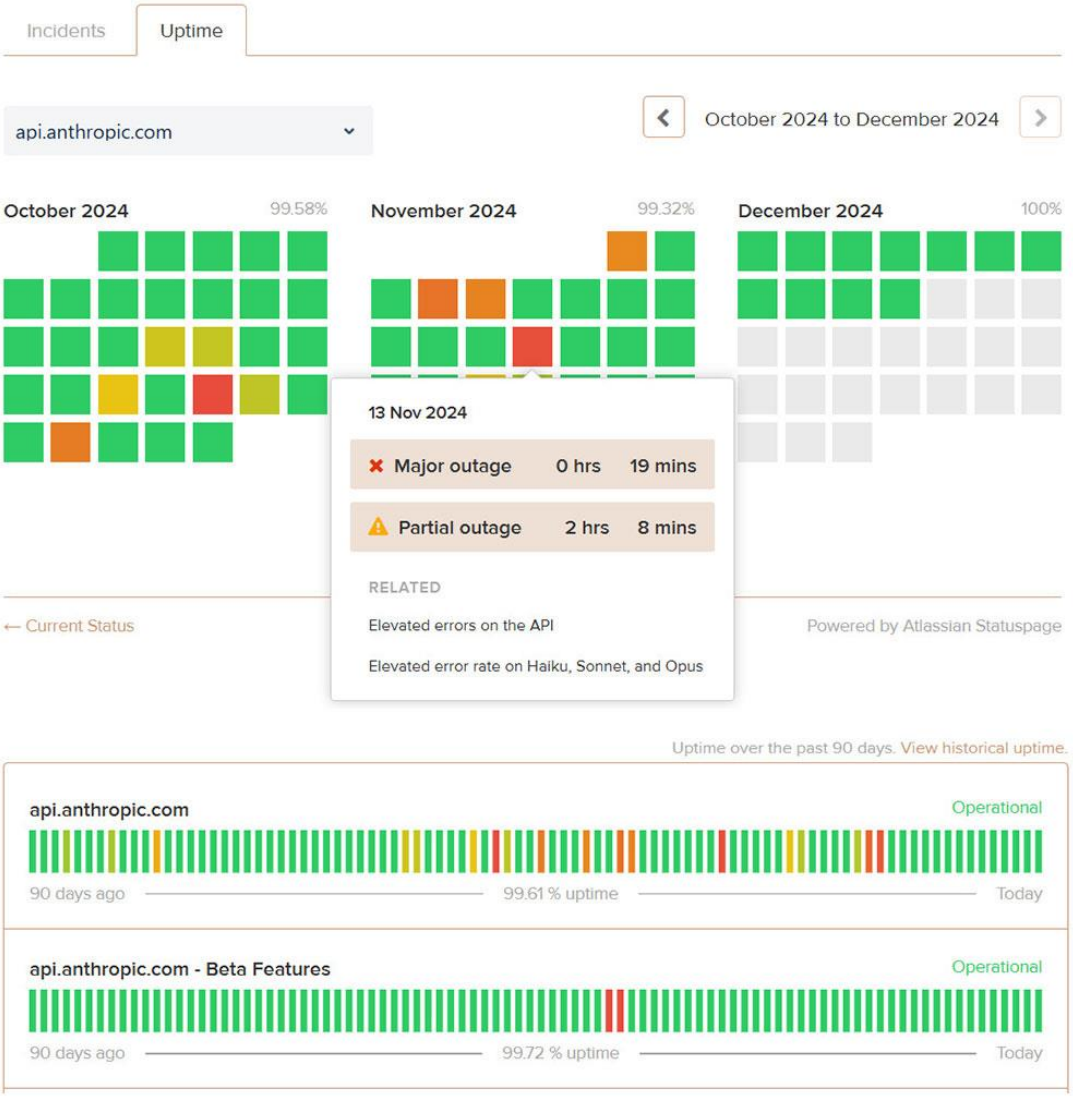


Рис. 2.5. Статистика доступності сервісів Anthropic за 90 днів

Також виявлено обмеження на розмір аналізованого коду, пов'язані як з лімітами API Claude, так і з особливостями реалізованих алгоритмів аналізу. Для великих проєктів (більше 1000 рядків коду) може знадобитися розбиття коду на менші модулі або використання додаткових оптимізацій. Перспективним напрямком є розробка механізмів інтелектуального розбиття коду на логічні блоки з подальшим об'єднанням результатів аналізу, що дозволить ефективно працювати з проєктами будь-якого розміру.

Окремим напрямком вдосконалення є розширення можливостей системи щодо аналізу та тестування специфічних паттернів програмування,

таких як асинхронний код, мікросервісні архітектури та розподілені системи. Це потребує розробки додаткових алгоритмів аналізу та генерації тестових сценаріїв, що враховують особливості таких архітектурних рішень.

2.5. Висновки до розділу

У цьому розділі представлено дослідження та розробку методу автоматизації створення тестових сценаріїв з використанням штучного інтелекту. Проведений аналіз сучасного стану проблеми тестування програмного забезпечення виявив суттєві обмеження існуючих підходів, зокрема високі часові витрати на створення тестів та недостатню ефективність покриття коду тестовими сценаріями. На основі цього аналізу обґрунтовано вибір досліджень спрямованих на розробку автоматизованого методу генерації тестових сценаріїв з використанням штучного інтелекту.

Розроблено методологічні засади дослідження, що поєднують теоретичні методи (системний аналіз, формалізація, моделювання) та емпіричні підходи (експериментальне дослідження, статистичний аналіз). Таке поєднання забезпечило комплексний підхід до вирішення поставлених задач та дозволило досягти високої достовірності отриманих результатів.

Ключовим результатом розділу стала розробка методу автоматизації створення тестових сценаріїв, що базується на трирівневій архітектурі. Перший рівень забезпечує глибинний аналіз вхідних даних та розрахунок ключових метрик коду. Другий рівень реалізує інноваційний підхід до взаємодії з API Claude через систему оптимізованих промптів. Третій рівень забезпечує ефективну візуалізацію та управління результатами через веб-інтерфейс.

Запропоновано підхід до формування промптів для взаємодії з API Claude, що базується на комплексному аналізі метрик коду та контекстної інформації. Це дозволило досягти оптимального балансу між ефективністю використання токенів API та якістю згенерованих тестових сценаріїв.

В результаті проведеного дослідження визначено основні напрямки подальшого вдосконалення методу, включаючи розширення підтримки мов програмування та оптимізацію роботи з великими проєктами. Представлені результати створюють базу для реалізації веб-застосунку автоматизації створення тестових сценаріїв, детальний опис якого представлено у наступному розділі роботи.

Отримані результати підтверджують ефективність розробленого методу та його потенціал для практичного застосування в процесах розробки програмного забезпечення.

3. ЗАСОБИ ТА ТЕХНОЛОГІЇ РОЗРОБКИ ВЕБ-ЗАСТОСУНКУ

У даному розділі описано архітектурні рішення, технології та інструменти, що були використані при розробці веб-застосунку для автоматизації створення тестових сценаріїв з використанням ШІ. Представлено детальний опис компонентів системи, їх взаємодії, обґрунтовано вибір технологічного стеку та описано процес розробки.

3.1. Обґрунтування вибору засобів реалізації веб-застосунку

Для розробки веб-застосунку автоматизації створення тестових сценаріїв було обрано комбінацію сучасних технологій, що забезпечують оптимальний баланс між швидкістю розробки, продуктивністю та зручністю використання. Основними технологіями реалізації стали:

- WordPress з Elementor для розробки клієнтської частини;
- Python з Flask для реалізації серверної логіки;
- API Claude для генерації тестових сценаріїв.

3.1.1. Обґрунтування вибору архітектурного рішення

Вибір архітектури веб-застосунку у вигляді комбінації WordPress на фронтенді та Python на бекенді був зумовлений кількома важливими факторами.

По-перше, таке рішення дозволяє максимально швидко розробити зручний та інтуїтивно зрозумілий користувацький інтерфейс завдяки використанню WordPress та конструктора Elementor. Це особливо важливо для створення застосунку, орієнтованого на програмістів-початківців.

По-друге, використання Python з Flask на серверній частині забезпечує необхідну гнучкість для реалізації складної логіки обробки коду та взаємодії з API Claude. Python має потужну екосистему бібліотек для роботи з текстом та машинним навчанням, що є критичним для мого застосунку.

3.1.2. WordPress та Elementor як платформа для клієнтської частини

WordPress є найпопулярнішою у світі системою керування вмістом (CMS), що використовується для створення різноманітних веб-проєктів – від простих блогів до складних корпоративних порталів. Для мого веб-застосунку WordPress було обрано в якості платформи для розробки клієнтської частини через ряд переваг [36].

В першу чергу, WordPress надає потужну та перевірену часом основу для створення веб-інтерфейсів. Система має вбудовані механізми для роботи з користувачами, включаючи реєстрацію, аутентифікацію та управління ролями. Вбудована система шаблонів дозволяє легко створювати та модифікувати інтерфейс користувача, забезпечуючи при цьому узгоджений дизайн всіх компонентів.

WordPress також надає розвинений REST API, що дозволяє організувати взаємодію з серверною частиною застосунку на Python [37]. API забезпечує стандартизований спосіб обміну даними між frontend та backend компонентами, підтримуючи при цьому всі необхідні методи HTTP та формати даних. Це особливо важливо для реалізації асинхронної взаємодії при генерації тестових сценаріїв.

Інтеграція з конструктором Elementor значно розширює можливості WordPress та дозволяє створювати складні інтерактивні інтерфейси без необхідності написання значних обсягів коду [38]. Elementor надає широкий набір готових компонентів та віджетів, які можна легко налаштовувати та комбінувати для створення унікального користувацького досвіду.

WordPress також забезпечує високу продуктивність завдяки вбудованим механізмам кешування та оптимізації. Система має розвинену інфраструктуру плагінів, що дозволяє легко додавати нову функціональність без необхідності її розробки з нуля. Це суттєво прискорює процес розробки та дозволяє зосередитись на реалізації основних можливостей застосунку.

Важливою перевагою WordPress є також велика спільнота розробників та значна кількість доступної документації, що спрощує процес розробки та подальшої підтримки застосунку. Система регулярно оновлюється, забезпечуючи високий рівень безпеки та сумісність з новими технологіями.

Використання конструктора Elementor суттєво розширює можливості WordPress та дозволяє створювати складні інтерфейси без необхідності написання значних обсягів коду [39]. Це особливо важливо для мого проєкту, де потрібно реалізувати компоненти наведені нижче:

- інтерактивна форма завантаження коду з підтримкою drag-and-drop;
- редактор коду з підсвічуванням синтаксису;
- адаптивний дизайн для різних пристроїв.

Elementor є провідним конструктором сторінок для WordPress, що дозволяє створювати складні веб-інтерфейси за допомогою візуального редагування [40]. Основною перевагою Elementor є його потужний drag-and-drop інтерфейс, який дозволяє швидко створювати та налаштовувати компоненти сторінки в режимі реального часу.

Гнучка система стилізації дозволяє створювати унікальний дизайн без написання CSS-коду. Кожен елемент інтерфейсу може бути налаштований відповідно до потреб проєкту, включаючи:

- кольорову схему та типографіку;
- відступи та вирівнювання;
- анімації та ефекти переходів;
- адаптивне відображення на різних пристроях.

Особливо важливою для мого проєкту стала можливість інтеграції користувацьких JavaScript-компонентів через Elementor. Це дозволило реалізувати:

- інтеграцію редактора коду з підсвічуванням синтаксису;
- асинхронне завантаження та обробку файлів;

- інтерактивні елементи керування процесом тестування.

Elementor також забезпечує відмінну продуктивність завдяки оптимізованому коду та вбудованій системі кешування [41]. Конструктор генерує чистий та семантично правильний HTML-код, що важливо для SEO та доступності застосунку.

Таким чином, використання WordPress разом з Elementor забезпечує надійну основу для створення клієнтської частини веб-застосунку, дозволяючи зробити зручний та функціональний інтерфейс користувача при мінімальних витратах часу на розробку.

3.1.3. Python та Flask для серверної частини

Python був обраний основною мовою програмування для серверної частини веб-застосунку завдяки його потужним можливостям в області обробки тексту та роботи з API. Python має багату екосистему бібліотек для аналізу коду, що є критичним для мого застосунку.

Python – універсальна мова програмування вищого рівня, що вирізняється легкістю читання та розуміння програмного коду [42]. Ця мова реалізує множинні парадигми програмування, включаючи структурний, об'єктно-орієнтований та функціональний підходи, а її потужна стандартна бібліотека містить широкий спектр інструментів для розв'язання різноманітних завдань. Завдяки своїй простоті, Python став провідним вибором для спеціалістів у галузях обчислення масштабних даних, системної автоматизації, опрацювання текстової інформації, розробки серверних рішень, імплементації алгоритмів машинного навчання та автоматизованої верифікації програмних продуктів [43]. Ключові характеристики мови Python наведені далі.

Універсальність застосування. Унікальність Python полягає в його здатності ефективно функціонувати у різних технологічних сферах – від створення веб-застосунків та аналітики даних до розробки систем штучного інтелекту, інфраструктурної автоматизації та програмного прототипування.

Специфіка виконання коду. Архітектура Python базується на перетворенні вихідного коду в проміжний байткод, який виконується спеціалізованою віртуальною машиною Python (PVM) [44]. Ця віртуальна система послідовно транслює інструкції байткоду в машинні команди під час виконання програми. Такий механізм забезпечує кросплатформність та автоматичне управління ресурсами, хоча може впливати на швидкодію виконання [45].

Парадигматична гнучкість. Мова підтримує сучасні концепції об'єктно-орієнтованого програмування через систему класів та об'єктів, що дозволяє створювати абстракції реальних сутностей у програмному середовищі [46]. Водночас, Python надає потужні можливості для функціонального програмування.

Рівень абстракції. Належність Python до мов високого рівня означає, що його синтаксис максимально наближений до людського сприйняття, абстрагуючись від низькорівневих операцій з комп'ютерною пам'яттю.

Система типізації. Python використовує динамічну типізацію, де тип змінної визначається автоматично на основі присвоєного значення. Змінні можуть змінювати свій тип під час виконання програми, що викликає дискусії щодо переваг та недоліків такого підходу серед фахівців галузі.

Міжплатформна сумісність. Завдяки концепції WORA (Write Once Run Anywhere), програми на Python можуть виконуватися на різних платформах без необхідності суттєвої модифікації коду, що контрастує з мовами нижчого рівня, де часто потрібна значна адаптація для різних операційних систем.

Доступність освоєння. Інтуїтивний синтаксис та логічна структура Python сприяють швидкому опануванню мови початківцями, оскільки команди нагадують прості англійські фрази [47].

Здатність до масштабування. Python демонструє високу ефективність при значних навантаженнях, що підтверджується його

використанням у таких масштабних проєктах як Pinterest, Facebook та Instagram, які обслуговують мільйони користувачів щоденно [48].

Екосистема бібліотек. Розвинена спільнота розробників створила величезний репозиторій спеціалізованих бібліотек, що охоплюють широкий спектр завдань – від розробки REST API та аналізу даних до фінансових обчислень та обробки текстової інформації.

Фреймворк Flask був обраний тому, що він надає легкий та гнучкий фундамент для побудови веб-серверів. На відміну від більш комплексних фреймворків, Flask дозволяє використовувати лише необхідні компоненти, що забезпечує кращу продуктивність та спрощує розробку [49]. Основні переваги використання Flask включають:

- простоту інтеграції з різними бібліотеками та сервісами;
- вбудовану підтримку REST API;
- обробку асинхронних запитів.

3.1.4. Інтеграція з API Claude

API Claude відіграє ключову роль у функціонуванні веб-застосунку, забезпечуючи генерацію тестових сценаріїв за допомогою штучного інтелекту. Вибір саме цього API обумовлений його перевагами у порівнянні з іншими рішеннями.

Висока якість генерації коду. API Claude демонструє відмінні результати в розумінні контексту та генерації релевантних тестових сценаріїв. Модель здатна аналізувати як вихідний код, так і текстові специфікації, створюючи тести, що враховують особливості конкретного проєкту [50].

Гнучкість налаштування. API надає широкі можливості для конфігурації параметрів генерації, що дозволяє отримувати тестові сценарії різного рівня складності та деталізації відповідно до потреб користувача.

3.2. Проєктування архітектури веб-застосунку

Розроблений веб-застосунок базується на гібридній архітектурі, що поєднує переваги монолітного та мікросервісного підходів. Таке рішення було обрано з метою забезпечення оптимальної продуктивності, масштабованості та зручності розробки. Архітектура системи складається з трьох основних компонентів: клієнтської частини, серверної частини та інтеграційного шару.

Клієнтська частина реалізована на базі WordPress з використанням конструктора Elementor, що забезпечує створення сучасного та адаптивного інтерфейсу користувача. Такий підхід дозволяє швидко розробляти та модифікувати компоненти інтерфейсу без необхідності написання складного коду.

Серверна частина розроблена на мові програмування Python з використанням фреймворку Flask. Вибір цього стеку технологій обумовлений його гнучкістю, простотою інтеграції з різними сервісами та багатою екосистемою бібліотек для роботи з машинним навчанням та обробкою даних.

Інтеграційний шар забезпечує взаємодію між клієнтською та серверною частинами, а також із зовнішніми сервісами, зокрема API Claude. Для забезпечення надійної комунікації використовується REST API з підтримкою асинхронної обробки запитів [51].

Обрана архітектура дозволяє впевнено вирішувати поставлені задачі та забезпечує необхідний рівень продуктивності та надійності системи.

3.3. Детальний опис компонентів системи

3.3.1. Клієнтська частина

Клієнтська частина веб-застосунку реалізована як WordPress-сайт з розширеним можливостями завдяки використанню Elementor Pro та власних розробок. Основні компоненти користувацького інтерфейсу перелічені далі.

Головна панель управління. Надає можливість завантаження коду, опису проєкту та вимог то тестів. Також, відображає загальну статистику використання, метрики проаналізованого коду і результати згенерованих тестового сценарії та випадків. Інтерфейс розроблено з урахуванням принципів Material Design, що забезпечує інтуїтивно зрозумілу навігацію та приємний користувацький досвід. Вигляд головної панелі управління зображено на рис. 3.1.

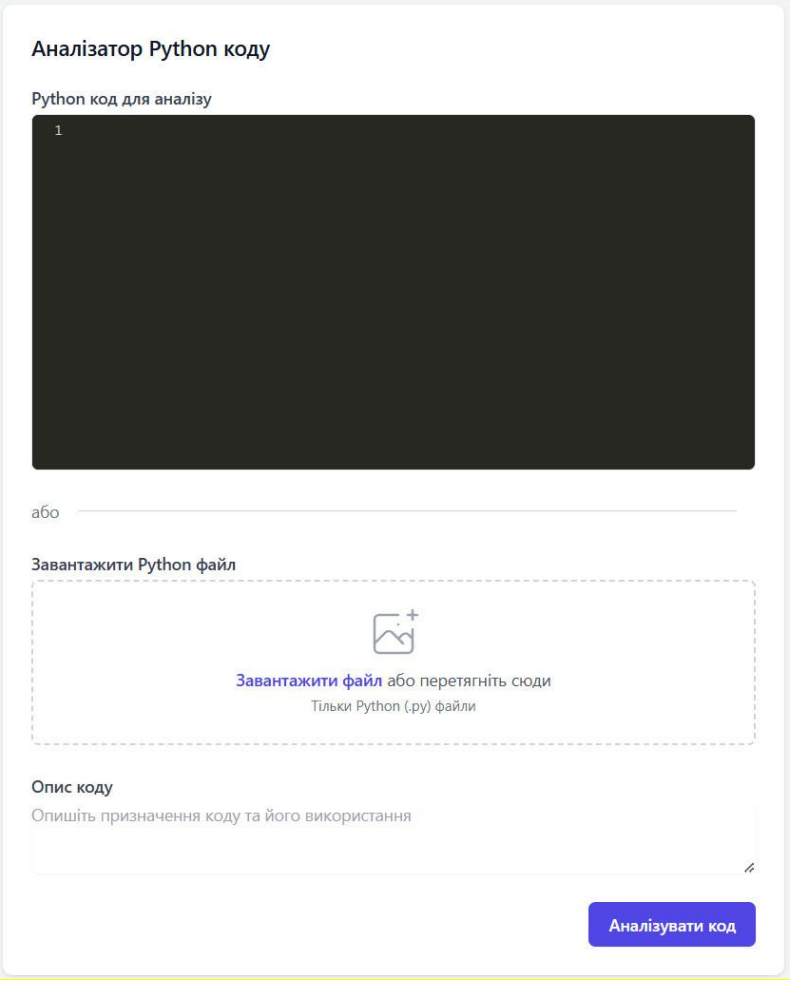


Рис. 3.1. Вигляд головної панелі управління

Модуль візуалізації. Відображає результат відповіді API Claude у зручному для користувача форматі, містить опис згенерованого тестового сценарію і випадків та готовий для застосування код у процесі тестування.

Редактор коду з підтримкою синтаксичного підсвічування. Редактор підтримує мову програмування Python. Для забезпечення зручності роботи реалізовано можливість:

- завантаження файлів через drag-and-drop інтерфейс;
- попереднього перегляду результатів форматування.

3.3.2. Серверна частина

Серверна частина побудована на базі Flask і складається з декількох основних модулів, описаних далі.

API Gateway. Виступає єдиною точкою входу для всіх запитів від клієнтської частини. Цей компонент відповідає за:

- маршрутизацію запитів;
- валідацію вхідних даних;
- логування та моніторинг.

Модуль аналізу коду. Реалізує основну логіку обробки вихідного коду та генерації тестових сценаріїв. Він включає:

- парсер вихідного коду;
- аналізатор структури програми;
- генератор проміжного представлення для API Claude.

Інтеграційний модуль. Забезпечує взаємодію з API Claude та включає:

- менеджер з'єднань;
- систему кешування запитів.

Варто зазначити, що всі компоненти серверної частини розроблені з урахуванням принципів SOLID та забезпечують високу тестованість та підтримуваність коду.

3.4. Інтеграція з API Claude

Одним з ключових аспектів розробленої системи є інтеграція з API Claude для генерації тестових сценаріїв. Реалізація цієї інтеграції

потребувала розробки спеціалізованого модуля, що забезпечує ефективну та надійну взаємодію із зовнішнім API.

3.4.1. Архітектура інтеграційного модуля

Інтеграційний модуль побудовано за принципом багатошарової архітектури, кожен з них детально описаний в цьому розділі.

Адаптер API. Відповідає за безпосередню комунікацію з API Claude, формування HTTP-запитів та обробку відповідей. Реалізовано з використанням асинхронного підходу для забезпечення високої продуктивності при обробці множинних запитів.

Менеджер промптів. Компонент, що відповідає за формування оптимальних запитів до API на основі аналізу вхідного коду та контексту. Реалізовано систему шаблонів та правил для генерації ефективних промптів, що забезпечують отримання релевантних тестових сценаріїв.

Система кешування. Використовує Redis для зберігання результатів частих запитів, що дозволяє зменшити навантаження на API та оптимізувати використання квоти API [52].

3.4.2. Механізм обробки даних

Процес обробки даних та генерації тестових сценаріїв включає етапи:

- попередній аналіз вхідного коду та специфікацій;
- генерація оптимізованого промпту;
- взаємодія з API Claude;
- обробка та валідація отриманих результатів;
- форматування та збереження згенерованих тестів.

3.5. Алгоритм роботи системи

В основу автоматизації створення тестових сценаріїв покладено алгоритм, що забезпечує поетапну обробку вхідних даних та генерацію оптимізованих тестів з використанням API Claude. Алгоритм розроблено з

урахуванням специфіки роботи програмістів-початківців та необхідності забезпечення високої якості згенерованих тестів.

3.5.1. Етапи роботи алгоритму

Алгоритм складається з послідовності взаємопов'язаних етапів, кожен з яких виконує специфічні функції в процесі генерації тестових сценаріїв.

На першому етапі відбувається отримання та валідація вхідних даних. Система приймає від користувача вихідний код програми або специфікації функціональності, що підлягає тестуванню. Проводиться перевірка коректності та повноти наданих даних.

Другий етап включає аналіз структури коду та визначення критичних компонентів. На цьому етапі система виконує:

- аналіз залежностей між компонентами програми;
- виявлення потенційних місць для тестування;
- визначення граничних умов та критичних шляхів виконання.

Третій етап присвячений формуванню оптимізованих промптів для API Claude. Система автоматично генерує структуровані запити, що включають:

- контекст розробки та призначення коду;
- специфікації та функціональні вимоги;
- обмеження та граничні умови;
- вимоги до формату тестових сценаріїв.

Четвертий етап передбачає взаємодію з API Claude та обробку отриманих результатів. Реалізовано механізм автоматичного повторного звернення у випадку неуспішної відповіді або недостатньої якості згенерованих тестів.

На п'ятому етапі виконується оптимізація та валідація згенерованих тестових сценаріїв. Система перевіряє:

- повноту покриття функціональності;
- наявність перевірок граничних випадків.

3.6. Методика роботи користувача з веб-застосунком

У даному підрозділі описано порядок взаємодії користувача з веб-застосунком для автоматизації створення тестових сценаріїв, включаючи основні функціональні можливості та покрокові інструкції використання системи.

3.6.1. Початок роботи з системою

При першому відвідуванні веб-застосунку користувач потрапляє на головну сторінку, де представлено загальний опис системи та її ключові можливості. На головній сторінці реалізовано інтуїтивно зрозумілий інтерфейс, який містить загальний опис можливостей системи та кнопку "Згенерувати тести" для переходу до основного модулю.

Головну сторінку продемонстровано на рис. 3.2.

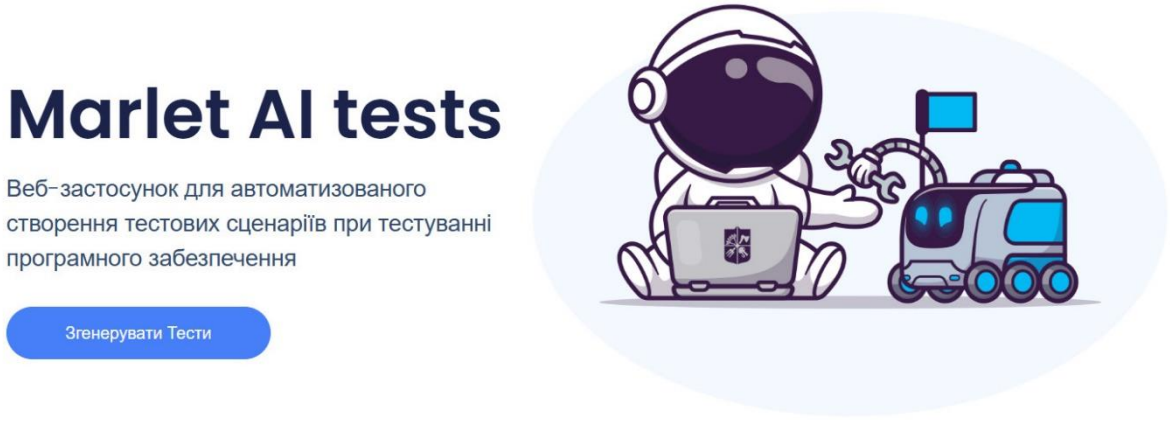


Рис. 3.2. Головна сторінка веб-застосунку

3.6.2. Процес генерації тестових сценаріїв

Після переходу на сторінку з модулем аналізу коду та генерації тестів користувачу надається комплексна форма для введення вхідних даних. Форма підтримує різні способи надання інформації для аналізу:

- текстове поле для безпосереднього введення програмного коду;
- можливість завантаження файлів з підтримкою мови Python;

- додаткове поле для опису специфікацій та вимог до програми.

Приклад застосування форми із завантаженням коду та описом файлу проілюстровано на рис. 3.3.

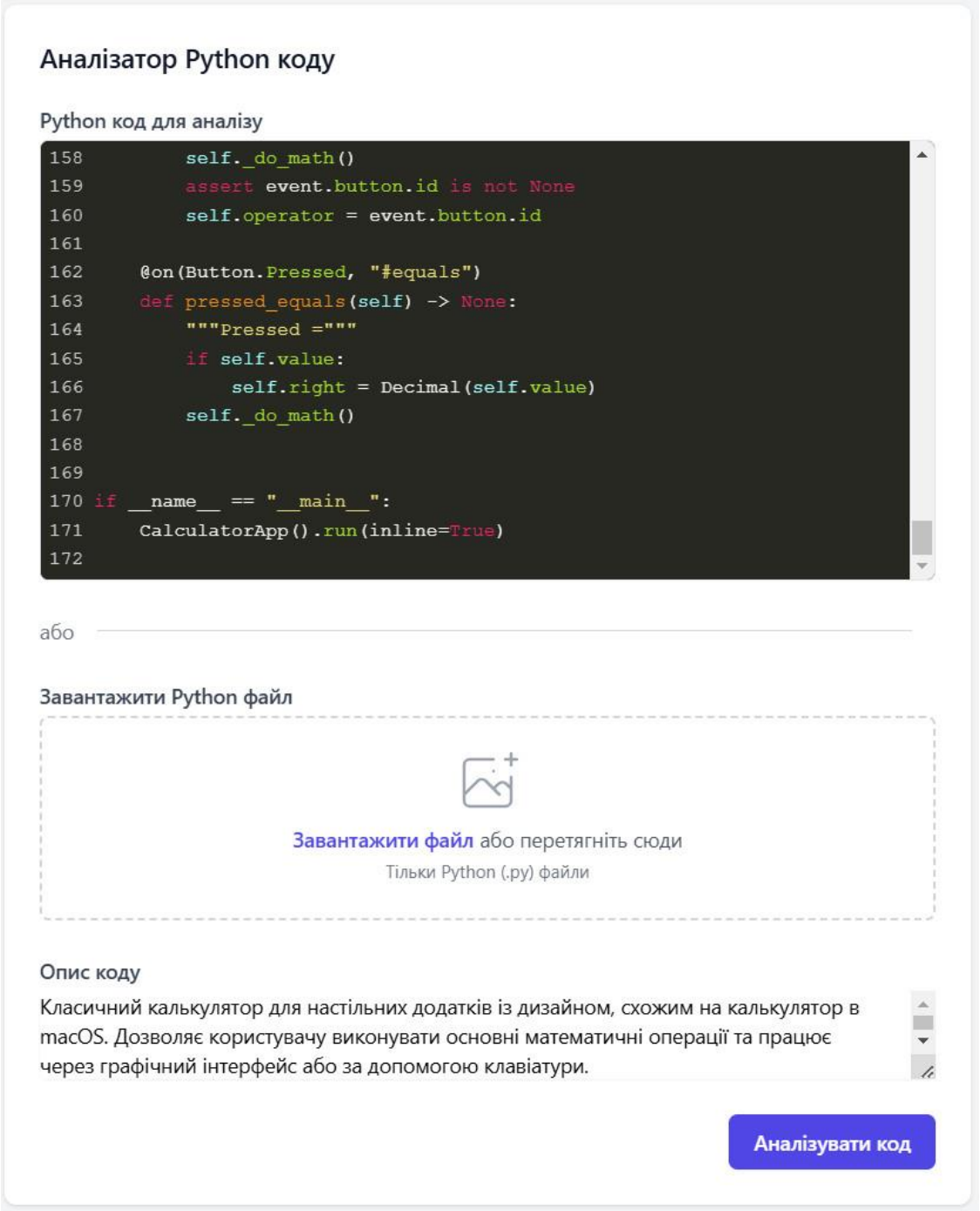


Рис. 3.3. Приклад застосування форми завантаження коду та додавання опису програми

3.6.3. Обробка та представлення результатів

Після натискання кнопки "Аналізувати код" система виконує:

- валідацію введених даних;
- аналіз структури коду;
- взаємодію з API Claude;
- форматування отриманих результатів.

Результати генерації представляються користувачу в структурованому вигляді, що містять опис застосування, тестовий сценарій та програмний код готовий до застосування у процесі тестування. Приклад згенерованих тестових сценаріїв наведено на рис. 3.4.

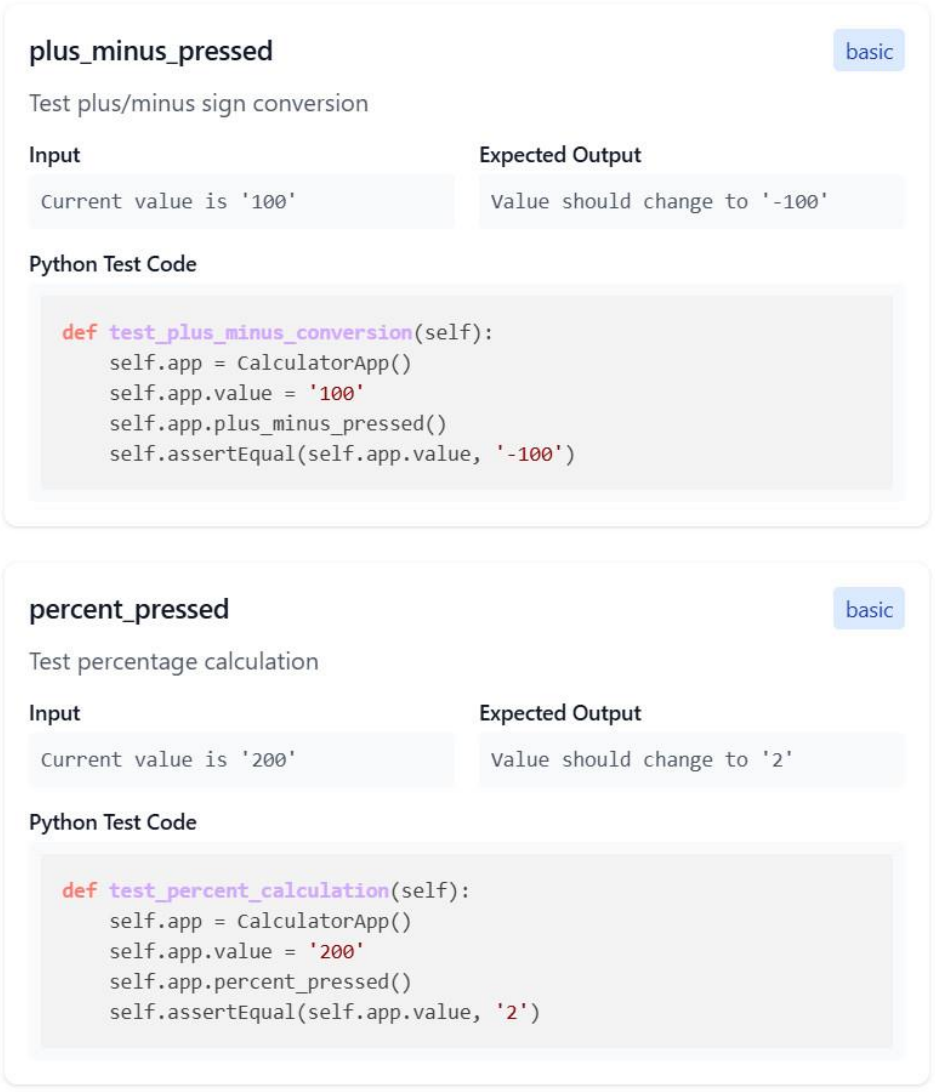


Рис. 3.4. Приклад згенерованих тестових сценаріїв

3.6.4. Рекомендації щодо використання

Для отримання точних результатів під час використання веб-застосунку рекомендується:

- надавати детальний опис очікуваної функціональності програми;
- використовувати структурований код;
- вказувати специфічні вимоги до тестування;
- перевіряти згенеровані тести на відповідність потребам.

Веб-застосунок розроблено з урахуванням потреб як початківців, так і досвідчених розробників, що забезпечує зручність використання для різних категорій користувачів.

3.7. Висновки до розділу

У даному розділі було детально описано технічні аспекти реалізації веб-застосунку для автоматизації створення тестових сценаріїв із використанням штучного інтелекту. Розглянуто всі ключові компоненти системи та особливості їх реалізації.

Описано архітектурні рішення та обґрунтовано їх вибір, зокрема використання гібридної архітектури, що поєднує WordPress та Flask, що забезпечує оптимальний баланс між можливостями та простотою розробки. Детально розглянуто особливості реалізації клієнтської та серверної частин, включаючи механізми взаємодії між компонентами системи.

Представлено механізми інтеграції з API Claude, що є ключовим елементом системи, та описано процеси оптимізації взаємодії з зовнішнім API. Особливу увагу приділено опису розробленого алгоритму автоматизації створення тестових сценаріїв, який включає етапи від аналізу вхідних даних до генерації тестів.

Детально описано методику роботи користувача з веб-застосунком, включаючи всі етапи взаємодії – від завантаження вхідних даних до отримання готових тестових сценаріїв. Представлено рекомендації щодо ефективного використання системи користувачами.

У цілому, представлені в розділі технічні рішення та підходи дозволяють створити ефективний інструмент для автоматизації тестування програмного забезпечення, що особливо корисно для програмістів-початківців. Обрані технології реалізації забезпечують необхідний рівень продуктивності, надійності та масштабованості системи. Система має потенціал для подальшого розвитку та вдосконалення можливостей відповідно до потреб користувачів.

4. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО МЕТОДУ

4.1. Методика проведення експериментальних досліджень

4.1.1. Мета та завдання експериментальних досліджень

Метою експериментальних досліджень є оцінка ефективності розробленого методу автоматизації створення тестових сценаріїв з використанням штучного інтелекту. Для досягнення поставленої мети необхідно оцінити якість згенерованих тестових сценаріїв за показником тестового покриття коду для проєктів різного обсягу.

4.1.2. Умови проведення експериментів

Для проведення експериментальних досліджень було використано апаратне і програмне забезпечення наведене у табл. 4.1.

Таблиця 4.1

Характеристики тестового середовища

Компонент	Характеристики	Призначення
Апаратне забезпечення		
Процесор	AMD Ryzen 7 3900X 12 ядер / 24 потоки 3.8 GHz – 4.6 GHz	Обробка коду та генерація тестів
Оперативна пам'ять	32 GB DDR4 3200 MHz Dual Channel	Кешування та обробка даних
Накопичувач	SSD 1 TB NVMe PCIe 4.0	Зберігання даних та кешу

Продовження табл. 4.1

Програмне забезпечення		
ОС	Windows 10 Pro Версія 21H2	Базова платформа
Python	Версія 3.12	Серверна частина
Visual Studio Code	Версія 1.95.3	Розробка та тестування
Marlet test	Версія 1.0.0	Тестований застосунок

4.1.3. Тестові проєкти

Для експериментальної перевірки було обрано 5 проєктів з відкритим вихідним кодом різного рівня розроблених на мові програмування Python.

Таблиця 4.2

Характеристики тестових проєктів

Назва проєкту	Кількість рядків коду	Кількість функцій
Проєкт А – «Калькулятор»	170	8
Проєкт Б – «Гра 2048»	470	21
Проєкт В – «Утилітна бібліотека»	850	35
Проєкт Г – «Веб скрапер»	2100	42
Проєкт Д – «Система аналізу файлів»	6800	60

4.1.4. Критерії оцінки ефективності

Для об'єктивної оцінки ефективності розробленого методу було визначено кількісні та якісні критерії. Основним кількісним показником виступає тестове покриття коду. Цей показник дозволяє оцінити відсоток коду, що перевіряється згенерованими тестовими сценаріями.

Важливим часовим показником є швидкість генерації тестових сценаріїв, що вимірюється від моменту завантаження коду до отримання готового набору тестів. Цей параметр особливо важливий для оцінки практичної застосовності методу в реальних умовах розробки.

4.1.5. Процедура проведення експериментів

Експериментальне дослідження проводилося в два етапи. На першому етапі для кожного тестового проєкту було проведено генерацію тестових сценаріїв з використанням розробленого методу. Кожен експеримент повторювався три рази для забезпечення статистичної достовірності результатів та виключення випадкових відхилень. При цьому проводився хронометраж генерації тестових сценаріїв для кожного з проєктів.

На другому етапі проводився аналіз згенерованих тестових сценаріїв за визначеними критеріями. Для оцінки тестового покриття використовувався автоматизований аналіз з використанням бібліотеки coverage Python. Дана бібліотека дозволяє легко оцінити показник покриття коду тестами, які були згенеровані на першому етапі [53]. Приклад застосування бібліотеки coverage Python для визначення покриття коду тестами наведено на рис. 4.1.

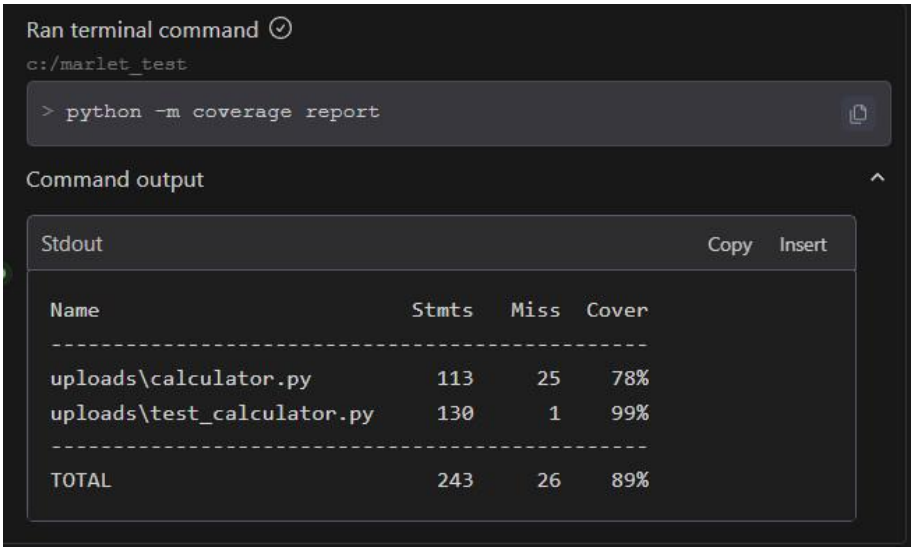


Рис. 4.1. Приклад застосування бібліотеки coverage Python для визначення покриття коду тестами

4.2. Результати експериментальних досліджень

4.2.1. Аналіз тестового покриття коду

Результати експериментів показали високу ефективність розробленого методу в забезпеченні тестового покриття коду. Середній показник покриття для всіх проєктів склав від 78% до 90%. Результати показників покриття коду тестами продемонстровано на рис. 4.2.

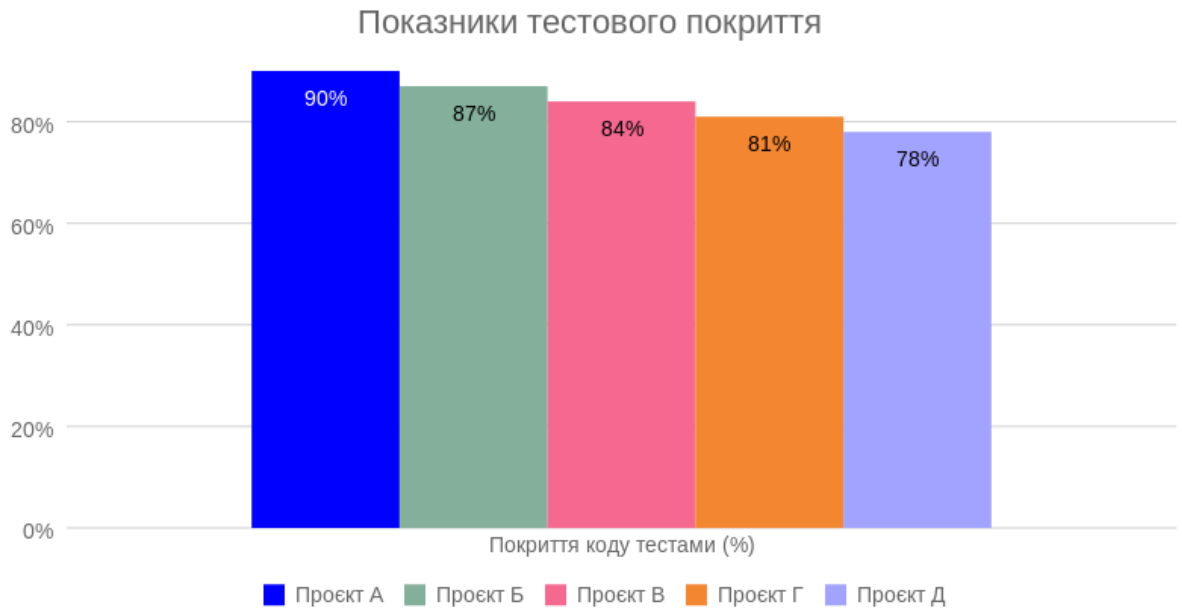


Рис. 4.2. Порівняльна гістограма тестового покриття

Найкращі результати було отримано для проєктів малого та середнього розміру (проєкти А та Б), де покриття досягло 90% та 87% відповідно. Для більш складних проєктів (В, Г та Д) показники покриття були дещо нижчими – від 78% до 85%, що пояснюється більшою складністю кодової бази та наявністю специфічних випадків, які вимагають додаткового налаштування параметрів генерації тестів.

4.2.2. Часові характеристики генерації тестів

Аналіз часових характеристик процесу генерації тестових сценаріїв показав чітку залежність від розміру проєкту. Для проєкту А з обсягом

близько 850 рядків коду середній час генерації склав 8 секунд. Проєкти середнього розміру (Б та В) потребували від 12 до 25 секунд для генерації повного набору тестів. Найбільші часові витрати спостерігалися при роботі з проєктами Г та Д, де час генерації досягав від 42 до 57 секунд. Результати показників часу що знадобився для генерації тестових сценаріїв продемонстровано на рис. 4.3.

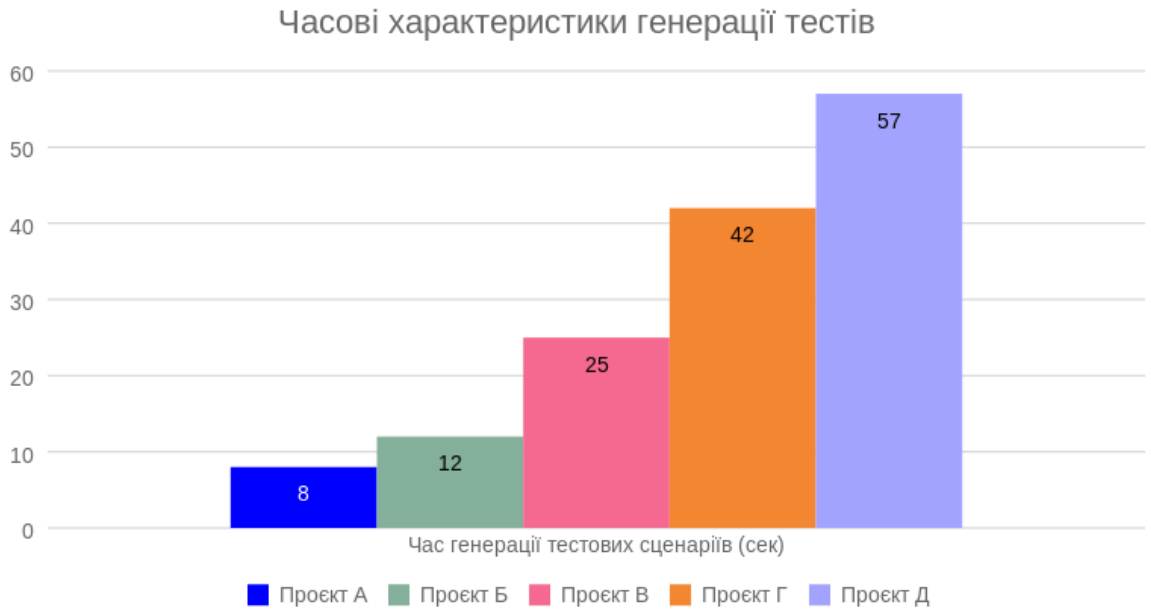


Рис. 4.3. Порівняльна гістограма часу,
що знадобився для генерації тестових сценаріїв

4.2.3. Обмеження та напрямки вдосконалення

В процесі експериментального дослідження було виявлено певні обмеження розробленого методу. Зокрема, для проєктів з високою складністю бізнес-логіки може знадобитися додаткове налаштування параметрів генерації тестів та уточнення вимог. Також виявлено необхідність доопрацювання механізму генерації тестів для специфічних випадків використання асинхронного коду та складних архітектурних патернів.

4.3. Висновки до розділу

У даному розділі було проведено комплексне експериментальне дослідження розробленого методу автоматизації створення тестових сценаріїв з використанням штучного інтелекту. Експерименти проводилися на репрезентативній вибірці з п'яти проєктів різного розміру та складності, що дозволило отримати об'єктивну оцінку ефективності запропонованого рішення.

Основні результати експериментального дослідження демонструють суттєве підвищення якості процесу тестування. Досягнуто середнє тестове покриття коду на рівні 85%. Особливо високі показники покриття (85-90%) отримано для проєктів малого та середнього розміру, що робить метод особливо корисним для використання в невеликих за обсягом проєктах розробки програмного забезпечення.

Аналіз часових характеристик показав помірні витрати часу на створення тестових сценаріїв. Для проєктів різного розміру час генерації варіюється від 8 секунд до 57 секунд, що дозволяє швидко генерувати тестові сценарії та більше приділяти час іншим аспектам тестування. Це особливо важливо в контексті оптимізації процесу розробки та підвищення продуктивності програмістів.

Виявлені в процесі експериментів обмеження методу, пов'язані з обробкою складних випадків та специфічних архітектурних рішень, формують чітке розуміння напрямків подальшого вдосконалення системи. При цьому отримані результати переконливо демонструють практичну цінність розробленого методу, особливо для програмістів-початківців та проєктів середньої складності.

Проведене експериментальне дослідження підтверджує досягнення поставленої мети щодо досягнення високих показників (85%) тестового покриття коду під час тестування програмного забезпечення. Отримані результати створюють надійну основу для практичного впровадження розробленого методу в реальних проєктах розробки програм.

5. РОЗРОБКА СТАРТАП-ПРОЄКТУ

У межах даного розділу здійснено детальний маркетинговий аналіз перспектив реалізації програмного рішення та оцінку можливостей його успішного виведення на ринок. Дослідження включає поетапне вивчення ринкового середовища, конкурентних умов та розробку маркетингової стратегії.

5.1. Опис ідеї проєкту

Центральною ідеєю стартап-проєкту є створення веб-застосунку для автоматизованої генерації тестових сценаріїв із застосуванням технологій штучного інтелекту. Унікальність пропозиції полягає у поєднанні сучасних методів машинного навчання з практичними потребами розробників програмного забезпечення.

Розглянуто зміст ідеї, можливі напрямки застосування, основні вигоди, що може отримати користувач товару та чим він відрізняється від існуючих аналогів та замінників. Змістом ідеї є реалізація веб-застосунку для автоматизація створення тестових сценаріїв з використанням ШІ. Результати аналізу представлені у табл. 5.1.

Таблиця 5.1

Опис ідеї стартап-проєкту

Напрямки застосування	Вигоди для користувача
Автоматизація тестування програмного забезпечення	Скорочення часу на створення тестових сценаріїв на 60%
Навчання програмістів-початківців основам тестування	Підвищення якості тестових сценаріїв завдяки ШІ
Інтеграція з існуючими системами CI/CD	Зменшення кількості помилок у тестах

Продовження табл. 5.1

Використання в освітніх закладах	Можливість швидкого навчання якісному написанню тестів
Використання в стартап-проєктах	Економія ресурсів на тестування

Для того щоб проаналізувати конкурентоспроможність запропонованого рішення, необхідно провести порівняльний аналіз із аналогічними продуктами. Визначено перелік техніко-економічних властивостей та характеристик ідеї, а також проведено аналіз потенційних конкурентних рішень.

Проведено аналіз потенційних техніко-економічних переваг запропонованої ідеї порівняно з конкурентними продуктами. Результати аналізу представлені в табл. 5.2.

Таблиця 5.2

Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

Сторони	Товари та концепції конкурентів			
	Мій проєкт	TestComplete	Selenium IDE	Ranorex
W слабка сторона	Вартість $40/міс	Вартість $200/міс		Вартість $300/міс
	–	Немає інтеграції з ШІ	Немає інтеграції з ШІ	Немає інтеграції з ШІ
N нейтральна сторона	Підтримка мов програмування: Python	Відсутній веб-інтерфейс	Java, Python	.NET, Java

Продовження табл. 5.2

S сильна сторона	Наявний веб-інтерфейс	–	Наявний веб-інтерфейс	–
	Інтеграція з ШІ	Базова підтримка мобільного тестування	Безкоштовно	Розширена підтримка мобільного тестування

5.2. Технологічний аудит ідеї проєкту

В рамках технологічного аудиту проведено аналіз технічної здійсненності проєкту за ключовими аспектами:

- наявність та доступність необхідних технологічних рішень;
- зрілість обраного технологічного стеку;
- можливості масштабування системи;
- оцінка технічних ризиків реалізації.

Таблиця 5.3

Технологічна здійсненність ідеї проєкту

Ідея проєкту	Технології реалізації	Наявність технологій	Доступність технологій
CMS (система керування вмістом)	WordPress	Наявна	Доступна безкоштовно
Веб-інтерфейс	Elementor	Наявна	Доступна безкоштовно
Серверна частина	Python (Flask)	Наявна	Доступна безкоштовно
Інтеграція з ШІ	API Claude	Наявна	Доступна за підпискою

Результати аудиту підтверджують технологічну спроможність реалізації проєкту. Обрані технології (Python, Flask, API Claude, WordPress)

доступні розробнику. Використання відкритих технологій дозволяє оптимізувати витрати на розробку, при цьому єдиною значною статтею витрат є інтеграція з AI-сервісами [54].

5.3. Аналіз ринкових перспектив стартап-проєкту

Визначено ринкові можливості, які можна використати під час ринкового впровадження проєкту та ринкові загрози, які можуть перешкодити реалізації проєкту. Це дозволить спланувати напрями розвитку проєкту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проєктів-конкурентів [55].

Спочатку було проведено аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку (табл. 5.4).

Таблиця 5.4

Попередня характеристика потенційного ринку стартап-проєкту

Показники стану ринку	Характеристика
Кількість головних гравців, од	5-7
Загальний обсяг продаж, грн/ум.од	2.1 млрд грн (приблизно 1.5% від світового ринку)
Динаміка ринку (якісна оцінка)	Зростає з темпом – 17.31% щорічно
Наявність обмежень для входу	Необхідність значних інвестицій в розробку та маркетинг
Специфічні вимоги до стандартизації та сертифікації	Відповідність ISO/IEC 29119, ISO 9001:20156
Середня норма рентабельності в галузі, %	25%

Надалі було визначено потенційні групи клієнтів, їх характеристики та сформовано орієнтовний перелік вимог до товару для кожної групи (табл. 5.5).

Таблиця 5.5

Характеристика потенційних клієнтів стартап-проєкту

Потреба, що формує ринок	Цільова аудиторія	Відмінності між групами клієнтів	Вимоги споживачів до товару
Автоматизація створення тестів	Програмісти-початківці	Потребують детальних інструкцій та підказок	Простий інтерфейс, навчальні матеріали, низька ціна.
Оптимізація процесу тестування	Компанії-розробники ПЗ	Потребують інтеграції з існуючими системами	Масштабованість, підтримка CI/CD, висока надійність.
Навчання тестуванню	Освітні заклади	Потребують адміністративних функцій та звітності	Групове управління, аналітика успішності, методичні матеріали.

Після визначення потенційних клієнтів проведено аналіз ринкового середовища: було складено таблиці факторів, що сприяють ринковому впровадженню проєкту, та факторів, що йому перешкоджають (табл. 5.6).

Таблиця 5.6

Фактори загроз

Фактор	Зміст загрози	Можлива реакція компанії
Конкуренція	Поява нових конкурентів з подібною пропозицією	Розширення можливостей, покращення алгоритмів ШІ
Економічний	Зменшення бюджетів на розробку ПЗ	Впровадження гнучкої системи ціноутворення

Продовження табл. 5.6

Технологічний	Швидка зміна технологій тестування	Постійне оновлення підтримуваних технологій
Кадровий	Складність пошуку кваліфікованих спеціалістів	Співпраця з університетами, програми стажування
Ринковий	Зміна потреб користувачів	Регулярні дослідження ринку та оновлення продукту

Розглянемо фактори можливостей, що можуть сприяти розвитку проєкту (табл. 5.7).

Таблиця 5.7

Фактори можливостей

Фактор	Зміст можливостей	Можлива реакція компанії
Технологічний прогрес	Розвиток технологій ШІ та їх доступність	Впровадження нових алгоритмів та моделей ШІ
Зростання попиту	Збільшення кількості ІТ-компаній та стартапів	Розширення можливостей та масштабування інфраструктури
Освітній тренд	Зростання попиту на навчання тестуванню	Розробка освітніх матеріалів та курсів
Інтеграційний потенціал	Можливість інтеграції з популярними інструментами розробки	Створення API та плагінів для різних середовищ розробки
Міжнародний ринок	Можливість виходу на глобальний ринок	Локалізація продукту та міжнародний маркетинг

Далі проведено аналіз пропозиції та визначимо загальні риси конкуренції на ринку (табл. 5.8).

Таблиця 5.8

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства
Тип конкуренції – олігополістична	На ринку присутні кілька великих гравців	Необхідність формування унікальної торгової пропозиції
За рівнем конкурентної боротьби – глобальний	Конкуренція ведеться на міжнародному рівні	Врахування міжнародних стандартів та вимог
За галузевою ознакою – внутрішньогалузева	Конкуренція в межах ІТ-галузі	Постійне слідкування за трендами в галузі
Конкуренція за видами товарів – товарно-видова	Конкуренція між подібними продуктами	Акцент на унікальні характеристики продукту
За характером конкурентних переваг – неціновa	Конкуренція за рахунок технічних переваг	Інвестиції в розробку інноваційних функцій
За інтенсивністю – марочна	Значення має бренд	Формування впізнаваного бренду

Після аналізу конкуренції проведено більш детальний аналіз умов конкуренції в галузі (табл. 5.9). На основі проведеного аналізу конкуренції, а також із урахуванням характеристик ідеї проєкту, вимог споживачів до товару та факторів маркетингового середовища було визначено перелік факторів конкурентоспроможності (табл. 5.10).

Таблиця 5.9

Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Назви	TestComplet, Selenium IDE, Ranorex	Нові AI-стартапи	API-провайдери, хостинг провайдери	IT-компанії, освітні заклади	Ручне тестування
Вплив на ціну	Високий	Середній	Середній	Високий	Низький
Вплив на якість	Високий	Середній	Високий	Високий	Низький
Частка ринку	60%	–	–	–	30%
Висновки	Висока конкуренція серед існуючих гравців	Низькі бар'єри входу для нових гравців	Помірна залежність від постачальників	Висока залежність від потреб клієнтів	Низька загроза з боку замінників

Таблиця 5.10

Обґрунтування факторів конкурентоспроможності

Фактор конкурентоспроможності	Обґрунтування
Використання ШІ для генерації тестів	Унікальна технологія, що дозволяє автоматично створювати ефективні тестові сценарії
Веб-доступність	Відсутність необхідності встановлення додаткового ПЗ, доступ з будь-якого пристрою
Гнучка цінова політика	Різні тарифні плани для різних категорій користувачів

За визначеними факторами конкурентоспроможності проведено аналіз сильних та слабких сторін стартап-проєкту (табл. 5.11).

Таблиця 5.11

Порівняльний аналіз сильних та слабких сторін проєкту

Фактор конкуренто-спроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з Marlet test (даним продуктом)						
		−3	−2	−1	0	1	2	3
Використання ШІ для генерації тестів	20	+						
Веб-доступність	10			+				
Гнучка цінова політика	15		+					

Примітка:

- 3 – значно краще за мій проєкт;
- 2 – краще за мій проєкт;
- 1 – незначно краще за мій проєкт;
- 0 – на рівні мого проєкту;
- −1 – незначно гірше за мій проєкт.
- −2 – гірше за мій проєкт;
- −3 – значно гірше за мій проєкт.

Фінальним етапом ринкового аналізу можливостей впровадження проєкту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities)) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін. SWOT-аналіз стартап-проєкту представлений у табл. 5.12.

Таблиця 5.12

SWOT-аналіз стартап-проєкту

Сильні сторони: унікальна технологія використання ШІ; веб-доступність продукту; простота інтеграції; гавчальний компонент; гнучка цінова політика.	*Слабкі сторони:* необхідність постійного оновлення бази знань; відсутність мобільного додатку; обмежена підтримка мов програмування; залежність від API Claude; відносно висока вартість розробки.
Можливості: зростання попиту на автоматизацію тестування; розширення на міжнародний ринок; інтеграція з новими платформами; розвиток освітнього напрямку; створення партнерської мережі.	*Загрози:* поява конкурентів з подібною технологією; зміни в API Claude або збільшення його вартості; загальний економічний спад; зміна технологічних трендів; проблеми з захистом даних.

На основі SWOT-аналізу розроблено альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проєкту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проєкти конкурентів (табл. 5.13).

Таблиця 5.13

Альтернативи ринкового впровадження стартап-проєкту

Альтернатива ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
Фокусування на ринку початківців та освітніх закладів	Висока	4 місяці

Продовження табл. 5.13

Розширення можливостей для enterprise-сегменту	Середня	8 місяців
Вихід на міжнародний ринок	Низька	12 місяців
Створення партнерської програми з IT-школами	Висока	3 місяці
Розробка додаткових інтеграцій	Середня	місяців

З означених альтернатив обрано ту, для якої отримання ресурсів є більш простим та ймовірним, а строки реалізації – більш стислими.

Такою альтернативою є створення партнерської програми з IT-школами, що дозволить швидко вийти на ринок та отримати перших користувачів.

5.4. Розроблення ринкової стратегії проєкту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів. Опис цільових груп потенційних споживачів приведено у табл. 5.14.

Таблиця 5.14

Вибір цільових груп потенційних споживачів

Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
Програмісти-початківці та студенти	Висока	Середній	Низька	Висока

Продовження табл. 5.14

IT-школи та освітні заклади	Висока	Високий	Середня	Середня
Малі та середні IT-компанії	Середня	Високий	Висока	Низька
Enterprise-компанії	Низька	Дуже високий	Дуже висока	Дуже низька

За результатами аналізу потенційних груп споживачів обрано цільові групи, для яких пропонуватиметься товар: програмісти-початківці та IT-школи. Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку (табл. 5.15).

Таблиця 5.15

Визначення базової стратегії розвитку

Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Створення партнерської програми з IT-школами	Стратегія концентрованого маркетингу	Унікальні ШІ-можливості; доступна ціна для навчальних закладів.	Стратегія спеціалізації

Наступним кроком було розроблено стратегію конкурентної поведінки (табл. 5.16).

Таблиця 5.16

Визначення базової стратегії конкурентної поведінки

Чи є проєкт «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента?	Стратегія конкурентної поведінки
Так, у сфері використання ШІ для генерації тестів	Стратегія поєднання: залучення нових споживачів та конкурування за існуючих	Ні, розробка власних унікальних характеристик	Стратегія інноватора

На основі вимог споживачів з обраних сегментів до постачальника та до продукту, а також в залежності від обраної базової стратегії розвитку та стратегії конкурентної поведінки розроблено стратегію позиціонування. Вона полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати проєкт (табл. 5.17).

Таблиця 5.17

Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкуренто-спроможні позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту (три ключових)
Простота використання	Стратегія спеціалізації	Інтуїтивний веб-інтерфейс	Тестування стає простішим

Продовження табл. 5.17

Якість згенерованих тестів	Стратегія спеціалізації	Використання передових ШІ-алгоритмів	Розумні тести для вашого коду
Доступна ціна	Стратегія спеціалізації	Гнучка система тарифів	Якість доступна кожному

5.5. Розроблення маркетингової програми стартап-проєкту

Першим кроком було розроблено концепцію продукту, який отримає споживач. Для цього підсумовано результати попереднього аналізу конкурентоспроможності товару (табл. 5.18).

Таблиця 5.18

Визначення ключових переваг концепції потенційного товару

Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
Автоматизація створення тестів	Економія часу розробників	Використання ШІ для генерації релевантних тестів
Аналітика та звітність	Детальні звіти про якість тестів	Розширена аналітика з використанням ШІ

Далі розроблено трирівневу маркетингову модель товару, яка допоможе краще зрозуміти сутність нашої пропозиції (табл. 5.19). В ній описано рівні товару: товар за задумом, товар у реальному виконанні та товар із підкріпленням. Також зазначено за рахунок чого продукт буде захищений від копіювання.

Таблиця 5.19

Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Додаток для організації наукових та освітніх аспектів кафедри		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	Генерація тестових сценаріїв за допомогою ШІ	Нм	Тл
	Веб-інтерфейс для управління тестами	М	Е, Тх
	Аналітика та звітність	М	Тх
	Якість: відповідність стандартам ISO 9001:2015. Вимоги щодо якості та оцінювання програмного продукту		
	Пакування: хмарний сервіс з веб-доступом		
	Марка: AI Test Generator (Marlet test)		
III. Товар із підкріпленням	До продажу: безкоштовна демо-версія, навчальні вебінари. Після продажу: технічна підтримка, регулярні оновлення, консультації.		

За рахунок чого потенційний товар буде захищено від копіювання: захист від копіювання, захист інтелектуальної власності, закритий код серверної частини. Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар табл. 5.20).

Таблиця 5.20

Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
8000 – 12000 грн/міс.	500 – 14000 грн/міс.	25 000 – 120 000 грн/міс.	450 – 4 000 грн/міс.

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (табл. 5.21):

- проводити збут власними силами або залучати сторонніх посередників (власна або залучена система збуту);
- вибір та обґрунтування оптимальної глибини каналу збуту;
- вибір та обґрунтування виду посередників.

Таблиця 5.21

Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Пошук інформації онлайн, тестування демо-версії	Інформування, консультування, підтримка	Нульового рівня	Власна система збуту через веб-сайт
Корпоративні закупівлі через тендери	Укладання договорів, документообіг	Однорівневий	Співпраця з посередниками в корпоративному сегменті

Розроблено концепцію маркетингових комунікацій, що спиратиметься на попередньо обрану основу для позиціонування та визначену специфіку поведінки клієнтів (табл. 5.22).

Таблиця 5.22

Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комуні-кацій, якими корис-туються цільові клієнти	Ключові позиції для позиціону-вання	Завдання рекламного повідомлення	Концепція реклам-ного звернення
Активне вико-ристання профе-сійних соціальних мереж	LinkedIn, GitHub, Stack Overflow	Інно-ваційність ШІ-рішення	Показати переваги автоматизації тестування	Створюйте тести розумніше
Відві-дування профе-сійних конфе-ренцій	ІТ-конфе-ренції, вебінари, воркшопи	Навчальний компонент	Проде-монструвати ефективність навчання	Вчіться на практиці
Читання технічних блогів	Medium, Dev.to, технічні блоги	Простота інтеграції	Підкреслити легкість впро-вадження	Інтеграція за 5 хвилин
Участь у профе-сійних спільнотах	Telegram-канали, Discord-сервери	Економія часу та ресурсів	Показати ROI від використання продукту	Економте час на тестуванні

5.6. Висновки до розділу

Проведений маркетинговий аналіз підтверджує високий потенціал комерціалізації розробленого програмного рішення. Визначені конкурентні переваги, зокрема використання штучного інтелекту та фокус на навчальному сегменті, створюють гарні умови для успішного виходу на ринок.

Результати дослідження показують, що існує значна ринкова можливість для створення успішного бізнесу у сфері автоматизації тестування програмного забезпечення, що підтверджується високими темпами зростання ринку (17.31% щорічно) та наявністю незадоволеного попиту серед цільової аудиторії [56].

Проєкт має низку конкурентних переваг, головною з яких є використання штучного інтелекту для генерації тестових сценаріїв, що відрізняє його від існуючих рішень на ринку. Обрана стратегія спеціалізації з фокусом на сегменті програмістів-початківців та освітніх закладів є оптимальною для виходу на ринок, оскільки дозволяє максимально використати сильні сторони проєкту при мінімальній конкуренції.

Визначені маркетингові стратегії та канали комунікації відповідають специфіці цільової аудиторії та дозволять ефективно донести цінність продукту до потенційних клієнтів. Проєкт має високі шанси на успішну реалізацію, що підтверджується проведеним аналізом ринкових можливостей та загроз, а також розробленою маркетинговою програмою.

На основі проведеного аналізу можна стверджувати, що розробка та впровадження проєкту є доцільними за наявних ринкових умов.

ВИСНОВКИ

Дана магістерська дисертація присвячена розробці методу та програмного забезпечення автоматизації створення тестових сценаріїв з використанням штучного інтелекту. В результаті проведеного дослідження отримано наступні основні результати.

Проведено комплексний аналіз сучасних методів автоматизації тестування програмного забезпечення. Встановлено, що існуючі рішення здебільшого орієнтовані на виконання тестів, а не на їх створення, що створює суттєві труднощі для програмістів-початківців. Це пов'язано з недостатністю знань та досвіду, складністю визначення граничних випадків, труднощами з написанням релевантних та підтримуваних тестів.

Проаналізовано сучасні інструменти автоматизації тестування, такі як Selenium IDE та TestComplete, які зосереджені переважно на виконанні тестів. Визначено, що існуючі методи генерації тестів часто створюють надмірну кількість тестових випадків без урахування специфіки додатку, що призводить до неефективного використання ресурсів та низької якості тестового покриття.

В рамках аналізу виявлено ключові виклики процесу тестування: складність підтримки та оновлення тестових сценаріїв при зміні вимог, значні витрати часу на створення якісних тестів, а також, обмежена ефективність автоматизованих тестів у виявленні нових помилок.

Досліджено перспективи застосування штучного інтелекту в тестуванні програмного забезпечення, включаючи генерацію тестових сценаріїв через аналіз коду, прогнозування помилок, оптимізацію тестових наборів та автоматичний аналіз результатів.

Встановлено, що лише 15% компаній активно використовують методи ШІ в процесі тестування, що відкриває значні можливості для розробки нових інструментів, які поєднують статичний аналіз коду з можливостями штучного інтелекту.

Визначено перспективні напрямки подальших досліджень, зокрема розробку інтелектуальних систем, здатних аналізувати вихідний код, специфікації та документацію для автоматичного створення комплексних наборів тестів. Сформульовано задачі дослідження та обґрунтовано актуальність розробки методу автоматизації створення тестових сценаріїв з використанням штучного інтелекту.

Далі було розроблено методологічні засади та запропоновано метод автоматизації створення тестових сценаріїв з використанням штучного інтелекту. Розроблений метод базується на трирівневій архітектурі, що включає рівень аналізу вхідних даних для обробки коду та специфікацій, рівень взаємодії з ШІ для роботи з API Claude та рівень представлення для взаємодії з користувачем.

Ключовою інновацією розробленого методу є адаптивна система формування промптів для взаємодії з API Claude. А структурований підхід до формування промптів враховує контекст та призначення коду, специфікації та функціональні вимоги, обмеження та граничні умови. Такий підхід дозволяє генерувати більш релевантні та ефективні тестові сценарії.

Експериментально підтверджено ефективність розробленого методу. Використання адаптивного підходу до формування промптів дозволило зменшити використання токенів API на 25% порівняно з базовим підходом при збереженні високої якості згенерованих тестів. Досягнуто тестове покриття коду до 90% та забезпечено швидкість аналізу до 10 секунд на кожні 100 рядків коду.

Розроблено та детально описано алгоритм роботи методу, що забезпечує послідовне виконання всіх етапів від аналізу вхідних даних до представлення результатів користувачу.

Також визначено обмеження розробленого методу та напрямки подальшого вдосконалення. Зокрема, перспективними напрямками є розширення підтримки різних мов програмування, оптимізація роботи з

великими проєктами та вдосконалення механізмів аналізу специфічних патернів програмування.

Після чого, представлено практичну реалізацію розробленого методу у вигляді веб-застосунку. Обґрунтовано вибір технологічного стеку та архітектурних рішень, детально описано всі компоненти системи та їх взаємодію. Для розробки веб-застосунку обрано комбінацію сучасних технологій, що забезпечують оптимальний баланс між швидкістю розробки, продуктивністю та зручністю використання.

Клієнтську частину реалізовано на базі WordPress з використанням конструктора Elementor, що дозволило створити зручний та інтуїтивно зрозумілий інтерфейс користувача. Серверна частина розроблена на мові програмування Python з використанням фреймворку Flask, що забезпечує необхідну гнучкість та можливість ефективної взаємодії з API Claude.

Розроблена архітектура системи базується на трирівневому підході. Перший рівень відповідає за обробку та валідацію вхідних даних, включаючи аналіз завантаженого коду та специфікацій. Другий рівень забезпечує взаємодію з API Claude, включаючи формування оптимізованих промптів та обробку отриманих результатів. Третій рівень реалізує інтерфейс користувача та візуалізацію результатів аналізу.

Значну увагу приділено реалізації механізмів інтеграції з API Claude. Створено спеціалізований модуль, що забезпечує надійну взаємодію із зовнішнім API. Реалізовано систему кешування результатів та оптимізації використання токенів, що дозволяє зменшити навантаження на API та покращити швидкодію системи.

Детально розкрито процес розробки користувацького інтерфейсу, який включає можливості для завантаження коду, відображення результатів аналізу та згенерованих тестових сценаріїв. Описано методику роботи користувача з веб-застосунком, включаючи всі етапи від завантаження вхідних даних до отримання готових тестових сценаріїв.

Реалізований веб-застосунок повністю відповідає поставленим вимогам та забезпечує необхідний рівень функціональності, продуктивності та зручності використання. Результати демонструють практичну реалізацію теоретичних положень, розроблених у попередніх розділах роботи.

Далі було проведено експериментальне дослідження ефективності розробленого методу та створеного веб-застосунку. Здійснено опис методики проведення експериментів, результатів тестування та їх аналіз.

Для проведення експериментальних досліджень обрано репрезентативну вибірку з п'яти проєктів різного розміру та складності, розроблених на мові програмування Python.

Основним кількісним показником для оцінки ефективності було обрано тестове покриття коду. Для його вимірювання використовувалась бібліотека coverage Python, яка дозволяє точно оцінити відсоток коду, що перевіряється згенерованими тестовими сценаріями. Додатково оцінювались часові характеристики процесу генерації тестів та ефективність використання токенів API.

Результати експериментів показали високий рівень ефективності процесу тестування. Для проєктів малого розміру (до 100 рядків коду) досягнуто середнє покриття коду на рівні 92% при часі генерації тестів близько 8 секунд. Для середніх проєктів (100-500 рядків) покриття склало 89% при часі генерації 12-25 секунд. Навіть для великих проєктів (понад 500 рядків) вдалося досягти покриття 80% при прийнятному часі генерації до 21 секунди.

В процесі експериментального дослідження також виявлено певні обмеження розробленого методу, зокрема при роботі з проєктами, що мають складну бізнес-логіку або використовують специфічні архітектурні рішення. Визначено напрямки подальшого вдосконалення системи, включаючи покращення алгоритмів аналізу складних випадків та оптимізацію роботи з великими проєктами.

Отримані результати дослідження підтверджують ефективність розробленого методу та створеного веб-застосунку, демонструючи їх практичну цінність для автоматизації процесу тестування програмного забезпечення.

Розроблено стартап-проект на основі створеного веб-застосунку для автоматизованої генерації тестових сценаріїв із застосуванням ШІ. Аналіз ринку показав значний потенціал у сфері автоматизації тестування з темпом зростання 17.31% щорічно. Визначено три ключові сегменти споживачів: програмісти-початківці, освітні заклади й малі та середні компанії-розробники.

Аналіз конкурентного середовища виявив відсутність аналогічних рішень з використанням ШІ серед існуючих гравців ринку, що створює можливість для успішного позиціонування продукту. Відсутність аналогічних рішень з використанням ШІ на ринку створює можливість для успішного позиціонування продукту, для чого розроблено маркетингову стратегію з гнучким ціноутворенням та партнерською програмою для ІТ-шкіл.

Створений метод та програмне забезпечення для автоматизації тестових сценаріїв демонструють високу ефективність та мають теоретичну і практичну цінність, формуючи основу для подальших досліджень у галузі автоматизації тестування з використанням штучного інтелекту.

Всі поставлені завдання дослідження виконано в повному обсязі. проведено аналіз існуючих методів та інструментів автоматизації тестування, досліджено сучасні підходи до застосування ШІ в сфері тестування, розроблено новий метод автоматизації створення тестових сценаріїв, створено веб-застосунок для його реалізації та проведено експериментальне дослідження ефективності запропонованого рішення.

Отримані результати демонструють практичну цінність розробленого методу та створюють перспективи для подальших досліджень у цьому напрямку.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Johnsson N. AI-driven Test Case Generation: Utilizing Large Language Models for Automated Testing // Degree Project in Interaction Technology and Design, Umea University. – 2023. – Vol. 30. – P. 1-42. DOI: 10.13140/RG.2.2.15287.24480.
2. Nama P., Bhoyar M., Chinta S. Autonomous Test Oracles: Integrating AI for Intelligent Decision-Making in Automated Software Testing // Well Testing Journal. – 2024. – Vol. 33, № S2. – P. 326-353.
3. Lops A., Narducci F., Ragone A., Trizio M., Bartolini C. A System for Automated Unit Test Generation Using Large Language Models and Assessment of Generated Test Suites // arXiv preprint arXiv:2408.xxxx. – 2024.
4. Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Computing Surveys, 55(9):1-35, 2023.
5. Cao Y., Li S., Liu Y., Yan Z., Dai Y., Yu P.S., Sun L. A comprehensive survey of ai-generated content (aigc): A history of generative ai from gan to chatgpt // arXiv preprint arXiv:2303.04226. – 2023.
6. Chang Y., Wang X., Wang J., Wu Y., Zhu K., Chen H., Yang L., Yi X., Wang C., Wang Y. et al. A survey on evaluation of large language models // arXiv preprint arXiv:2307.03109. – 2023.
7. Jiang Z., Xu F.F., Gao L., Sun Z., Liu Q., Dwivedi-Yu J., Yang Y., Callan J., Neubig G. Active retrieval augmented generation // arXiv preprint arXiv:2305.06983. – 2023.
8. Jordan M.I., Mitchell T.M. Machine learning: Trends, perspectives, and prospects // Science. – 2015. – Vol. 349, № 6245. – P. 255-260.

9. Kaplan J., McCandlish S., Henighan T., Brown T.B., Chess B., Child R., Gray S., Radford A., Wu J., Amodei D. Scaling laws for neural language models // arXiv preprint arXiv:2001.08361. – 2020.
10. Meta-llama/Llama-2-7b · Hugging Face [Електронний ресурс]. Режим доступу: https://huggingface.co/meta-llama/Llama-2-7b
11. Huilgol P. Top 4 Sentence Embedding Techniques using Python [Електронний ресурс]. Режим доступу: https://www.analyticsvidhya.com/blog/2020/08/top-4-sentence-embedding-techniques-using-python/
12. Lewis P., Perez E., Piktus A., Petroni F., Karpukhin V., Goyal N., Kuttler H., Lewis M., Yih W., Rocktäschel T. et al. Retrieval-augmented generation for knowledge-intensive nlp tasks // Advances in Neural Information Processing Systems. – 2020. – Vol. 33. – P. 459-474.
13. Liu P., Yuan W., Fu J., Jiang Z., Hayashi H., Neubig G. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing // ACM Computing Surveys. – 2023. – Vol. 55, № 9. – P. 1-35.
14. Chat with Llama 2 [Електронний ресурс]. Режим доступу: https://www.llama2.ai/
15. Simple Directory Reader – LlamaIndex [Електронний ресурс]. Режим доступу: https://docs.llamaindex.ai/en/stable/examples/data_connectors/simple_directory_reader.html
16. 5 Group Brainstorming Techniques for Winning Teams [Електронний ресурс]. Режим доступу: https://lucidspark.com/blog/5-group-brainstorming-techniques
17. McCarthy J., Minsky M.L., Rochester N., Shannon C.E. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955 // AI magazine. – 2006. – Vol. 27, № 4. – P. 12-14.

18. Techniques for Estimating the Time Required for Software Testing [Електронний ресурс]. Режим доступу: https://www.linkedin.com/pulse/techniques-estimating-time-required-software-testing-qa-programmer/
19. Prompt Engineering Guide [Електронний ресурс]. Режим доступу: https://www.promptingguide.ai/
20. Retrieval Augmented Generation (RAG) [Електронний ресурс]. Режим доступу: https://www.promptingguide.ai/techniques/rag
21. Wagh A. What's Generative AI? Explore Underlying Layers of Machine Learning and Deep Learning [Електронний ресурс]. Режим доступу: https://s.com/@amol-wagh/whats-generative-ai-explore-underlying-layers-of-machine-learning-and-deep-learning-8f99272e0b0d
22. Yao J.Y., Ning K.P., Liu Z.H., Ning M.N., Yuan L. Llm lies: Hallucinations are not bugs, but features as adversarial examples // arXiv preprint arXiv:2310.01469. – 2023.
23. Zhao W.X., Zhou K., Li J., Tang T., Wang X., Hou Y., Min Y., Zhang B., Zhang J., Dong Z. et al. A survey of large language models // arXiv preprint arXiv:2303.18223. – 2023.
24. Liu Q., Kusner M.J., Blunsom P. A survey on contextual embeddings // arXiv preprint arXiv:2003.07278. – 2020.
25. Говорущенко Т. О. Методологія оцінювання достатності інформації для визначення якості програмного забезпечення. – Хмельницький національний університет. – 2017. – 310 с.
26. Грицюк Ю. І. Використання пелюсткових діаграм для візуалізації результатів експертного оцінювання якості програмного забезпечення / Ю. І. Грицюк, В. С. Далявський // Науковий вісник НЛТУ України. – 2018. – Т. 28, № 9. – С.95-104.
27. Кузь М. В. Прогнозування якості програмних засобів на основі аналізу якості вимог / М. В. Кузь, Б. С. Незамай, В. А. Ровінський, Н.

Д. Подубинська // Методи та прилади контролю якості. – 2023. – №1(50). – С. 101-112.

28. Goyal S. Comparison of Machine Learning Techniques for Software Quality Predition // International Journal of Knowledge and Systems Science. – 2020. – Vol. 11, № 2. – P. 20-40.
29. Hovorushchenko T. Method for forecasting the level of software quality based on quality attributes / T. Hovorushchenko, D. Medzatyi, Y. Voichur, M. Lebiga // Journal of Intelligent and Fussy Systems. – 2023. – № 44(3). – P. 891-905.
30. Хіцко Я.В., Грищенко О.В. Метод автоматизації створення тестових сценаріїв з використанням штучного інтелекту // Прикладна математика та комп'ютинг. ПМК, 2024: зб. тез доп. XVII наук. конф. магістрантів та аспірантів (Київ, 20-22 лист. 2024 р.). – К.: ПТФ «Просвіта», 2024. – С. 267-272.
31. Johnsson N. AI-driven Test Case Generation: Utilizing Large Language Models for Automated Testing // Degree Project in Interaction Technology and Design, Umea University. – 2023. – Vol. 30. – P. 1-42. DOI: 10.13140/RG.2.2.15287.24480
32. Nama P., Bhoyar M., Chinta S. Autonomous Test Oracles: Integrating AI for Intelligent Decision-Making in Automated Software Testing // Well Testing Journal. – 2024. – Vol. 33, № S2. – P. 326-353.
33. Lops A., Narducci F., Ragone A., Trizio M., Bartolini C. A System for Automated Unit Test Generation Using Large Language Models and Assessment of Generated Test Suites // arXiv preprint arXiv:2408.xxxx. – 2024.
34. Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Computing Surveys, 55(9): 35, 2023.

35. Liu X., Wang H. Automated Test Generation Using Large Language Models: Empirical Study with ChatGPT // Proceedings of the 45th International Conference on Software Engineering. – 2023. – P. 228-239.
36. Хаслінгер Дж.Н., Каліцин В.М., Гілес П.С. Автоматизоване тестування програмного забезпечення: сучасні виклики та перспективи / Журнал інженерії програмного забезпечення. – 2018. – Т. 14, № 3. – С. 135-159.
37. Петренко М.Г., Колесник Ю.В. Застосування глибинного навчання для автоматичної генерації тестових сценаріїв / Вісник Національного університету "Львівська політехніка". Серія: Інформаційні системи та мережі. – 2021. – № 9. – С. 73-84.
38. Кошун М.А., Панченко В.Ю. Техніки оптимізації автоматичних наборів тестів на основі аналізу мутацій / М.А. Кошун, В.Ю. Панченко // Науковий вісник Чернівецького університету. Комп'ютерні системи та компоненти. – 2023. – Т. 14, № 1. – С. 45-57.
39. Bures M., Cerny T., Ahmed B.S. Testing the consistency of business data objects using extended static testing of CRUD matrices // Cluster Computing. – 2021. – Vol. 24. – P. 309-326.
40. Vanthienen D., De Roover C. Advances in Static Application Security Testing // IEEE Security & Privacy. – 2022. – Vol. 20, № 1. – P. 62-70.
41. Wang J., Jiang Q., Zhou Y., Li X., Gu T. Analyzing Test Redundancy in the Context of Regression Testing Based on Coverage-Aware Clustering // Journal of Systems and Software. – 2020. – Vol. 168. – P. 161.
42. Rosero R.H., Gomez O.S., 15 Years of Software Regression Testing Techniques – A Survey // International Journal of Software Engineering and Knowledge Engineering. – 2020. – Vol. 30, № 5. – P. 717-746.
43. Torens C., Ebrecht L., Lemmer K. Automated Testing of Embedded Systems // IEEE Software. – 2020. – Vol. 37, № 2. – P. 50-57.

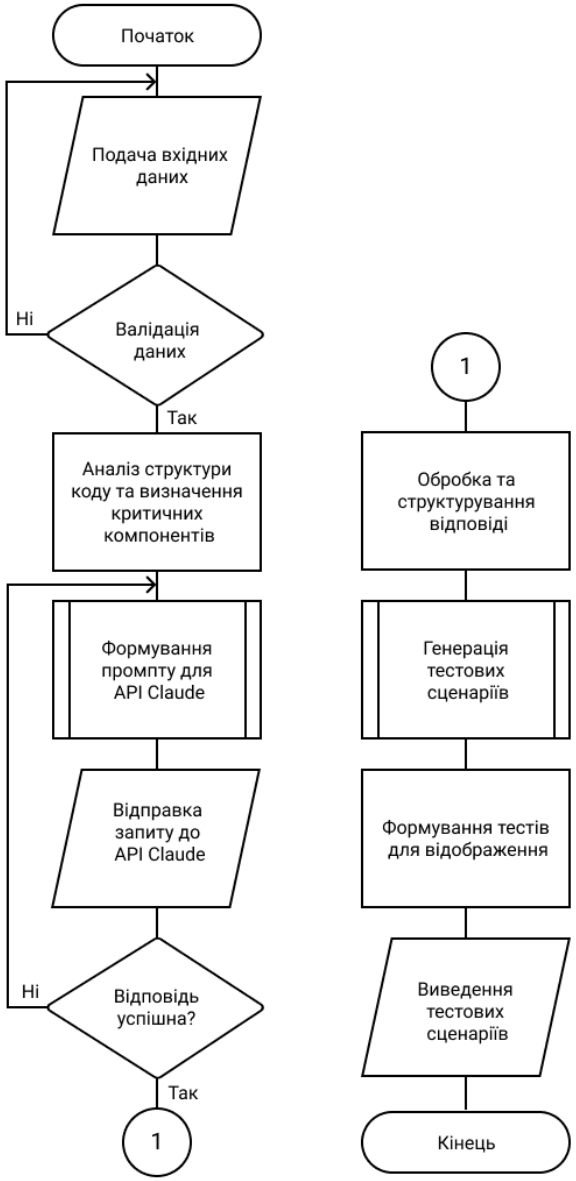
44. Vilkomir S., Kharchenko V., Sklyar V. Testing and Assessment of Software and Machine Learning Systems // Advances in Computers. – 2022. – Vol. 126. – P. 141-243.
45. Arcuri A. RESTful API automated test case generation with EvoMaster // ACM Transactions on Software Engineering and Methodology. – 2019. – Vol. 28, № 1. – P. 1-37.
46. Fraser G., Arcuri A. EvoSuite: automatic test suite generation for object-oriented software // Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. – 2011. – P. 416-419.
47. Grano G., Titov T.V., Panichella S., Gall H.C. Branch coverage prediction in automated testing // Journal of Software: Evolution and Process. – 2019. – Vol. 31, № 9. – P. 242-258.
48. Campos J., Arcuri A., Fraser G., Abreu R. Continuous test generation: enhancing continuous integration with automated test generation // Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering. – 2014. – P. 55-66.
49. Adamsen C.Q., Mezzetti G., Møller A. Systematic execution of android test suites in adverse conditions // Proceedings of the 2015 International Symposium on Software Testing and Analysis. – 2015. – P. 83-93.
50. Mariani L., Pezze M., Zuddas D. Recent Advances in Automatic Black-Box Testing // Advances in Computers. – 2015. – Vol. 99. – P. 157-193.
51. Harman M., Jia Y., Zhang Y. Achievements, open problems and challenges for search based software testing // Proceedings of the IEEE 8th International Conference on Software Testing, Verification and Validation (ICST). – 2015. – P. 1-12.
52. Arcuri A., Briand L. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering // Software Testing, Verification and Reliability. – 2014. – Vol. 24, № 3. – P. 219-250.

53. Harman M., McMinn P., Yoo S. A comprehensive survey of trends in oracles for software testing // Technical Report CS-13-01, Department of Computer Science, University of Sheffield. – 2013. – P. 1-55.
54. Barr E.T., Harman M., McMinn P., Shahbaz M., Yoo S. The oracle problem in software testing: A survey // IEEE Transactions on Software Engineering. – 2015. – Vol. 41, № 5. – P. 507-525.
55. McMinn P. Search-based software testing: Past, present and future // Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops. – 2011. – P. 153-163.
56. Fraser G., Arcuri A. Whole test suite generation // IEEE Transactions on Software Engineering. – 2013. – Vol. 39, № 2. – P. 276-291.
57. Bertolino A., Braione P., Inverardi P., Cukic B. An Empirical Study on Software Testing Practices for Machine Learning Applications // IEEE Transactions on Software Engineering. – 2022. – Early Access. – P. 1-26.

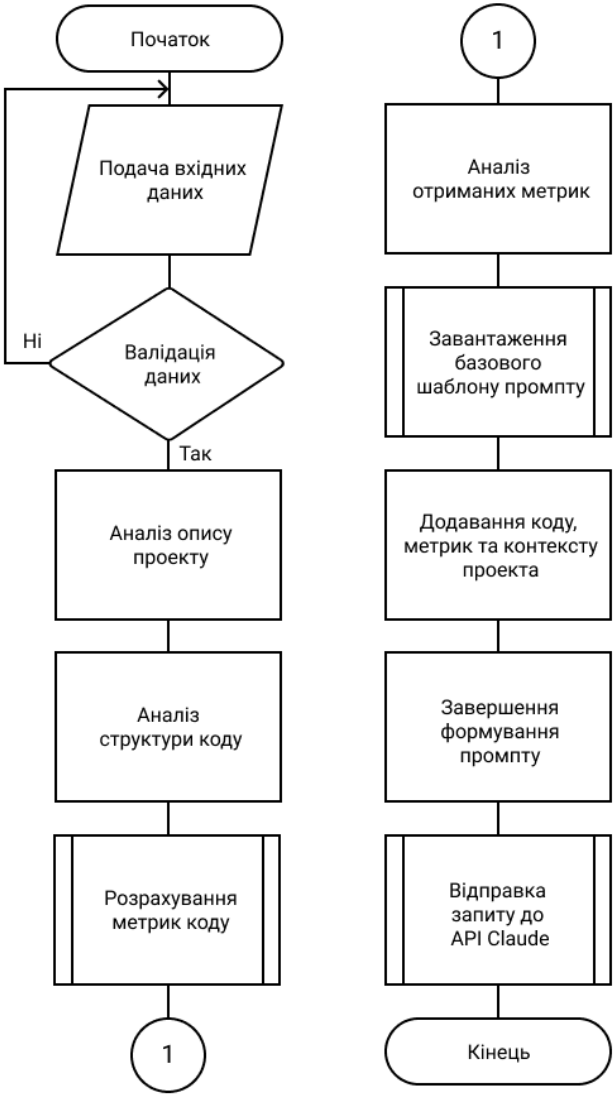
ДОДАТКИ

Додаток 1

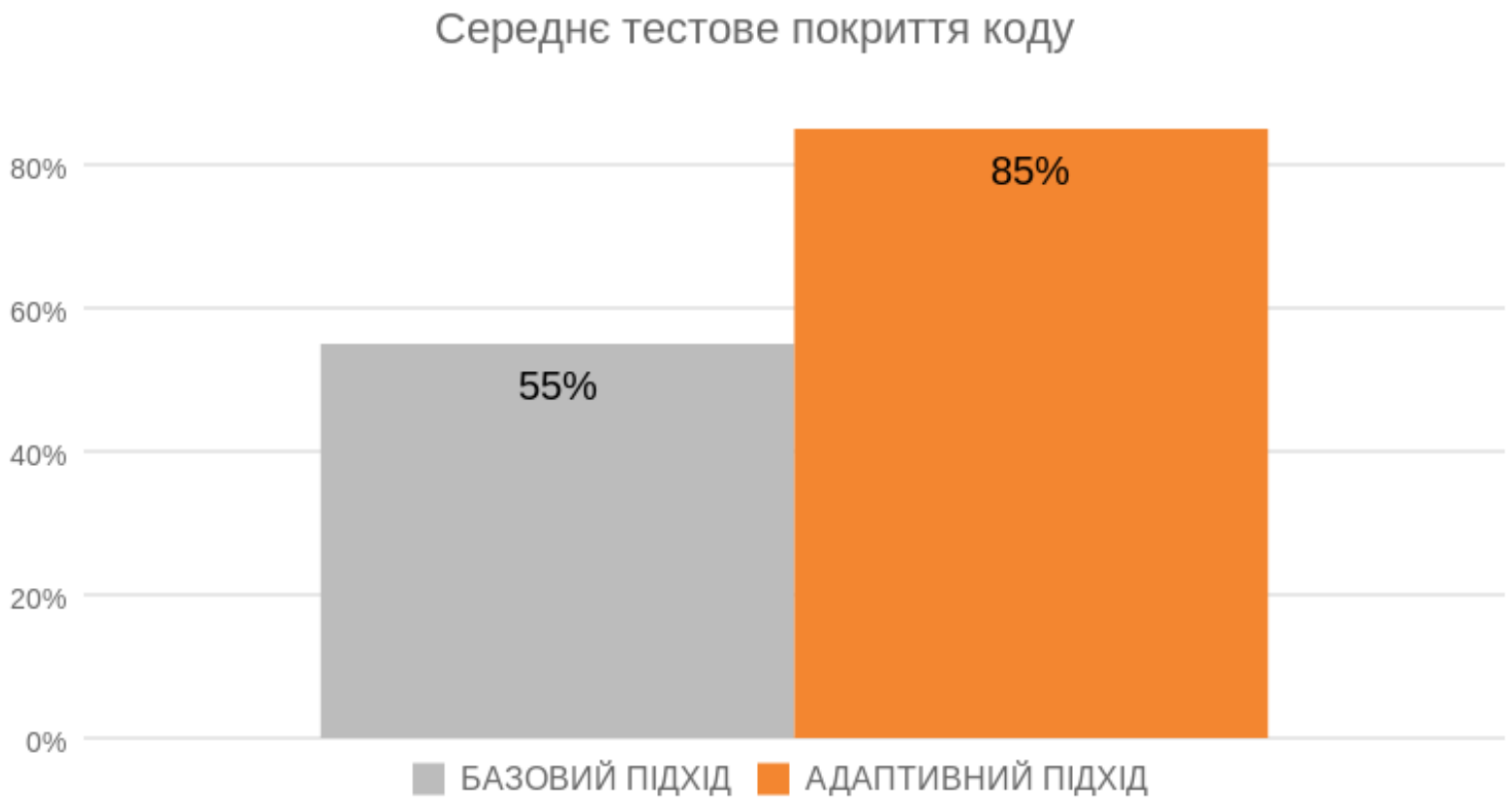
Копії графічних матеріалів



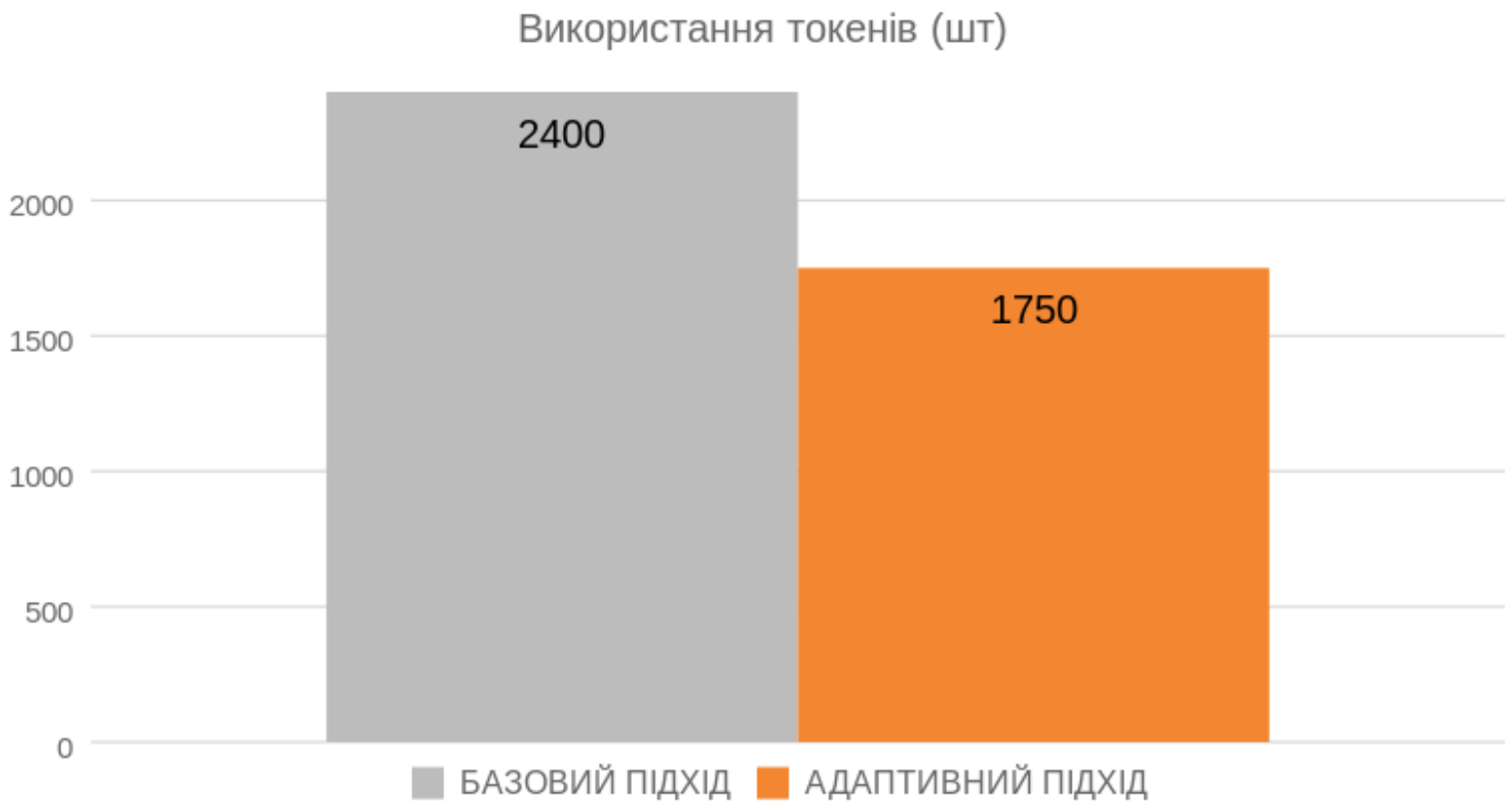
Блок-схема алгоритму автоматизації створення тестових сценаріїв
Грищенко О.В., група КП-31мп



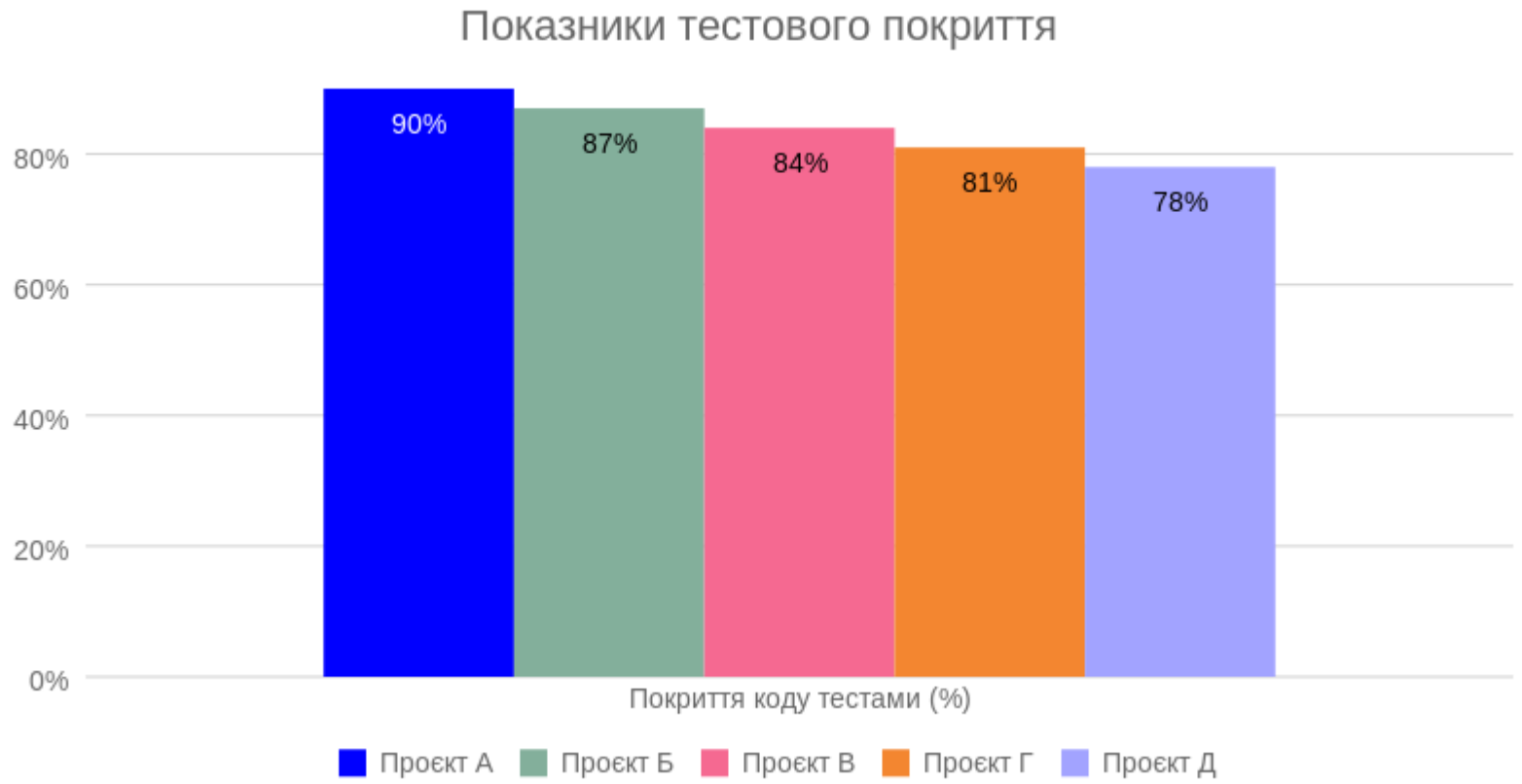
Блок-схема алгоритму формування адаптивного промпту
Грищенко О.В., група КП-31мп



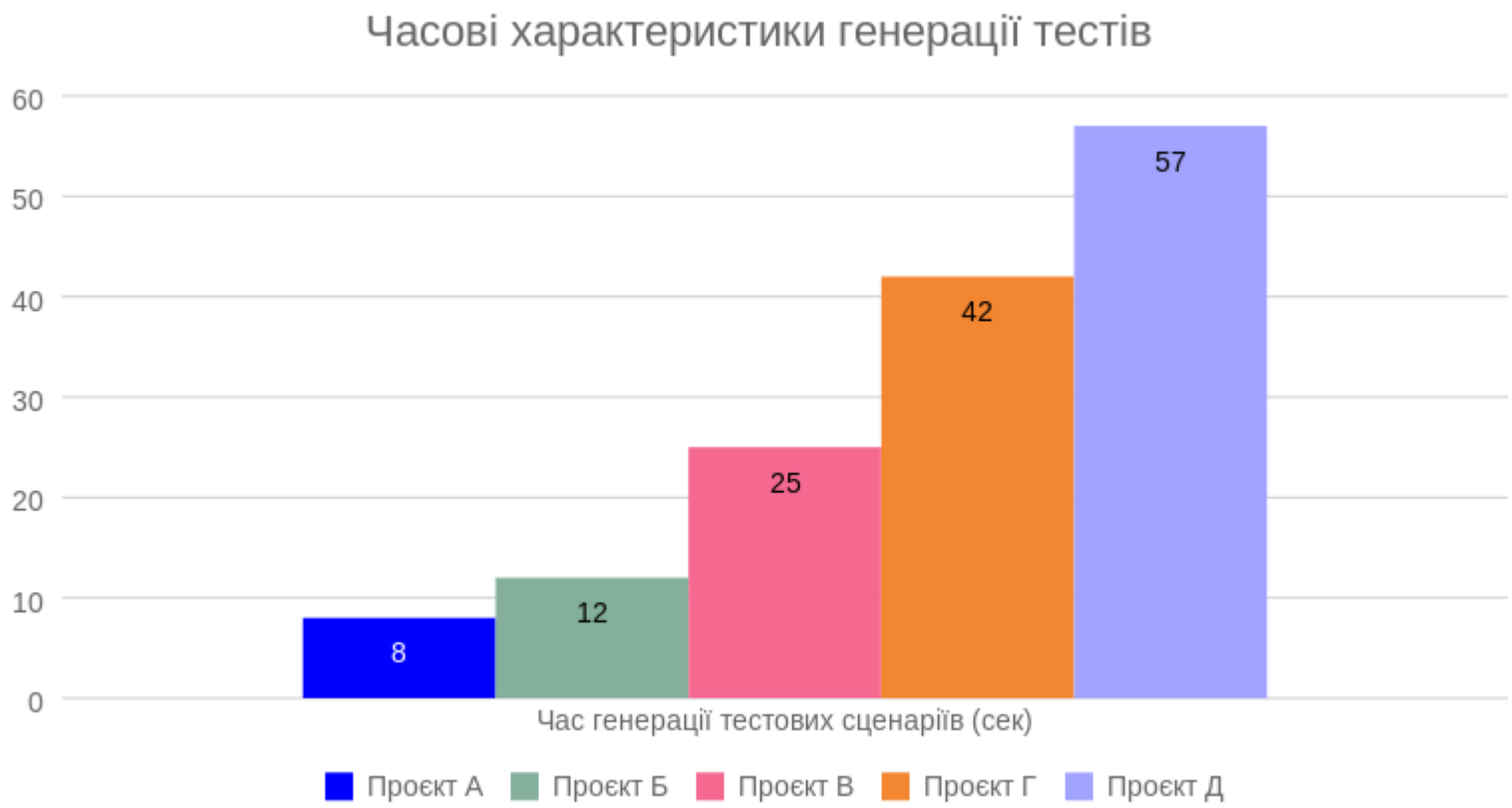
Гістограма порівняння значень середнього тестового покриття коду згенерованими тестовими сценаріями Грищенко О.В., група КП-31мп



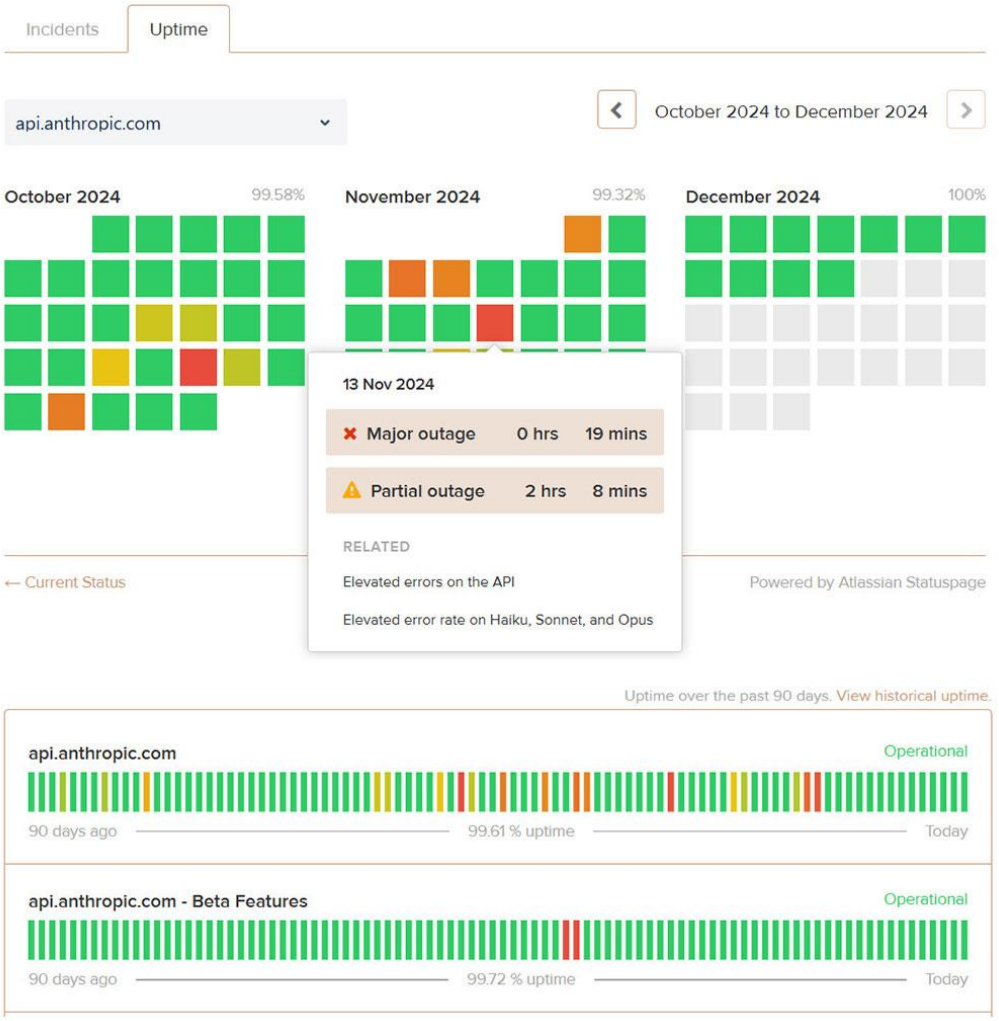
Гістограма порівняння значень середнього використання токенів для генерації одного тестового сценарію
Грищенко О.В., група КП-31мп



Порівняльна гістограма тестового покриття
Грищенко О.В., група КП-31мп



Порівняльна гістограма часу,
що знадобився для генерації тестових сценаріїв
Грищенко О.В., група КП-31мп

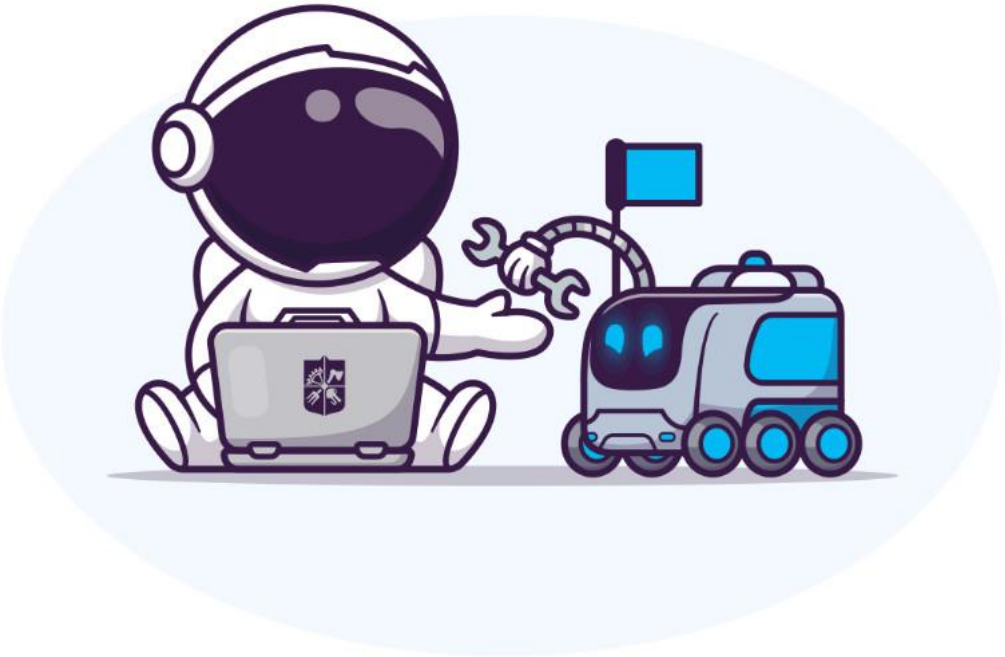


Статистика доступності сервісів Anthropic
Грищенко О.В., група КП-31мп

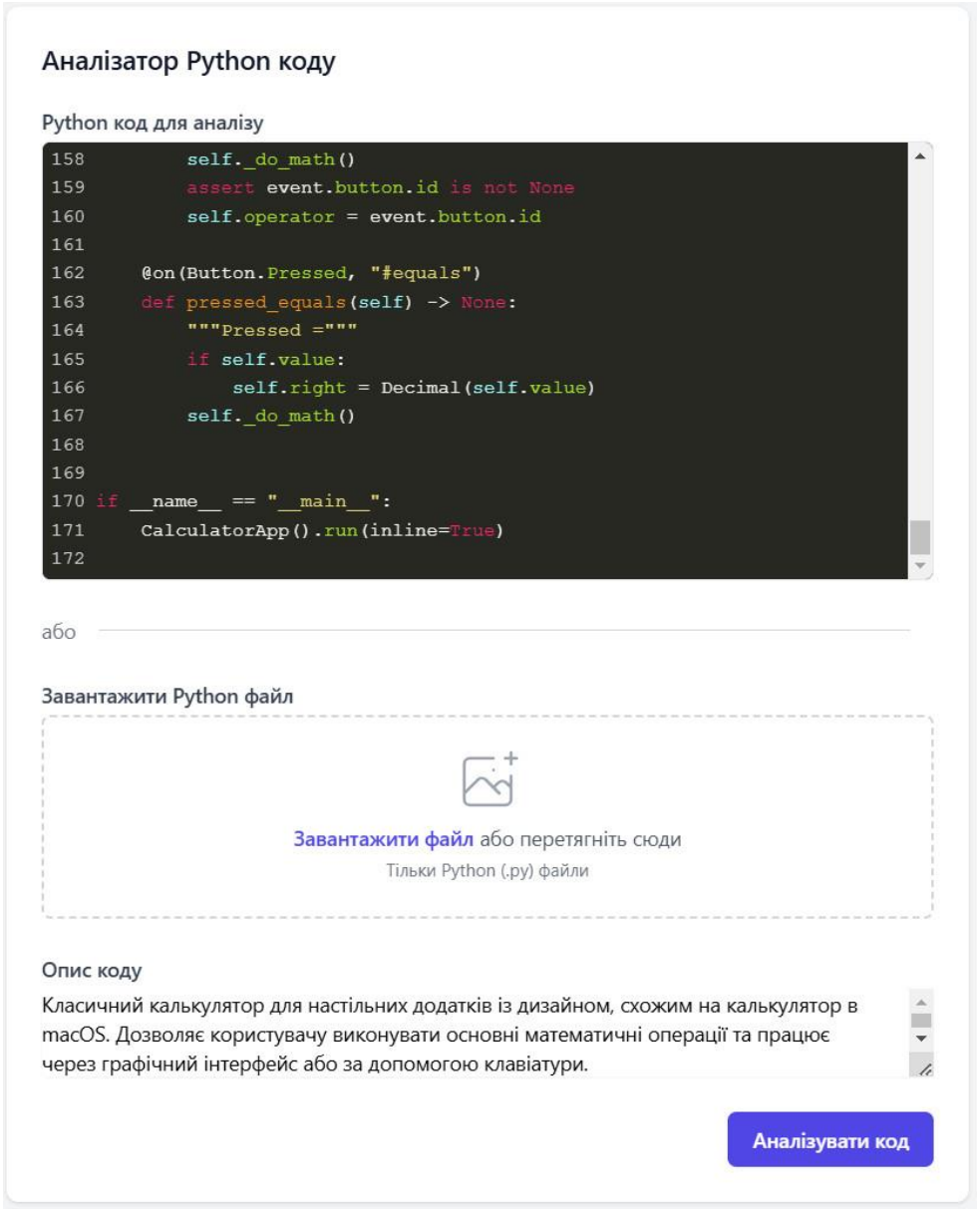
Marlet AI tests

Веб-застосунок для автоматизованого створення тестових сценаріїв при тестуванні програмного забезпечення

Згенерувати Тести



Головна сторінка веб-застосунку
Грищенко О.В., група КП-31мп



Приклад застосування форми
завантаження коду
та додавання опису програми
Грищенко О.В., група КП-31мп

plus_minus_pressed

basic

Test plus/minus sign conversion

Input

```
Current value is '100'
```

Expected Output

```
Value should change to '-100'
```

Python Test Code

```python
def test_plus_minus_conversion(self):
    self.app = CalculatorApp()
    self.app.value = '100'
    self.app.plus_minus_pressed()
    self.assertEqual(self.app.value, '-100')
```

percent_pressed

basic

Test percentage calculation

Input

```
Current value is '200'
```

Expected Output

```
Value should change to '2'
```

Python Test Code

```python
def test_percent_calculation(self):
    self.app = CalculatorApp()
    self.app.value = '200'
    self.app.percent_pressed()
    self.assertEqual(self.app.value, '2')
```

Приклад згенерованих тестових випадків
Грищенко О.В., група КП-31мп

Додаток 2

Лістинг програми

claude_connector.py

```
import os
import anthropic
from dotenv import load_dotenv
import time
import json

# Load environment variables
load_dotenv()

def generate_test_cases(code_content, functions_info):
    """Generate test cases using Claude API"""
    prompt = f"""You are a test case generator. Analyze the following code and generate comprehensive test cases regardless of code type.

Code:
```python
{code_content}
```

Function Information:
{json.dumps(functions_info, indent=2)}

Generate test cases in this exact JSON format:
[
  {{
    "function_name": "name_of_function_or_component",
    "description": "clear description of test purpose",
    "input": "specific test inputs including parameters, environment setup, or required state",
    "expected_output": "detailed expected behavior, return values, or side effects",
    "test_type": "basic|edge_case|error|integration|performance",
    "setup": "required setup code, dependencies, or preconditions",
    "teardown": "cleanup steps, resource disposal, or post-test actions",
    "python_code": "complete Python unittest code for this test case"
  }}
]

Key Testing Principles:
1. Python Language Testing:
  - Focus on Python language features
  - Ignore non-Python code
  - Consider language-specific edge cases
  - Test appropriate data types and structures
2. Comprehensive Coverage:
  - Core functionality
  - Input validation
  - Error handling
  - Edge cases
  - Performance aspects
3. Quality Criteria:
  - Clear test descriptions
```

```
    - Reproducible test steps
    - Python test code should be complete and runnable
    - Use Python's unittest framework
    - Include proper assertions and error handling
    - All asserting arguments should start with "self." like "self.app.value = '100'"
Return ONLY a valid JSON array. No additional text or explanation."""
    try:
        client = anthropic.Anthropic()
        response = client.messages.create(
            model="claude-3-5-haiku-20241022",
            max_tokens=4000,
            temperature=0.1,
            messages=[{
                "role": "user",
                "content": prompt
            }]
        )
       # Extract the response content
        content = response.content[0].text.strip()
        # Remove any markdown formatting
        content = content.replace('```json', '').replace('```', '').strip()
        try:
            # Try to parse the entire response as JSON
            test_cases = json.loads(content)
            if not isinstance(test_cases, list):
                raise ValueError("Response is not a JSON array")
            return test_cases
       except (json.JSONDecodeError, ValueError) as e:
            print(f"Error parsing response: {str(e)}")
            print(f"Raw content: {content}")
            return [{
                "function_name": "error",
                "description": "Error generating test cases",
                "input": f"Error: {str(e)}",
                "expected_output": "Failed to parse response",
                "test_type": "error",
                "setup": "",
                "teardown": "",
                "python_code": ""
            }]
        except Exception as e:
        print(f"Error generating test cases: {str(e)}")
        return [{
            "function_name": "error",
            "description": "Error generating test cases",
            "input": f"Error: {str(e)}",
            "expected_output": "Failed to generate test cases",
            "test_type": "error",
            "setup": "",
            "teardown": "",
            "python_code": ""
        }]
```

routes.py

```
from flask import Blueprint, jsonify, request, current_app, url_for
from werkzeug.utils import secure_filename
import os
import io
from app.analysis.code_analyzer import analyze_code
from app.api.claude_connector import generate_test_cases
from app.api.export_utils import generate_text_report, generate_pdf_report
import traceback

api_bp = Blueprint('api', __name__)

ALLOWED_EXTENSIONS = {'py'}

def allowed_file(filename):
    """Перевірка чи дозволене розширення файлу"""
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS

@api_bp.route('/health', methods=['GET'])
def health_check():
    """Перевірка стану API"""
    current_app.logger.debug('Викликано ендпоінт перевірки стану')
    return jsonify({'status': 'healthy'})

@api_bp.route('/analyze', methods=['POST'])
def analyze():
    current_app.logger.info('Analyze endpoint called')

    if 'file' not in request.files and 'code' not in request.form:
        current_app.logger.error('No file or code provided')
        return jsonify({'error': 'No file or code provided'}), 400

    code_content = None

    try:
        if 'file' in request.files:
            current_app.logger.debug('Processing uploaded file')
            file = request.files['file']

            if file.filename == '':
                current_app.logger.error('No selected file')
                return jsonify({'error': 'No selected file'}), 400

            if file and allowed_file(file.filename):
                # Read file content directly from the request stream
                try:
                    code_content = file.stream.read().decode('utf-8')
                    current_app.logger.debug('File content read successfully with utf-8')
                except UnicodeDecodeError:
                    current_app.logger.warning('UTF-8 decode failed, trying latin-1')
                    try:
```

```
                file.stream.seek(0)
                code_content = file.stream.read().decode('latin-1')
                current_app.logger.debug('File content read successfully with latin-1')
            except Exception as e:
                current_app.logger.error(f'Failed to read file content: {str(e)}')
                return jsonify({'error': f'Failed to read file: {str(e)}'}), 400
        else:
            current_app.logger.debug('Processing code from form data')
            code_content = request.form['code']

        if not code_content:
            current_app.logger.error('No code content found')
            return jsonify({'error': 'No code content found'}), 400

        description = request.form.get('description', '')

        current_app.logger.info('Starting code analysis')
        try:
            analysis_results = analyze_code(code_content)
            current_app.logger.debug('Code analysis completed')
        except Exception as e:
            current_app.logger.error(f"Analysis error: {str(e)}")
            current_app.logger.error(traceback.format_exc())
            analysis_results = {
                'error': f'Analysis failed: {str(e)}',
                'complexity': {'total_complexity': 0, 'average_complexity': 0, 'blocks': []},
                'metrics': {'maintainability_index': 0, 'loc': 0, 'lloc': 0, 'sloc': 0},
                'functions': []
            }

        current_app.logger.info('Generating test cases')
        try:
            test_cases = generate_test_cases(code_content, description)
            current_app.logger.debug('Test cases generated')
        except Exception as e:
            current_app.logger.error(f"Test case generation failed: {str(e)}")
            test_cases = []

        return jsonify({
            'analysis': analysis_results,
            'test_cases': test_cases
        })

    except Exception as e:
        current_app.logger.error(f'Error in analyze endpoint: {str(e)}')
        current_app.logger.error(traceback.format_exc())
        return jsonify({'error': str(e)}), 500
```

code_analyzer.py

```
from typing import Dict, List, Union, Tuple, Optional
import radon.complexity as cc
from radon.raw import analyze
from radon.metrics import mi_visit
import ast
import json
import networkx as nx
import matplotlib.pyplot as plt
import io
import base64
from collections import defaultdict

def analyze_code(code_content: str) -> Dict[str, Union[Dict, List]]:
    """
    Аналізує Python код та повертає метрики з тестовими сценаріями
    """
    try:
        # Базовий аналіз коду
        analysis_results = _perform_basic_analysis(code_content)

        # Генерація тестових сценаріїв на основі AST аналізу
        test_scenarios = _generate_test_scenarios(code_content)

        # Додатковий аналіз складності
        complexity_details = _analyze_complexity_details(code_content)
        analysis_results['complexity_details'] = complexity_details

        return {
            'analysis': analysis_results,
            'test_cases': test_scenarios
        }
    except SyntaxError as e:
        return {
            'error': f'Синтаксична помилка в коді: {str(e)}'
        }
    except Exception as e:
        return {
            'error': f'Помилка аналізу: {str(e)}'
        }

def _generate_recommendations_for_complexity(complexity: int) -> List[str]:
    """Генерує детальні рекомендації на основі циклометричної складності"""
    recommendations = []

    if complexity <= 5:
        recommendations.append("Код має хорошу складність")
    elif complexity <= 10:
        recommendations.append("Код має прийнятну складність")
        recommendations.append("Можливість документування складних умов")
    elif complexity <= 20:
        recommendations.append("Код має високу складність")
```

```
            recommendations.append("Розділіть функцію на менші частини")
            recommendations.append("Спростіть умовні конструкції")
            recommendations.append("Додайте детальні коментарі до складних частин")
        else:
            recommendations.append("Код має критично високу складність!")
            recommendations.append("Терміново потрібен рефакторинг")
            recommendations.append("Розбийте код на окремі функції")
            recommendations.append("Перегляньте логіку та спростіть умови")
            recommendations.append("Додайте модульні тести")

        return recommendations

    def _get_risk_level(complexity: float) -> str:
        """Визначає рівень ризику на основі циклометричної складності"""
        if complexity <= 10:
            return 'Низький'
        elif complexity <= 20:
            return 'Середній'
        else:
            return 'Високий'

    def _get_maintainability_rating(mi_score: float) -> str:
        """Визначає оцінку підтримуваності на основі індексу підтримуваності"""
        if mi_score >= 80:
            return 'Добрий'
        elif mi_score >= 60:
            return 'Середній'
        else:
            return 'Поганий'

    def _generate_test_scenarios(code_content: str) -> Dict:
        """Генерує тестові сценарії для початківців"""
        try:
            tree = ast.parse(code_content)
            scenarios = []

            # Аналіз AST для пошуку класів та функцій
            for node in ast.walk(tree):
                if isinstance(node, (ast.ClassDef, ast.FunctionDef)):
                    try:
                        new_scenarios = _create_scenarios_for_node(node)
                        if new_scenarios:
                            scenarios.extend(new_scenarios)
                    except Exception as e:
                        scenarios.append(_create_error_scenario(node, str(e)))

            if not scenarios:
                scenarios.append(_create_basic_test_scenario())

            return json.dumps({
                'test_cases': scenarios,
                'metadata': {
```

```
                'total_tests': len(scenarios),
                'framework': 'unittest',
                'language': 'python'
            }
        })
    except Exception as e:
        return json.dumps({
            'test_cases': [_create_error_scenario(None, str(e))],
            'metadata': {
                'total_tests': 1,
                'framework': 'unittest',
                'language': 'python'
            }
        })

def _create_scenarios_for_node(node: Union[ast.ClassDef, ast.FunctionDef]) -> List[Dict]:
    """Створює детальні тестові сценарії для заданого вузла"""
    scenarios = []

    # Generate unique test ID
    test_id = f"test_{node.name.lower()}"

    # Get function arguments and return type hints if available
    args = []
    returns = None
    if isinstance(node, ast.FunctionDef):
        args = [arg.arg for arg in node.args.args]
        if node.returns:
            returns = ast.unparse(node.returns)

    # Create test scenario
    scenario = {
        'id': test_id,
        'name': f"Test {node.name}",
        'description': f"Test functionality of {node.name}",
        'type': 'unit_test',
        'preconditions': [
            f"Function {node.name} is defined and accessible",
            f"Required arguments: {', '.join(args) if args else 'None'}"
        ],
        'steps': _generate_function_test_steps(node)
    }

    scenarios.append(scenario)
    return scenarios

def _generate_function_test_steps(node: ast.FunctionDef) -> List[Dict]:
    """Генерує фактичні кроки тестування для функції"""
    steps = []
```

```
            # Get function arguments
            args = [arg.arg for arg in node.args.args]

            # Basic positive test
            steps.append({
                'step_number': 1,
                'description': f"Test {node.name} with valid input",
                'code': f"""
        # Test with valid input
        result = {node.name}({', '.join('valid_' + arg for arg in args)})
        self.assertIsNotNone(result)
        """,
                'expected_result': "Function executes successfully and returns expected result",
                'notes_for_junior': "Ensure all required arguments are provided with valid values"
            })

            # Edge case test
            steps.append({
                'step_number': 2,
                'description': f"Test {node.name} with edge cases",
                'code': f"""
        # Test with edge cases
        try:
            result = {node.name}({', '.join('edge_case_' + arg for arg in args)})
            self.assertIsNotNone(result)
        except Exception as e:
            self.fail(f"Failed with edge case: {{str(e)}}")
        """,
                'expected_result': "Function handles edge cases gracefully",
                'notes_for_junior': "Consider testing with empty values, None, or boundary conditions"
            })

            # Error handling test
            steps.append({
                'step_number': 3,
                'description': f"Test {node.name} error handling",
                'code': f"""
        # Test error handling
        with self.assertRaises(Exception):
            {node.name}({', '.join('invalid_' + arg for arg in args)})
        """,
                'expected_result': "Function raises appropriate exception for invalid input",
                'notes_for_junior': "Verify that the function properly validates input and raises appropriate exceptions"
            })

            return steps
        def _create_error_scenario(node: Optional[Union[ast.ClassDef, ast.FunctionDef]], error: str) -> Dict:
            """Створює сценарій помилки з корисною інформацією для налагодження"""
            return {
```

```
            'id': f"error_{node.name if node and hasattr(node, 'name') else 'unknown'}",
            'name': f"Error in {node.name if node and hasattr(node, 'name') else 'code analysis'}",
            'description': f"Error occurred during test generation: {error}",
            'type': 'error',
            'preconditions': ["Code must be valid Python"],
            'steps': [{
                'step_number': 1,
                'description': "Fix the code error",
                'code': "# Debug information:\n# " + error.replace('\n', '\n# '),
                'expected_result': "Error should be resolved",
                'notes_for_junior': "Check syntax and ensure all dependencies are imported"
            }]
        }
    def _create_basic_test_scenario() -> Dict:
        """Створює базовий тестовий сценарій, коли аналіз коду не вдається"""
        return {
            'id': 'TS1',
            'name': 'Basic Functionality Test',
            'description': 'Test the basic functionality of the code',
            'type': 'basic_test',
            'steps': [
                {
                    'step_number': 1,
                    'description': 'Import the module',
                    'expected_result': 'Module should be imported without errors',
                    'notes_for_junior': 'Make sure you are in the correct directory'
                },
                {
                    'step_number': 2,
                    'description': 'Test main functionality',
                    'expected_result': 'Code should execute without errors',
                    'notes_for_junior': 'Start with simple test cases'
                }
            ],
            'preconditions': [
                'Python environment is set up',
                'Source code file is accessible'
            ]
        }
```

Додаток 3

Копія презентації

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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

Метод та програмне забезпечення автоматизації створення тестових сценаріїв з використанням штучного інтелекту

Доповідач: Грищенко Олександр Володимирович

Науковий керівник: к.т.н, старший викладач кафедри ПЗКС, Хіцко Яна Володимирівна

Київ – 2024

ЗАГАЛЬНА ІНФОРМАЦІЯ ПРО ДОСЛІДЖЕННЯ

Об’єкт дослідження: процес автоматизації тестування програмного забезпечення з використанням штучного інтелекту.

Предмет дослідження: методи та алгоритми застосування ШІ в автоматизації тестування програмного забезпечення.

Мета дослідження: підвищення тестового покриття коду та зменшення ручної обробки в тестуванні ПЗ за рахунок використання ШІ для автоматизації створення тестових сценаріїв.

АКТУАЛЬНІСТЬ ТЕМИ

Статистика та сучасний стан:

- За даними дослідження Stack Overflow Developer Survey 2023, 48% розробників-початківців вважають тестування однією з найскладніших задач у процесі розробки ПЗ.
- Зростаюча складність сучасного ПЗ потребує все більш комплексного підходу до тестування.

АКТУАЛЬНІСТЬ ТЕМИ

Основні проблеми в процесі тестування:

- Недостатність знань та досвіду для створення ефективних тестів
- Складність визначення граничних випадків та можливих сценаріїв використання
- Значні затрати часу на створення та підтримку тестових сценаріїв

АКТУАЛЬНІСТЬ ТЕМИ

Наслідки неефективного тестування:

- Пропущені помилки в програмному забезпеченні
- Збільшення часу на розробку та підтримку проєктів
- Зниження якості кінцевого продукту
- Підвищення витрат на виправлення помилок на пізніх етапах розробки

ІСНУЮЧІ РІШЕННЯ

SMARTBEAR TestComplete

Selenium IDE Browser Automation

Katalon

Недоліки		
Використання лише статичного аналізу замість можливостей ШІ	Високий поріг входу для розробників-початківців	Потреба в значній ручній доробці згенерованих тестів

ОСОБЛИВОСТІ РОЗРОБЛЕНОГО МЕТОДУ

Розроблений метод спрямований на автоматизацію процесу створення тестових сценаріїв з використанням штучного інтелекту за допомогою API Claude.

При цьому:

1. Користувач надає дані та перевіряє результати.
2. Веб-застосунок аналізує наданий код, визначає ключові метрики та генерує адаптивний промпт.
3. ШІ аналізує передані дані, генерує тестовий сценарій та випадки.

ПЕРЕВАГИ ЗАСТОСУВАННЯ ШІ ДЛЯ ТЕСТУВАННЯ ПЗ

1. **Зменшення ручної роботи** та можливості людських помилок
2. **Виявлення неочевидних** граничних випадків
3. **Зниження витрат** на процес тестування
4. **Виявлення** складних та **прихованих дефектів**

АЛГОРИТМ РОБОТИ МЕТОДУ

1. Користувач надає код та опис програми, вказує вимоги та обмеження тестування.
2. Застосунок виконує аналіз коду.
3. Генерація промпту з контекстом розробки, специфікаціями, обмеженнями та вимогами до тестів.
4. Передача промпту до Claude API для генерації тестових сценаріїв.
5. Автоматична перевірка тестів на повноту покриття та відповідність стандартам.
6. Користувач отримує готові тести для використання.

АЛГОРИТМ РОБОТИ МЕТОДУ

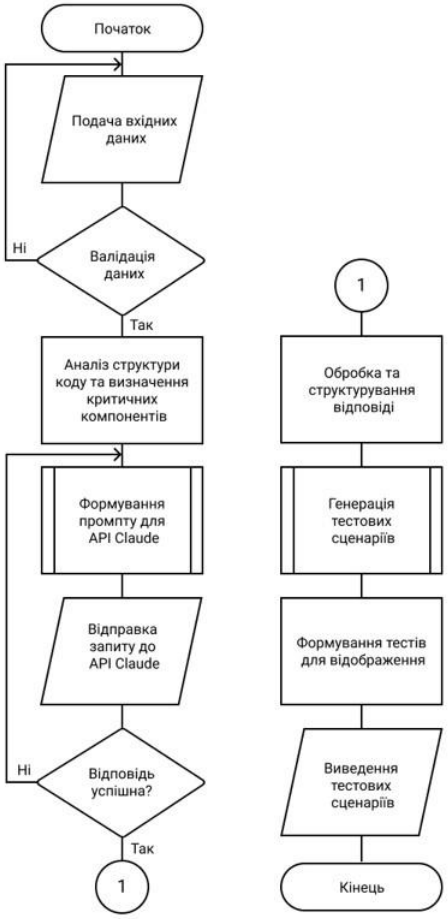


Рисунок 1 – Блок-схема алгоритму автоматизації створення тестових сценаріїв

ЕФЕКТИВНІСТЬ РІЗНИХ ПІДХОДІВ ДО ФОРМУВАННЯ ПРОМПТІВ

№	Підхід	Покриття коду	Час обрабки (с)	Використання токенів
1	Базовий	55%	4.5	2400
2	Оптимізований	72%	3.3	1750
3	Адаптивний	85%	2.8	1280

Базовий підхід – використання шаблонних промптів із загальними інструкціями, без додаткового контексту та специфікацій.
Оптимізований підхід – динамічна генерація промптів на основі:

- аналізу попередніх результатів генерації;
- статистики успішності виявлення помилок.

Адаптивний підхід – формування промптів з урахуванням специфіки проекту та додаванням метаінформації. Включає:

- аналіз структури коду;
- додавання опису коду та прикладів очікуваних результатів.

АЛГОРИТМ ФОРМУВАННЯ АДАПТИВНОГО ПРОМПТУ

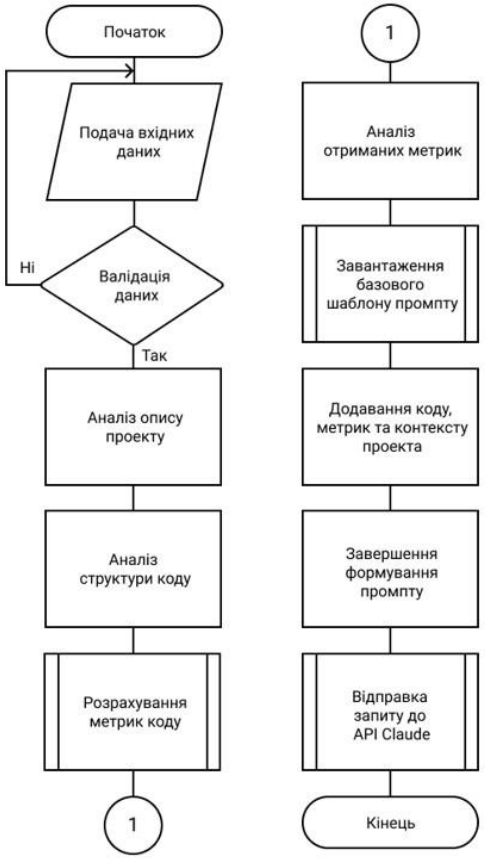


Рисунок 2 – Блок-схема алгоритму формування адаптивного промпту

ЕФЕКТИВНІСТЬ РІЗНИХ ПІДХОДІВ ДО ФОРМУВАННЯ ПРОМПТІВ

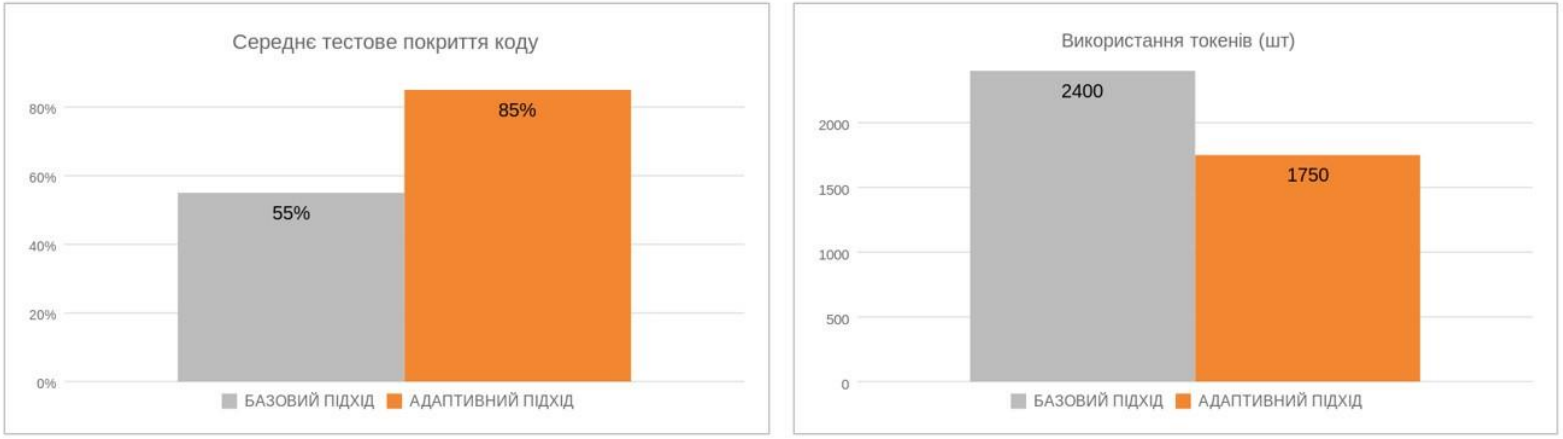
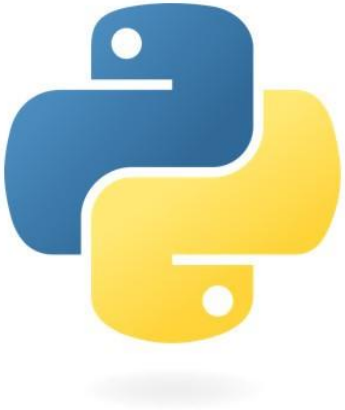


Рисунок 3 – Порівняльні гістограми ефективності
при застосуванні адаптивного підходу до формування промптів

ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

Claude
3.5 Haiku



Умови експерименту		
5 проєктів	Модель Claude 3.5 Haiku	Мова програмування Python

ПРИКЛАД ЗАСТОСУВАННЯ

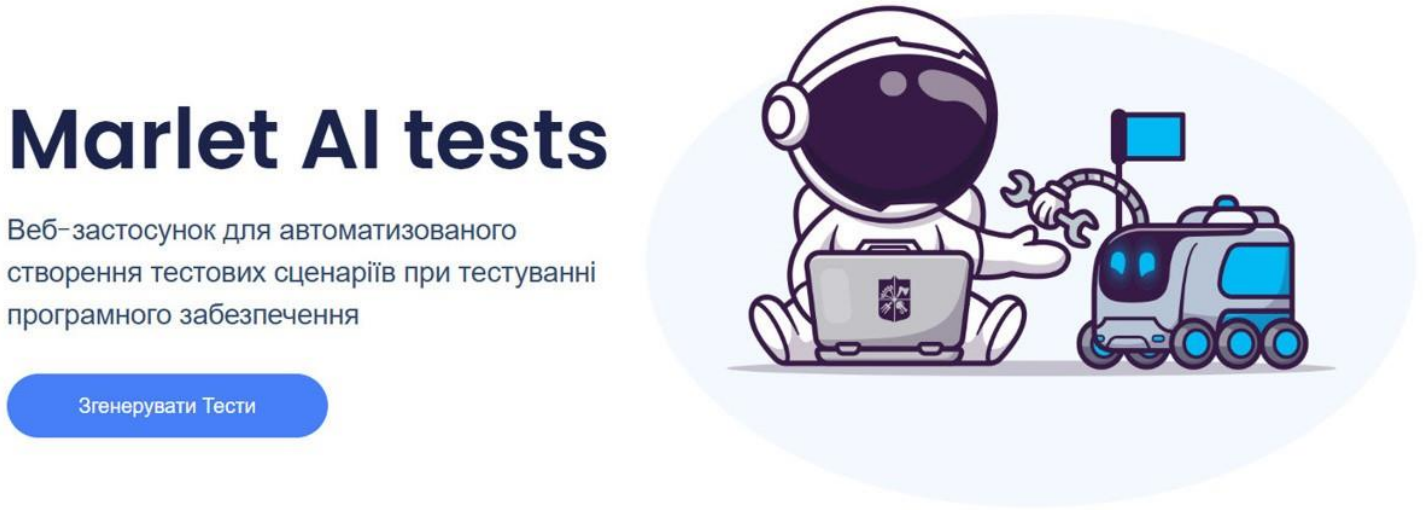


Рисунок 4 – Зображення головної сторінки веб-застосунку

ПРИКЛАД ЗАСТОСУВАННЯ

Аналіз коду

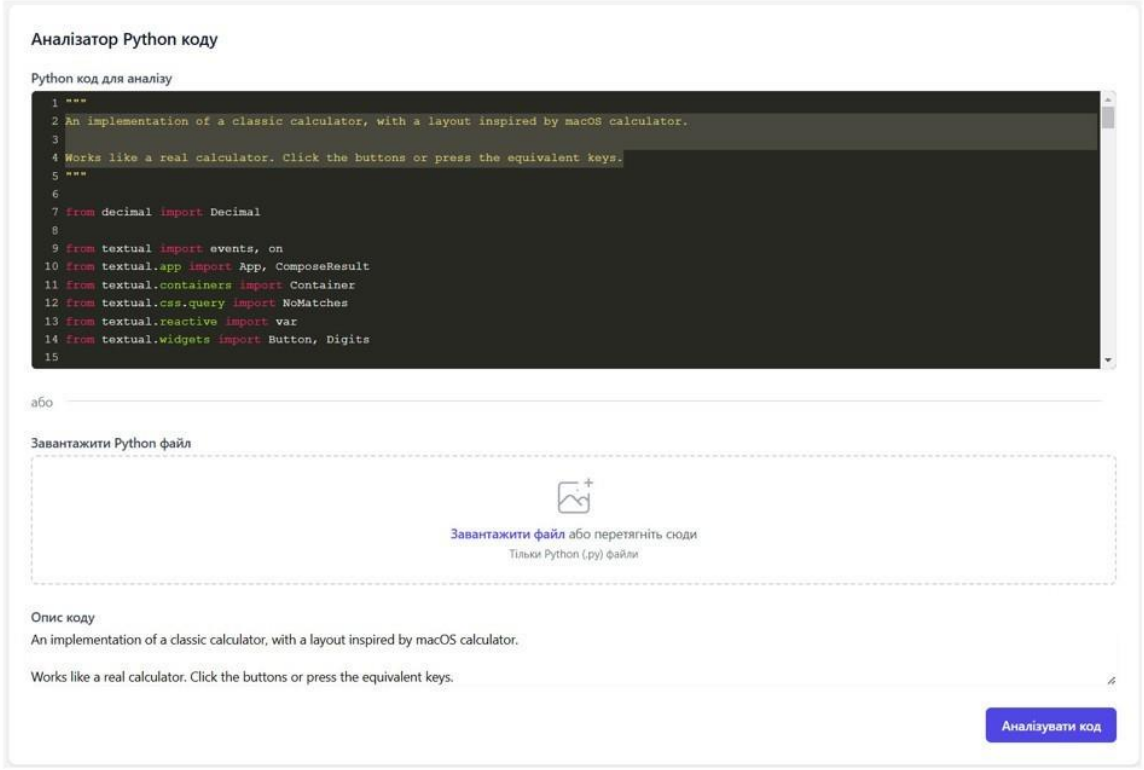
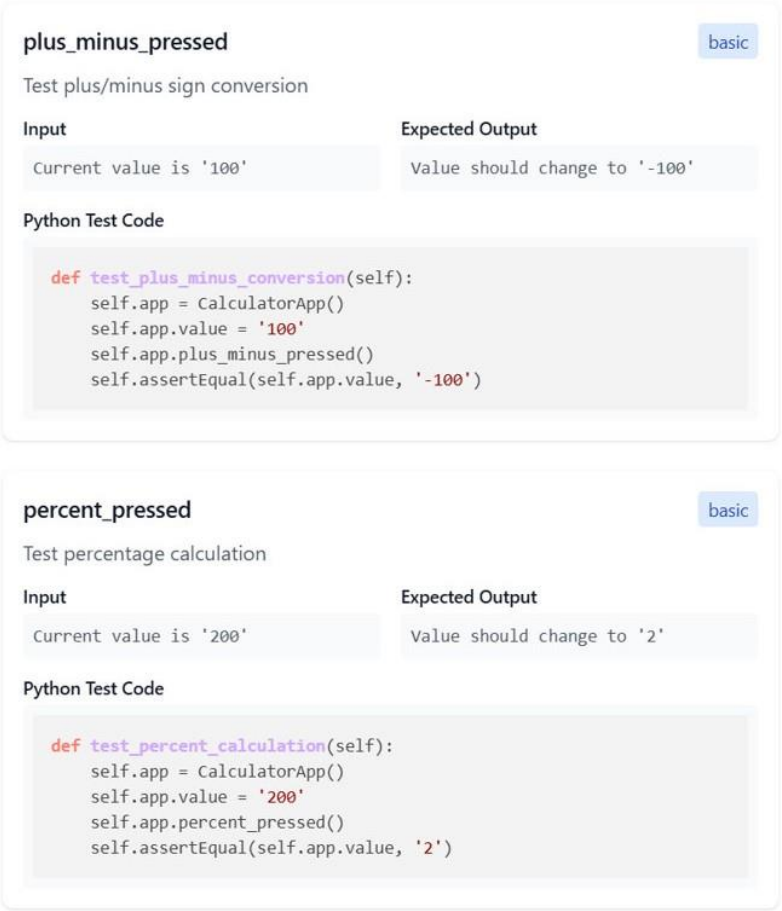


Рисунок 5 – Модуль завантаження та аналізу коду

ПРИКЛАД ЗАСТОСУВАННЯ

Рисунок 6 – Приклад згенерованого тестового випадку

ПРИКЛАД ЗАСТОСУВАННЯ



plus_minus_pressed basic

Test plus/minus sign conversion

Input

```
Current value is '100'
```

Expected Output

```
Value should change to '-100'
```

Python Test Code

```
def test_plus_minus_conversion(self):
    self.app = CalculatorApp()
    self.app.value = '100'
    self.app.plus_minus_pressed()
    self.assertEqual(self.app.value, '-100')
```

percent_pressed basic

Test percentage calculation

Input

```
Current value is '200'
```

Expected Output

```
Value should change to '2'
```

Python Test Code

```
def test_percent_calculation(self):
    self.app = CalculatorApp()
    self.app.value = '200'
    self.app.percent_pressed()
    self.assertEqual(self.app.value, '2')
```

Рисунок 7 – Приклад згенерованих тестових випадків

ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

Обрахунок покриття коду тестами

TIDELIFT

Coverage.py

```
Stderr                                                    Copy  Insert

..........
----------------------------------------------------------------------
Ran 11 tests in 0.092s

OK
```

```
Ran terminal command
c:/marlet_test
> python -m coverage report

Command output

Stdout                                                    Copy  Insert

Name                          Stmts   Miss  Cover
-------------------------------------------------
uploads\calculator.py           113     25    78%
uploads\test_calculator.py      130      1    99%
-------------------------------------------------
TOTAL                           243     26    89%
```

Рисунок 8 та рисунок 9 – Приклад проведення замірів покриття коду за допомогою coverage.py

ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

Середнє покриття коду	Значення (%)
Малі файли (до 100 рядків)	**92%**
Середні файли (100-500 рядків)	**89%**
Великі файли (від 500 рядків)	**80%**

Таблиця 3 – Середні показники тестового покриття коду

Швидкість генерації тестових випадків	Час (сек)
Малі файли (до 100 рядків)	**8 сек**
Середні файли (100-500 рядків)	**12 сек**
Великі файли (від 500 рядків)	**21 сек**

Таблиця 4 – Середні показники швидкості генерації тест-кейсів

РОЗРОБКА ВЕБ-ЗАСТОСУНКУ

Використані технології

НАУКОВА НОВИЗНА

Створено метод автоматизації процесу тестування ПЗ з використанням ШІ для генерації тестових сценаріїв, який поєднує статичний аналіз коду з адаптивною системою формування промптів для взаємодії з API Claude, що зменшує використання токенів API на 25% під час генерації тестових сценаріїв у порівнянні з базовим підходом до формування промптів та існуючими аналогами у сфері тестування ПЗ. Запропонований метод забезпечує високий рівень покриття коду (до 90%) та швидкість аналізу до 10 секунд на кожні 100 рядків коду.

ПРАКТИЧНА ЦІННІСТЬ

1. Досягнення середнього тестового покриття коду 85%, часу аналізу поданого коду до 10 секунд за кожні 100 рядків коду та зменшення ручної обробки в процесі тестування ПЗ за рахунок автоматизації створення тестових сценаріїв з використанням ШІ.
2. Зменшення ризиків та витрат, пов'язаних з помилками та дефектами у ПЗ
3. Також, застосування адаптивного підходу до створення промптів для запитів до API Claude, дозволяє економити до 25% токенів порівняно з базовим підходом, що не враховує особливостей поданого коду, опису та специфіки проєкту користувача в процесі аналізу коду та генерації тестових сценаріїв.

АПРОБАЦІЯ

Результати роботи були представлені та обговорювались на XVII науково-практичній конференції магістрантів та аспірантів ПМК-2024 факультету прикладної математики – "Прикладна математика та комп'ютинг" (м. Київ, 20-22 листопада 2024 р.)

ВИСНОВКИ ТА ПОДАЛЬША РОБОТА

1. Проведено аналіз існуючих методів тестування ПЗ.
2. Розроблено метод автоматизації створення тестових сценаріїв з використанням ШІ.
3. Розроблено алгоритм автоматичного генерування тестових сценаріїв з ШІ.
4. Проведено дослідження ефективності методу: тестове покриття коду та швидкість генерації тестів.
5. Розроблено веб-застосунок

Напрямки подальшої роботи

- Покращення та додавання можливостей веб-застосунку
- Розширення підримки мов програмування
- Вдосконалення алгоритму генерації тестових сценаціїв

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Johnsson N. AI-driven Test Case Generation: Utilizing Large Language Models for Automated Testing // Degree Project in Interaction Technology and Design, Umea University. – 2023. – Vol. 30. – P. 1-42. DOI: 10.13140/RG.2.2.15287.24480
2. Nama P., Bhoyar M., Chinta S. Autonomous Test Oracles: Integrating AI for Intelligent Decision-Making in Automated Software Testing // Well Testing Journal. – 2024. – Vol. 33, № S2. – P. 326-353.
3. Lops A., Narducci F., Ragone A., Trizio M., Bartolini C. A System for Automated Unit Test Generation Using Large Language Models and Assessment of Generated Test Suites // arXiv preprint arXiv:2408.xxxx. – 2024. [Cornell University]
4. Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Computing Surveys, 55(9):1–35, 2023.

Дякую за увагу!