Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження застосування методів штучного інтелекту в автоматизації тестування програмного забезпечення
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗздм-22-1

Ларченко С. О.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц.кафедри ПІ Лановий О.Ф.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Ларченку Сергію Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження застосування методів штучного інтелекту в автоматизації тестування програмного забезпечення»
Затверджена наказом по університету від 22.04.2024 №60Стз
2. Термін подання студентом роботи до екзаменаційної комісії 15.06.2024р
3. Вихідні дані до роботи дослідити методи штучного інтелекту, які базуються на технології трансформерів та використовуються для обробки природної мови та генерації тексту для використання в автоматизованому тестуванні програмного забезпечення.
4. Перелік питань, що потрібно опрацювати в роботі
аналіз предметної галузі та постановка задачі, вибір методу штучного інтелекту для автоматизації програмного забезпеченняогляд та аналіз існуючих методів штучного інтелекту, дослідження теоретичне, дослідження практичне.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів	Примітка
1	Аналіз предметної галузі та постановка задачі	14.02.2024 - 24.02.2024	Виконано
2	Дослідження та аналіз існуючих методів ШІ та їх використання в автоматизації ПЗ	25.02.2024 - 05.03.2024	Виконано
3	Підготовка до експерименту та збір даних	06.03.2024 - 20.03.2024	Виконано
4	Написання та оформлення статті та тез доповіді	15.03.2024 - 04.04.2024	Виконано
5	Нормалізація даних та проведення експериментів	21.03.2024 - 11.04.2024	Виконано
6	Аналіз та оформлення результатів	12.04.2024 - 25.04.2024	Виконано
7	Підготовка пояснювальної записки	26.04.2024 - 18.05.2024	Виконано
8	Підготовка презентації та доповіді	18.05.2024 - 26.05.2024	Виконано
9	Перевірка на плагіат	15.06.2024	Виконано
10	Нормоконтроль	15.06.2024	Виконано
11	Оцінка роботи рецензентом	20.06.2024	Виконано
12	Занесення роботи в електронний архів	20.06.224	Виконано
13	Попередній захист роботи	20.06.2024	Виконано
14	Допуск до захисту завідуючим кафедри	21.06.2024	Виконано

Дата видачі завдання «01» лютого 2024 р.

Студент _____ Ларченко С.О.

Керівник роботи _____ доц. каф. ПІ Лановий О.Ф.

РЕФЕРАТ/ABSTRACT

Кваліфікаційна робота магістра містить: 71 с., 34 рис., 4 табл., 24 джерела.

АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, ВЕЛИКІ МОВНІ МОДЕЛІ, МАШИННЕ НАВЧАННЯ, МЕТОДИ ШТУЧНОГО ІНТЕЛЕКТУ, ПРОМПТ ІНЖЕНІРІНГ.

Об’єктом дослідження є методи штучного інтелекту, які базуються на технології трансформерів та використовуються для обробки природної мови та генерації тексту.

Метою роботи є проведення дослідження ефективності використання методів штучного інтелекту для генерації автоматизованих тестів для тестування програмного забезпечення.

Методи дослідження. В роботі використовуються інструменти генерації природньої мови, для проведення дослідження шляхом порівняння ефективності використання їх у автоматизації програмного забезпечення.

У результаті кваліфікаційної роботи було згенеровано три тестових набори, по одному для кожної з вибраної моделі: GPT4, GPT-3.5 та Gemini(Bard).

AUTOMATED TESTING, LARGE LANGUAGE MODELS, MACHINE LEARNING, ARTIFICIAL INTELLIGENCE METHODS, PROMPT ENGINEERING.

The object of research are artificial intelligence methods based on transformers technology and used for natural language processing and text generation.

The purpose of the work is to conduct a study of the effectiveness of using artificial intelligence methods to generate automated tests for testing software.

Methods of research. The work uses natural language generation tools to conduct research by comparing the effectiveness of their use in software automation.

As a result of the qualification work, three test sets were generated, one for each of the selected models: GPT4, GPT-3.5, and Gemini (Bard).

Я, Ларченко Сергій Олексійович студент групи ІПЗздм-22-1 здобувач вищої освіти на другому (магістерському) рівні кафедра програмної інженерії, заявляю: моя кваліфікаційна робота на тему «Дослідження застосування методів штучного інтелекту в автоматизації тестування програмного забезпечення», що буде представлена до ЕК для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – Програмне забезпечення

ПІ – програмна інженерія

МН - машинне навчання

ШІ – Штучний інтелект

CI/CD - безперервна інтеграція та безперервна доставка

ISTQB - Міжнародна Асоціація Тестування Програмного Забезпечення

LLM – велика мовна модель

ВСТУП

Швидкість та якість розробки програмного забезпечення (ПЗ) стають ключовими факторами успіху для будь-якої ІТ-компанії. Для досягнення цих цілей автоматизоване тестування стає надзвичайно важливим етапом розробки програмного продукту. Щоб забезпечити високу якість та ефективність тестування, використання методів штучного інтелекту (ШІ) набуває все більшого значення в індустрії розробки ПЗ. Актуальність теми обумовлена загальним зростанням складності програмного забезпечення, що вимагає все більш продуктивних методів тестування. Традиційні методи тестування вимагають значних зусиль і часу, а також суттєво залежать від кваліфікації тестувальників. Втім, застосування штучного інтелекту може сприяти автоматизації процесів створення автоматизованих тестів, забезпечивши більш точне та ефективне тестування програмного забезпечення.

Одним із основних завдань дослідження є аналіз потреби в автоматизації тестування програмного забезпечення та вибір методів штучного інтелекту, які можуть бути застосовані використанні в автоматизації тестування ПЗ. Додатковою метою є побудова математичної моделі, яка дозволить визначити найбільш придатний метод ШІ для використанні в автоматизації тестування ПЗ.

У процесі наукового дослідження зосереджено увагу на сучасних тенденціях та інноваціях в сфері штучного інтелекту, що застосовуються для тестування програмного забезпечення. Особлива увага приділена такому методу ШІ, як машинне навчання (МН) та його здатності адаптуватися до різноманітних тестових сценаріїв. Розглянуті можливості використання ШІ для підвищення ефективності тестування, потенційні виклики та обмеження, пов'язані з інтеграцією штучного інтелекту у процеси автоматизації тестування програмного забезпечення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення та ключові характеристики автоматизації тестування

Автоматизація тестування ПЗ – це ключ до ефективного та економного підходу розробки програмного забезпечення. Використання автоматизації тестування дозволяє значно скоротити час на тестування, підвищити його точність та повторюваність, а також інтегрувати його з іншими процесами розробки. Завдяки автоматизації тестування можна виявити більше помилок, покращити загальну якість програмного продукту та заощадити кошти. В наш час програмні продукти нерідко розробляються малими ітераціями в стислі терміни і при обмежених бюджетах проектів[1], тому необхідність впровадження автоматизації тестування ПЗ зростає, особливо при постійному розширенню функціоналу застосунка, а також для впевненості у якості готового продукту при подальшому супроводі.

Автоматизація процесу тестування стала невід'ємною частиною сучасних практик в індустрії розробки програмного забезпечення. Існує значна кількість методів та технологій контролю за якістю та надійністю ПЗ [2]. Автоматизовані тести можуть бути виконані швидко та ефективно та здатні виконувати багато операцій одночасно і працювати безперервно. Виконання автоматизованого тестування визначає, чи було отримано відповідні результати, та чи може бути проведено об'єднання призвести до успіху або збою інтеграції [3]. Машинне навчання (ML) та інші технології штучного інтелекту (ШІ) можуть сприяти ширшому покриттю тестування і виявленню багів, які можуть знизити рівень мануального тестування. Нарешті, автоматизоване тестування має важливе значення для збереження ресурсів[4].

Згідно з Міжнародної асоціації тестування програмного забезпечення (ISTQB), тестування програмного забезпечення – це процес у життєвому циклі розробки програмного забезпечення, який оцінює якість компонента чи системи та відповідних робочих продуктів[5].

Можна виділити такі ключові характеристики автоматизації тестування такі як ефективність, точність, повторюваність, інтеграція, звітність. Автоматизація

тестування може значно скоротити час, необхідний для виконання тестових випадків, що може призвести до економії коштів та ресурсів. Ефективність автоматизації полягає в здатності значно скоротити час, потрібний для виконання тестових випадків, що призводить до економії коштів та ресурсів.

Точність автоматизованих тестів гарантується їх послідовністю та відсутністю помилок, що дозволяє виявити більше дефектів, що можуть бути пропущені при ручному тестуванні.

Повторюваність є важливою особливістю автоматизації, оскільки тести можна запускати безліч разів без жодних змін, що дозволяє ефективно проводити тестування в різних середовищах та відстежувати зміни з часом.

Можливість легкої інтеграції з іншими процесами розробки, такими як безперервна інтеграція та безперервна доставка (CI/CD), сприяє автоматизації процесу тестування та робить його більш ефективним.

Звітність автоматизованих тестів гарантується генерацією детальних звітів про результати тестування, що допомагає розробникам швидко знаходити та виправляти помилки.

Тестування програмного забезпечення є важливою частиною процесу розробки ПЗ, оскільки воно допомагає забезпечити якість та надійність продукту перед його випуском на ринок. В інформаційній епохи, коли програмне забезпечення стає все більш складним та різноманітним, ефективність тестування вирішально важлива. Існують різні види тестування (див. рис. 1.1), кожен з яких спрямований на перевірку певних аспектів програмного забезпечення.

Рисунок 1.1 – Види тестування (за даними [6])

Один з найпоширеніших видів тестування – це функціональне тестування, яке перевіряє, чи відповідає програмне забезпечення очікуваній поведінці та чи виконує всі свої функції відповідно до специфікацій. Під час функціонального тестування перевіряються різні аспекти програмного забезпечення, такі як його основні функції, реакція на введення користувача, обробка даних та виведення результатів. Тестування проводиться шляхом виконання різних тестових сценаріїв або випадків, які покривають різні можливі варіанти використання програмного забезпечення. Функціональне тестування може бути виконане як вручну, так і за допомогою автоматизованих тестів, в залежності від складності програмного продукту та доступних ресурсів.

Нефункціональне тестування спрямоване на перевірку нефункціональних аспектів програмного забезпечення, таких як його продуктивність, безпека, надійність, зручність використання та сумісність. Наприклад, тестування навантаження допомагає визначити, як програма працює під навантаженням, а тестування безпеки допомагає виявити вразливості програмного забезпечення.

Системне тестування перевіряє, чи працює вся система програмного забезпечення разом як єдине ціле та чи відповідає вона загальним вимогам. Цей вид тестування важливий, оскільки він дозволяє переконатися, що всі компоненти програмного забезпечення працюють правильно та взаємодіють один з одним.

Приймальне тестування проводиться користувачами або замовниками програмного забезпечення з метою переконатися, що воно відповідає їхнім потребам та очікуванням. Цей вид тестування дозволяє забезпечити задоволення клієнта та виявити можливі проблеми з функціональністю чи іншими аспектами програмного забезпечення.

Регресійне тестування проводиться після внесення змін до програмного забезпечення для переконання, що ці зміни не призвели до появи нових помилок або не вплинули на роботу існуючих функцій. Автоматизоване тестування використовує програмні інструменти для автоматизації процесу тестування, що дозволяє прискорити процес та покращити ефективність тестування.

Важливо зазначити, що автоматизоване тестування має свої переваги та обмеження. З одного боку, воно дозволяє здійснювати швидке та повторюване тестування, що значно збільшує ефективність розробки програмного забезпечення. З іншого боку, не всі аспекти тестування можна автоматизувати, особливо ті, які вимагають людського експертного оцінювання або творчого підходу.

Наприклад, дослідницьке тестування, що ґрунтується на творчому підході та експертному оцінюванні, складно або неможливо автоматизувати через його залежність від інтуїції та досвіду тестувальників. Такий процес тестування вимагає специфічних знань про систему, виявлення особливостей функціональності, яка досягається шляхом глибокого занурення в продукт.

Тестування користувацького інтерфейсу, яке включає оцінку зручності використання та якості інтерфейсу користувача, також може бути складним для автоматизації. Це пов'язано з суб'єктивністю оцінки та необхідністю взаємодії з реальними користувачами, що ускладнює створення автоматизованих сценаріїв тестування. Важливо враховувати індивідуальні потреби та переваги різних категорій користувачів, а також забезпечити, щоб інтерфейс був інтуїтивно зрозумілим та легким у використанні для всіх користувачів. Таке тестування може включати аналіз реакцій користувачів на взаємодію з інтерфейсом, оцінку часу виконання різних завдань, а також перевірку відповідності інтерфейсу вимогам дизайну та стандартам доступності. Додатково, тестування користувацького інтерфейсу може включати перевірку реакції на різні розміри та типи екранів, адаптацію інтерфейсу до різних мов та культурних особливостей, а також взаємодію з різними пристроями та платформами.

Тестування в реальних умовах також не завжди може бути автоматизоване через потребу в специфічних умовах або доступі до реального обладнання. Наприклад, для тестування взаємодії програмного забезпечення з апаратним забезпеченням або мережевим середовищем може бути необхідно створювати спеціальні умови, що ускладнює автоматизацію цих процесів.

Також тестування з експертною оцінкою може бути непридатним для автоматизації через суб'єктивність та неоднорідність оцінки. Деякі тести вимагають експертного бачення та досвіду спеціалістів для оцінки відповідності програмного забезпечення вимогам або стандартам якості, що ускладнює їхню автоматизацію.

Завдяки постійному розвитку технологій штучного інтелекту та машинного навчання, стає можливим автоматизувати все більше тестових процесів, включаючи виявлення дефектів, генерацію тестових даних, аналіз результатів тестування, створення автоматизованих тестів. Це робить процес тестування ще більш ефективним та допомагає забезпечити високу якість кінцевого програмного продукту.

Оскільки організації прагнуть поставляти програмні продукти високої якості прискореними темпами, роль ШІ в автоматизації тестування стає все більш вирішальним [7]. Використання автоматизації тестування ПЗ з використанням штучного інтелекту є необхідним кроком при розробці програмного забезпечення. Це дозволяє підвищити ефективність, точність та повторюваність тестування, що в свою чергу призводить до поліпшення якості та швидкості розробки програмного забезпечення.

1.2 Дослідження та аналіз застосування методів штучного інтелекту для автоматизації тестування програмного забезпечення

Термін "штучний інтелект" був введений Джоном Маккарті в 1955 році на конференції, організованій Дартмутським університетом. Цей термін використовувався для позначення всіх "систем програмування, в яких машина імітує деяку розумну поведінку людини". За словами Джона Маккарті, це "наука та інженерія створення інтелектуальних машин, особливо інтелектуальних комп'ютерних програм" [8]. Як науковий напрям, машинне навчання виросло з пошуків штучного інтелекту [9]. Штучний інтелект (ШІ) відноситься до галузі науки, яка займається розробкою і використанням комп'ютерних алгоритмів та моделей для моделювання імітації розумової активності людини. Головна ідея ШІ

полягає в тому, щоб дати комп'ютерам здатність розуміти, аналізувати та приймати рішення на основі великої кількості даних швидко та ефективно, подібно до того, як це робить людина.

Основні складові ШІ включають машинне навчання, глибоке навчання, нейронні мережі, обробку природної мови і розпізнавання образів (див. рис. 1.2).

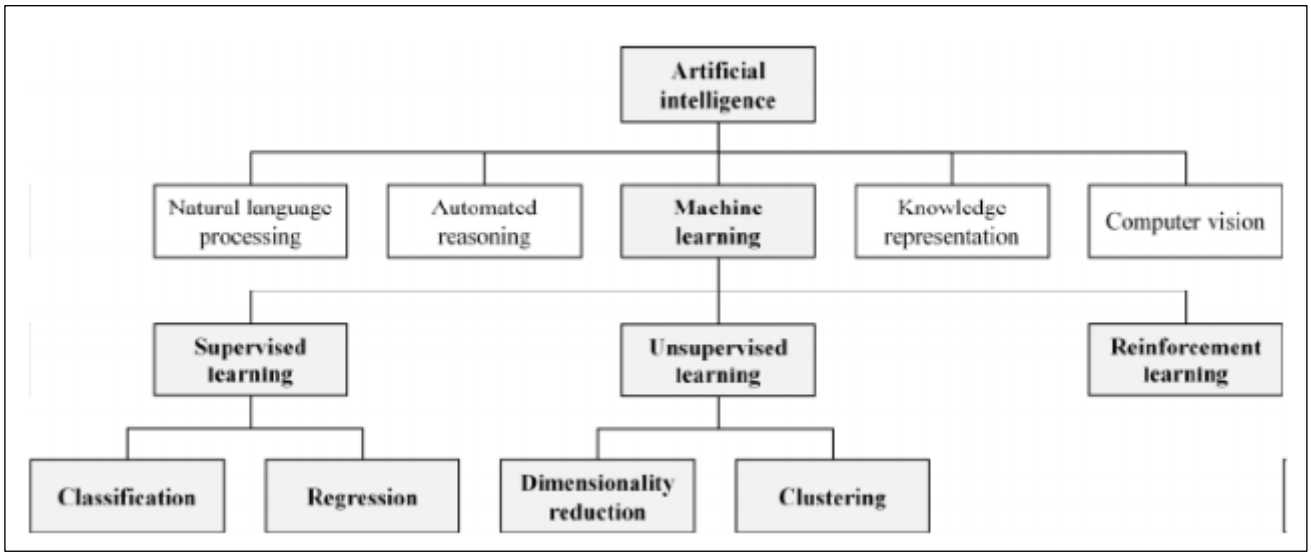

Рисунок 1.2 – Категоризація штучного інтелекту та машинного навчання (за даними [10])

Ці методи дозволяють комп'ютерам вчитися і вдосконалювати свої здібності, враховуючи нові дані та взаємодіючи з навколишнім середовищем.

Методи штучного інтелекту використовуються в різних областях, включаючи обробку даних, фінанси, медицину, автоматизацію, гральну індустрію та багато інших. Штучний інтелект використовується для розробки автономних систем, оптимізації бізнес-процесів, вирішення складних задач та прийняття рішень.

Ключові концепції штучного інтелекту включають [11]:

– машинне навчання: методи, що дозволяють комп'ютерам самостійно вчитися на основі даних та навчитися робити прогнози та вирішувати завдання без явного програмування (див. рис. 1.3);

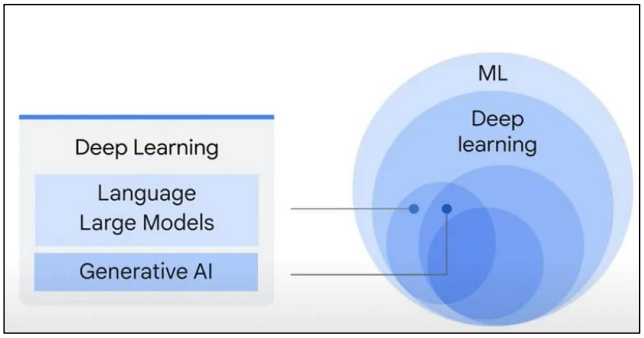

Рисунок 1.3 – Великі мовні моделі та генеративний штучний інтелект в структурі ML (за даними [9])

– глибоке навчання: піднабір машинного навчання, який використовує нейронні мережі для моделювання та імітації складних функцій розумової активності людини [12]. Великі мовні моделі (LLM) і генеративний штучний інтелект (GenAI) є двома важливими частинами глибокого навчання. Генеративні системи ШІ та великі мовні моделі є підмножинами глибокого навчання. Мовні моделі, такі як GPT-3, GPT-4 і Google Gemini, зосереджені на створенні тексту, тоді як генеративні системи ШІ, такі як DALL-E і Midjourney, створюють зображення. GitHub Copilot створює код тощо. Більшість LLM створюють свої результати у відповідь на підказки;
– нейронні мережі: математичні моделі, що імітують структуру та функціональність нейронної системи людини для обробки інформації та прийняття рішень;
– обробка природної мови: галузь ШІ, що вивчає комп'ютерне розуміння та генерацію людської мови;
– розпізнавання образів: галузь ШІ, що вивчає комп'ютерне розпізнавання або інтерпретацію зображень та відео.

1.2.1 Методи машинного навчання

Методи машинного навчання дозволяють вирішувати завдання, пов'язані з призначенням об'єктів чи явищ до певних категорій чи класів на основі їх

характеристик. Основна мета класифікації полягає в тому, щоб навчити модель розрізняти об'єкти з різних класів та правильно призначати їм мітки класів. Це може бути важливим у багатьох сферах, включаючи медицину, фінанси, маркетинг, а також у сфері інтернет-технологій та тестування де необхідно робити прогнози, класифікувати користувачів або виявляти аномальні патерни. Методи класифікації включають в себе широкий спектр алгоритмів, від класичних до сучасних, таких як наївний Байєсівський класифікатор, логістична регресія, дерева рішень, метод опорних векторів, нейронні мережі та ансамблеві методи. Кожен з цих методів має свої переваги та обмеження, і вибір конкретного алгоритму залежить від властивостей даних та поставленої задачі.

Методи машинного навчання можна класифікувати за різними критеріями, включаючи тип завдання, яке вони вирішують, та підхід до навчання. За типом завдання методи машинного навчання можна поділити на три основні категорії[13]: навчання з учителем, навчання без учителя та навчання з підкріплеенням [14] (див. рис 1.4). На практиці існує ще один тип навчання - напівкероване навчання.

Рисунок 1.4 – Класифікація методів машинного навчання (за даними [13])

Навчання з учителем базується на наявності надійного набору вхідних даних та відповідних вихідних міток. У цьому методі модель навчається на основі вхідно-вихідних пар, що дозволяє алгоритму навчатися і вивчати зв'язки між вхідними даними та відповідними вихідними значеннями. Процес навчання

полягає в пошуку оптимальних параметрів моделі, які найкраще відображають вхідні дані та забезпечують точність передбачень. Навчання з учителем використовується в багатьох областях, включаючи класифікацію, регресію, виявлення аномалій та інші завдання, де необхідно встановити зв'язок між вхідними та вихідними даними [15].

Неконтрольоване навчання (навчання без учителя) відрізняється від інших методів машинного навчання тим, що воно відбувається без явного надання зворотного зв'язку, і машина чи алгоритм самостійно вивчає закономірності у вихідних даних. Під час застосування неконтрольованого навчання відбувається реструктуризація даних, що може призвести до виявлення нових особливостей в наборі даних.

Навчання з підкріпленням ґрунтується на заохоченнях або покараннях, які отримує програма у вигляді зворотного зв'язку. Цей метод можна описати як навчання методом спроб і помилок, де машина під час навчання визначає, наскільки правильними були її дії, використовуючи зазначений зворотний зв'язок.

Напівконтрольоване навчання заповнює прогалину між контрольованим і неконтрольованим навчанням, коли є шум, внесений набором даних, або дані є неповними, що призводить до того, що деякі або більшість входів не мають пов'язаних з ними виходів.

Машинне навчання має свої обмеження, які включають у себе представлення, перенавчання та неадекватне узагальнення. Ці обмеження необхідно враховувати, оскільки вони впливають на здатність алгоритмів машинного навчання до розв'язання завдань та узагальнення результатів.

1.3 Застосування штучного інтелекту в автоматизації тестування ПЗ

Застосування штучного інтелекту в автоматизації тестування ПЗ відкриває нові можливості для поліпшення процесу тестування. Застосування методів ШІ дозволяє знизити зусилля та час, необхідний для створення тестових сценаріїв, написання автоматизованих тестів, покращити точність використання тестових

даних і виявлення помилок, а також забезпечити високу якість програмного забезпечення.

IT-компанія SoftServe провела внутрішнє дослідження, яке показало, що генеративний штучний інтелект (Generative AI) може збільшити продуктивність роботи команд розробників до 45% [16].

У ході 1500 проведених експериментів було встановлено, що впровадження генеративного штучного інтелекту в діяльність виробничих команд може сприяти підвищенню їх продуктивності у всіх аспектах роботи. В результаті цього, середньо, час, необхідний для виконання завдань усією командою, може бути скорочений на 31%, тоді як ефективність командної роботи може збільшитись на 45% [16] (див. рис. 1.5).

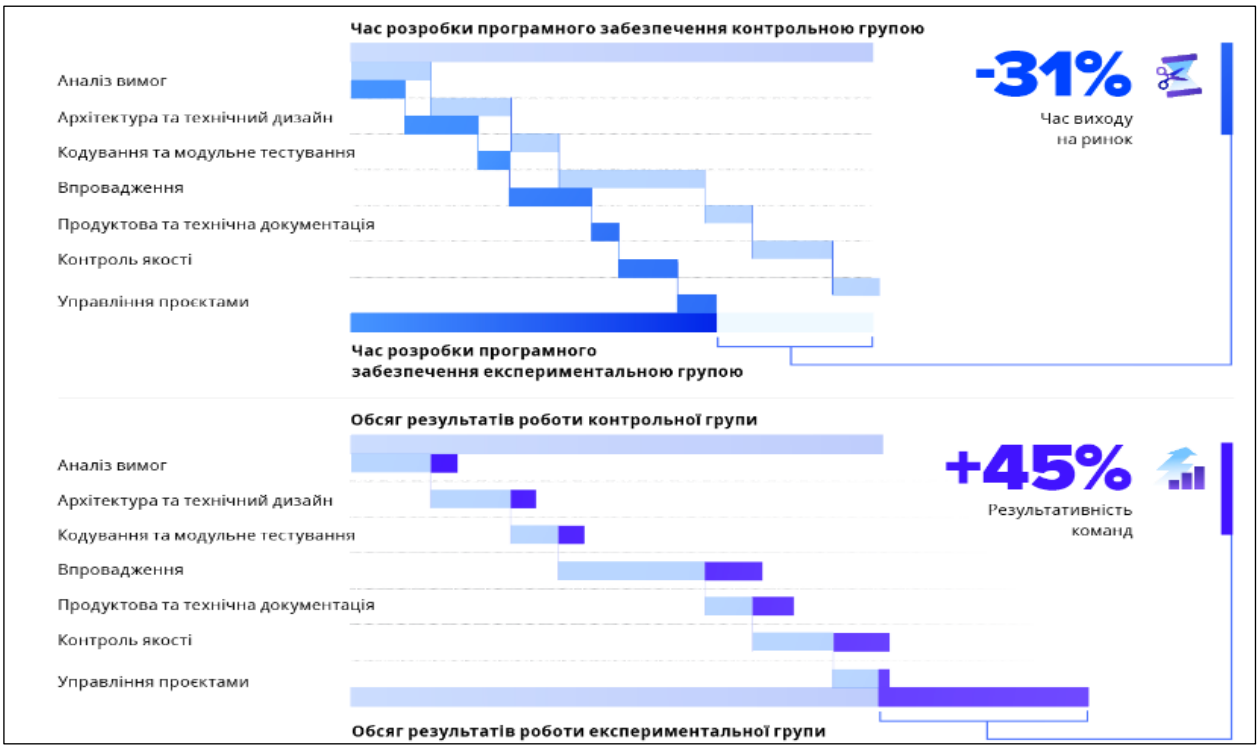

Рисунок 1.3 – Результати роботи команд розробників ПЗ з використанням ШІ (за даними [16])

Основні області застосування штучного інтелекту в тестуванні ПЗ включають [17]:

– автоматична генерація тестових сценаріїв: ШІ може використовуватися для створення автоматичних тестів, що генеруються з використанням

алгоритмів машинного навчання. Це дозволяє знизити зусилля, пов'язані з ручним створенням тестових сценаріїв [18], а також забезпечити більш широке покриття тестування;

- зневаги до помилок: ШІ може аналізувати результати тестування та відразу діагностувати та розпізнати помилки. Це дозволяє швидше виявляти і виправляти проблеми, що можуть бути пропущені традиційними методами тестування;
- оптимізація тестування: ШІ може визначити найбільш ефективні шляхи виконання тестових сценаріїв, враховуючи різні фактори, такі як пріоритети, покриття, ризики та обмеження [17]. Це дозволяє економити час і зусилля і забезпечує оптимальне використання ресурсів;
- тестування в реальному часі: Застосування ШІ дозволяє виконувати тестування в режимі реального часу, аналізувати дані, що надходять в режимі реального часу, і виявляти відхилення або помилки. Це особливо корисно в системах, де важлива низька латентність та відповідь в режимі реального часу;
- автоматичне виявлення помилок: ШІ може використовуватися для автоматичного виявлення та локалізації помилок у програмному забезпеченні. Він може аналізувати траси виконання програми, моніторити систему та виявляти нетипову поведінку або відхилення;
- генерація тестових даних: ШІ може бути використано для автоматичної генерації тестових даних [19, 20], які потрібні для виконання тестування. Це може бути корисним для тестування програм з великою кількістю вхідних даних або програм, для яких складно знайти репрезентативні тестові дані;
- тестування реалістичних сценаріїв: ШІ дозволяє моделювати та відтворювати реалістичні сценарії тестування, включаючи різні умови та взаємодії з системою [21]. Це допомагає забезпечити, що програмне забезпечення буде успішно працювати в реальних умовах;

- аналіз тестових даних: ШІ може бути використано для аналізу та обробки тестових даних, щоб виявити патерни, тенденції та незвичайне поведінку. Це допомагає зрозуміти результати тестування та шукати можливі проблеми чи вразливості в програмному забезпеченні;
- генерація та виконання тестових скриптів: ШІ може бути використано для автоматичної генерації та виконання тестових скриптів. Він може розпізнавати поведінку програми та створювати тестові скрипти, які максимально охоплюють широкий спектр можливих сценаріїв;
- підтримка тестування продуктивності та масштабованість: ШІ може бути використано для тестування продуктивності та масштабованості програмного забезпечення. Він може моделювати та симулювати різні навантаження та обсяги даних [22], щоб оцінити, як програма веде себе в різних умовах.

Застосування штучного інтелекту в автоматизованному тестуванні ПЗ є важливим напрямком розвитку, який дозволяє покращити ефективність тестування та забезпечити високу якість програмного забезпечення. Вивчення та застосування методів ШІ в цій галузі є актуальним і має великий потенціал для подальшого розвитку. Однак, варто пам'ятати, що збалансоване поєднання традиційних підходів тестування та штучного інтелекту є найбільш оптимальним для досягнення кращих результатів тестування.

1.4 Постановка задачі дослідження

Метою роботи є дослідження використання методів штучного інтелекту для генерації автоматизованих тестів для використанні в тестуванні програмного забезпечення. В результаті проведеного дослідження першоджерел виявилось, що використання методів штучного інтелекту може вплинути на удосконалення підходу розробки автоматизованих тестів, що забезпечить продуктивність написання тестів, збільшить якість коду та використання технік тест дизайну при генерації автотестів. Тому використання методів ШІ може підвищити

ефективність написання тестових скриптів так і самого тестування застосунків в цілому.

Автоматизація тестування програмного забезпечення (ПЗ) є важливим етапом в процесі розробки та впровадження програмних продуктів, оскільки дозволяє підвищити ефективність, знизити витрати та збільшити якість ПЗ. Цей підхід передбачає використання спеціальних програмних засобів для автоматизації виконання тестів, які раніше виконувались вручну. Автоматизація тестування дозволяє швидше виявляти помилки та дефекти в програмному коді, забезпечуючи тим самим збільшення продуктивності розробки та підвищення якості продукту. Крім того, автоматизовані тести можуть бути легко повторюваними, що дозволяє швидко перевіряти ПЗ на відповідність функціональним вимогам та виявляти проблеми на ранніх етапах розробки. Таким чином, автоматизація тестування є важливим інструментом для забезпечення якості та надійності програмного забезпечення, а також зменшення ризиків інцидентів під час його експлуатації.

Метою роботи є дослідження методів ШІ, за допомогою яких можливо згенерувати набори автоматизованих тестових сценаріїв, які можна буде використати для тестування веб застосунку і оцінити отримані результати. Для досягнення мети планується створити тестовий фреймворк з використанням Page Object моделі, для запуску наборів автоматичних тестів, які будуть згенеровані трьома різними мовними моделями. Тести будуть створені на основі Use Cases, за допомогою промптів, та подальшим уточненням промптів, до моменту поки автоматичний тест стане придатним для тестування веб застосунку.

Для досягнення цілі дослідження виконаємо уточнення задач, які необхідно виконати в роботі.

Задача 1. Визначити метод ШІ, який буду використовуватись у дослідженні.

Задача 2. Необхідно визначити мовні моделі, які будуть використовуватись у дослідженні.

Задача 3. Визначити Use Cases для побудови на основі них автоматизованих тестів.

Задача 4. Створити оточення для виконання отриманих тестів, та збору інформації про результати виконання тестів.

Задача 5. Аналіз отриманих результатів та визначення висновків роботи.

В роботі планується дослідити використання трьох мовних моделей, для генерації автоматичних тестів для тестування веб застосунку. Це дозволить визначити доцільність використання методів ШІ в автоматизації тестування ПЗ, оцінити швидкість створення автоматичних тестових сценаріїв, якість виконання тестів, здібність та швидкість знаходження багів.

2 ЗАСТОСУВАННЯ МЕТОДІВ ШІ ДЛЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз потреби в автоматизації тестування ПЗ

Перш за все, з метою отримання інформації про дослідження методів ШІ в автоматизації тестування ПЗ, необхідно провести аналіз потреби в цій автоматизації. Це включає оцінку існуючих тестових процесів, виявлення проблем та визначення потенційних переваг, які можуть бути досягнуті за допомогою автоматизації.

Під час аналізу потреби в автоматизації тестування ПЗ необхідно врахувати наступні аспекти:

- обсяг та складність проекту: Якщо проект великий або має складність, автоматизація тестування може допомогти збільшити продуктивність тестового процесу та знизити його тривалість. Великі обсяги тестування, низька частота помилок та потреба у повторюваності тестових сценаріїв є також чинниками, які вказують на потребу в автоматизації;
- частота випуску нових версій ПЗ: Якщо організація постійно випускає нові версії ПЗ або проводить регулярні оновлення, автоматизований тестовий процес може забезпечити швидку та ефективну перевірку нових функцій та забезпечити швидкість випуску;
- підтримка різних пристроїв та платформ: Якщо ПЗ потрібно перевіряти на різних пристроях, операційних системах та конфігураціях, автоматизація тестування може допомогти покрити широкий спектр тестових сценаріїв та забезпечити сумісність програмного забезпечення з різними середовищами;
- вимоги до якості та стабільності: Якщо ПЗ потрібно перевіряти на високу якість та стабільність, щоб задовольнити потреби користувачів, автоматизація допоможе забезпечити повторюваність і послідовність тестів та швидкість виявлення помилок;

- вартість та ресурси [23]: Автоматизація тестування може знадобитися значні ресурси, включаючи інвестиції у інфраструктуру, навчання команди тестувальників та підтримку автоматизованих тестових скриптів. Тому, вимоги до бюджету та доступність ресурсів також слід враховувати при аналізі.

Після проведення аналізу потреби в автоматизації тестування ПЗ можна деталізувати завдання та поставити конкретні цілі для використання ШІ в тестуванні.

2.2 Вибір методів штучного інтелекту для застосування в тестуванні

Після аналізу потреби в автоматизації тестування ПЗ можна приступити до вибору методів штучного інтелекту (ШІ), які можуть застосовуватися в тестуванні. Основні методи ШІ, що використовуються в тестуванні, включають:

- машинне навчання (Machine Learning): Машинне навчання використовується для навчання комп'ютерних систем розпізнавати певні патерни та зробити висновки на основі навчальних даних. Цей метод може бути застосований для автоматичного виявлення та класифікації помилок, прогнозування можливих проблем та оцінки якості ПЗ;
- нейронні мережі (Neural Networks): Нейронні мережі можуть бути використані для моделювання складних взаємозв'язків у тестуванні ПЗ. Вони можуть бути навчені розпізнавати складні міжзалежності в тестових даних та зробити висновки на основі цих залежностей;
- натуральна мова та обробка мови (Natural Language Processing, NLP): NLP використовується для аналізу та розуміння природної мови, що дозволяє автоматично аналізувати та оцінювати тестову документацію, звіти про помилки та іншу текстову інформацію, що пов'язана з тестуванням;
- генетичні алгоритми (Genetic Algorithms): Генетичні алгоритми можуть бути застосовані для оптимізації тестових сценаріїв та виявлення

оптимальних послідовностей тестів. Вони можуть створювати нові комбінації та відбирати найпродуктивніші з них;

– аналіз даних та великі дані (Data Analytics, Big Data): Аналіз даних та великі дані можуть використовуватися для виявлення спільних міжзалежностей, трендів та шаблонів у великих обсягах даних, що пов'язані з тестуванням. Цей метод може допомогти виявити ризики та покращити ефективність тестування.

При виборі методів ШІ для застосування в тестуванні необхідно враховувати особливості конкретного проекту та потреби тестування. Важливо також врахувати доступність експертних знань та ресурсів для реалізації обраного методу. Поєднання кількох методів ШІ може бути ефективним для досягнення найкращих результатів в тестуванні ПЗ.

2.3 Опис множини критеріїв для задачі багатокритеріального вибору.

Введення багатокритеріальної задачі вибору передбачає наявність кількох конкуруючих критеріїв, які мають бути оптимізовані для досягнення найкращого результату у контексті цільової системи. Ці критерії можуть бути представлені різними метриками, такими як точність, час, адаптивність, масштабованість, вартість тощо.

Точність і надійність виявлення дефектів може бути оцінена за шкалою від 0 до 3, де значення 1 вказує на 0-50% точність, значення 2 - на 50% - 80%, і значення 3 - на 80% - 99.5% точність. Щодо швидкодії, вона також може мати значення від 1 до 3, де 1 означає низьку продуктивність, 2 - середню, і 3 - високу швидкодію та ефективність. Адаптивність до змін у програмному коді також може бути оцінена за шкалою від 1 до 3, де 1 вказує на низьку адаптивність, 2 - на середню, і 3 - на високу. Щодо масштабованості, вона також оцінюється за шкалою від 1 до 3, де 1 показує ефективність при невеликій кількості тестових даних, 2 - при середній, і 3 - при великій. Вартість впровадження може бути визначена за шкалою від 1 до 3, де 1 вказує на високі витрати понад 100 тис.

гривень, 2 - на середні 50-100 тис. гривень, і 3 - на низькі менше 50 тис. гривень. Відобразимо дані в таблиці (див. табл. 2.1)

Таблиця 2.1 – Векторний опис альтернатив(таблиця виконана самостійно)

Назва методу	Accuracy and Reliability	Performance	Adaptability	Scalability	Implementation Cost
1	2	3	4	5	6
Machine Learning	85	Висока	Висока	Висока	Низькі
Neural Networks	82	Висока	Середня	Висока	Низькі
Support Vector Machines	76	Середня	Середня	Середня	Високі
Natural Language Processing	83	Висока	Висока	Висока	Високі
Logic Programming	81	Низька	Середня	Висока	Середня

Згідно аналізу шкал ми можемо поділити наші показники на три групи: низька (1), середня (2) і висока (3), точність та надійність ми також можемо поділити на три групи згідно опису. Для показника вартість впровадження шкала буде виглядати – низька (3), середня (2) і висока (1). Створимо таблицю де відобразимо зміни. Відобразимо в таблиці для кожної критерії його вагу (див табл. 2.2). Точність та надійність, швидкодія це найбільш впливові та важливі критерії тому їх вага буде дорівнювати 3, адаптивність, масштабованість і вартість провадження дорівнює 2, як не суттєво важливими.

Проаналізувавши критеріальну таблицю згідно з множиною Парето ми бачимо альтернативи, які мають оцінки по всім критеріям гірші, ніж інші альтернативи. Такі альтернативи ми можемо назвати неконкурентоспроможними і їх можна прибрати з таблиці. Такими альтернативами будуть Метод опорних векторів та Логічне програмування.

Таблиця 2.2. – Векторний опис альтернатив(порядкова шкала) (таблиця виконана самостійно)

	Accuracy and Reliability	Performance	Adaptability	Scalability	Implementation Cost
Machine Learning	3	3	3	3	3
Neural Networks	3	3	2	3	2
Support Vector Machines	2	2	2	2	1
Natural Language Processing	3	3	3	3	1
Logic Programming	3	1	2	3	2
Вагові коефіцієнти	3	3	2	2	2

Нормалізуємо наявні критерії та приведемо їх у шкалу від 0 до 1. Використаємо для цього формулу 2.1:

$$f = \frac{f_{вим}}{f_{ет}} \qquad (2.1)$$

де fвим – показник критерію,

fет – еталонне значення.

За еталонне значення ми взяли 3, як найбільш впливовий варіант в нашому розрахунку. Для нашого дослідження ми вирішили обрати лінійну адитивну згортку з ваговими коефіцієнтами. Це обумовлено тим, що цей метод є широко використовуваним і вже довів свою ефективність у вимірюванні. Зокрема, нам важливо врахувати, що наші критерії мають різну вагу та значення при виборі методів штучного інтелекту в автоматизації тестування програмного.

Наприклад, ми вважаємо, що найбільший акцент слід робити на точності, оскільки нам потрібна надійна робота нашої системи, щоб уникнути зайвих

перевірок результатів та швидкодія, для швидкого виконання алгоритму для більш швидкого створення необхідних тест кейсів для цих, адаптивність, масштабованість і вартість провадження дорівнює 2, як не суттєво важливими.

Створимо нову таблицю(див. табл. 2.3) та заповнимо її нормалізованими даними та розрахуємо вагові коефіцієнти методом простого ранжування.

Таблиця 2.3 – Оцінка алгоритмів за обраними критеріями (нормалізовані критерії) (таблиця виконана самостійно)

	Accuracy and Reliability	Performance	Adaptability	Scalability	Implementation Cost
Machine Learning	1	1	1	1	1
Neural Networks	1	1	0,67	1	0,67
Natural Language Processing	1	1	1	1	0,33
Вагові коеф.	0,25	0,25	0,17	0,17	0,17

Проведемо розрахунки за формулою 2.2 лінійної адитивної згортки з ваговими коефіцієнтами:

$$Z = \max \sum_{j=0}^{n} \alpha_j \beta_j a_{ij} \qquad (2.2)$$

де α_j і β_j – вагові коефіцієнти для кожного j,j-го елемента,

α_{ij} – значення елементів, що залежать від індексів i та j,

n – кількість елементів.

Machine Learning = 1*0,25+1*0,25+1*0,17+1*0,17+1*0,17 = 1,00

Neural Networks = 1*0,25+1*0,25+0,67*0,17+1*0,17+1*0,17 = 0,95

Natural Language Processing = 1*0,25 + 1*0,25 + 1*0,17+1*0,17 + 0,33*0,17 = 0,896

Під час виконання задачі багатокритеріального вибору, були обрані альтернативи та критерії їхньої оцінки. Всі критерії нормалізувалися та приведені до шкали від 0 до 1. Ці дані використовувалися для розрахунку корисності альтернатив за лінійною адитивною згорткою з ваговими коефіцієнтами.

За результатами проведених розрахунків можна визначити, що найкращою альтернативою для нашого дослідження є використання методу Machine Learning. В той час як Метод опорних векторів та логічне програмування, відповідно до множини Парето, можна не розглядати, оскільки вони по всім критеріям гірші, ніж інші альтернативи.

Слід відзначити, що без фактичного виконання експерименту та оцінки результатів роботи алгоритму можна лише приблизно оцінити та дати більш суб'єктивну оцінку, наскільки нам підходить той чи інший метод. Це через те, що в дослідженні використовувалися здебільш якісні характеристики, та спиралися на інформацію щодо роботи вибраних алгоритмів, знайдені зовнішніх ресурсах.

2.4 Математична модель методів штучного інтелекту для автоматизації тестування програмного забезпечення

Мета даного підрозділу – ретельно розглянути та науково обґрунтувати математичну модель методів штучного інтелекту, спрямовану на автоматизацію тестування програмного забезпечення. Під час вивчення цієї моделі ми звернемо увагу на визначення та формалізацію ключових показників ефективності тестових процесів з використанням ШІ. Аналіз такої моделі дозволить врахувати не лише кількісні аспекти виявлення помилок, але й вартість розробки та підтримки тестових сценаріїв, що відкриє шлях до створення оптимальних стратегій тестування в умовах використання штучного інтелекту.

Для формулювання математичної моделі методів штучного інтелекту (ШІ), що використовуються для автоматизації тестування програмного забезпечення, визначимо п'ять ключових параметрів, які мають критичне значення для оцінки ефективності цього процесу.

Час виявлення помилок (EDT) визначається як період, необхідний для ідентифікації помилок в програмному забезпеченні. Цей параметр вимірюється у годинах або днях та враховує час, який потрібно для виявлення та виправлення помилок в процесі тестування.

Точність виявлення помилок (EDA) визначається як відсоток правильно виявлених помилок за допомогою методів штучного інтелекту. Цей параметр відображає ефективність системи у виявленні реальних помилок та вимірюється у відсотках.

Вартість розробки та підтримки тестових сценаріїв (CDS) враховує сумарні витрати на створення та утримання тестових сценаріїв, які використовуються для автоматизації тестування з використанням ШІ.

Ефективність використання ресурсів (EUR) визначає ступінь використання обчислювальних ресурсів для автоматизації тестування з штучним інтелектом. Цей показник вимірюється у відсотках та відображає оптимальне використання доступних обчислювальних потужностей.

Загальна складність моделі (GCM) є комплексним показником, що враховує всі вищезазначені параметри та визначає загальну ефективність математичної моделі методів штучного інтелекту для автоматизації тестування програмного забезпечення. Цей параметр визначається з урахуванням взаємозв'язку і вагомості кожного із попередніх показників.

Побудова математичної моделі.

Побудуємо математичний вираз 2.3 для розрахунку загального часу виявлення помилок (EDT):

$$\mathrm{EDT} = \frac{1}{\mathrm{EDA}}, \qquad (2.3)$$

де EDA – точність виявлення помилок.

Чим більше значення Точності виявлення помилок (EDA), тим менше буде EDT, що вказує на більш ефективний та швидкий процес виявлення помилок.

Побудуємо математичний вираз 2.4 для розрахунку точності виявлення помилок (EDA):

$$\mathrm{EDA} = \frac{\mathrm{DQ}}{\mathrm{D}} * 100\%, \qquad (2.4)$$

де DQ – число правильно виявлених помилок,

D – загальна кількість помилок.

Побудуємо математичний вираз 2.5 для розрахунку ефективності використання ресурсів (EUR):

$$\mathrm{EUR} = \frac{\mathrm{Rf}}{\mathrm{Rp}} * 100\%, \qquad (2.5)$$

де Rf – фактичне використання ресурсів,

Rp – планове використання ресурсів.

Ця формула вимірює ефективність, з якою використовуються ресурси в порівнянні з тим, що було заплановано.

Фактичне використання ресурсів вказує на обсяг ресурсів (час, матеріали, фінанси тощо), який був реально використаний під час виконання роботи або проекту.

Плановане використання ресурсів відображає кількість ресурсів, яка передбачалася до використання згідно з планом або бюджетом.

Результатом цього виразу є відсоток, який показує, наскільки ефективно були використані ресурси. Якщо цей показник становить 100%, це означає, що ресурси були використані точно за планом. Якщо показник більше 100%, це може свідчити про перевикористання ресурсів, а якщо менше – про їх недовикористання.

Побудуємо математичний вираз 2.6 для розрахунку загальної складності моделі(GCM):

$$GCM = \frac{EDT*(1-EUR)}{CDS}, \quad (2.6)$$

де – EDT – загальний час виявлення помилок,

EUR – коеф. ефективності використання ресурсів,

CDS – вартість розробки тестових сценаріїв.

Ця формула визначає загальну складність моделі, враховуючи вплив часу виявлення помилок, ефективності використання ресурсів та вартості розробки тестових сценаріїв. Чим менше значення GCM, тим меншою вважається загальна складність моделі.

На основі результатів проведених розрахунків можна зробити висновок, що найбільш ефективним рішенням для нашого дослідження є використання методу машинного навчання. Метод опорних векторів та логічне програмування, відповідно до множини Парето, можна відкинути, оскільки вони за всіма критеріями гірші, ніж інші альтернативи. Важливо зауважити, що без проведення фактичного експерименту та оцінки результатів роботи алгоритму можна лише наблизити та надати більш суб'єктивну оцінку ефективності кожного методу. Це зумовлено тим, що дослідження базувалося переважно на якісних характеристиках та інформації про функціонування обраних алгоритмів, яку було отримано з зовнішніх джерел.

3 ПРАКТИЧНЕ ДОСЛІДЖЕННЯ

3.1 Постановка експерименту

В попередньому розділі було проведено аналіз різних методів штучного інтелекту (ШІ) для автоматизації тестування програмного забезпечення (ПЗ). На основі отриманих результатів, метод машинного навчання (ML) визначено як найефективніший підхід для автоматизації процесу тестування.

З огляду на це, пропонується провести подальші дослідження в напрямку використання великих мовних моделей для генерації автоматичних тестів. LLM, такі як GPT-3.5, GPT-4 та Gemini, володіють значним потенціалом для генерування тестових випадків, які охоплюють широкий спектр функціональних та нефункціональних аспектів ПЗ.

Особливо перспективними для дослідження є:

GPT-3.5: Ця модель володіє значною мовною моделлю, що робить її здатною генерувати тестові кейси з високою точністю.

GPT-4: Завдяки покращеним алгоритмам та більшому набору даних, GPT-4 може генерувати більш складні та реалістичні тестові кейси.

Gemini: Ця модель, розроблена Google AI, володіє унікальними можливостями для генерування тестових кейсів, які враховують контекст та специфіку ПЗ.

Практичне дослідження, запропоноване у даній роботі, необхідне для визначення ефективності використання мовних моделей GPT-3.5, GPT-4 та Gemini у процесі автоматизації тестування програмного забезпечення. Основною метою дослідження є створення автоматизованих тестів на основі реальних сценаріїв використання, які відображають різноманітні вимоги та функціональні можливості програмного продукту. Це дозволить визначити, наскільки ефективно кожна з наведених мовних моделей може генерувати тестові випадки, а також які саме типи тестів вони найкраще підходять для створення. Після генерації та запуску тестів планується провести аналіз результатів та порівняння моделей між собою, з метою визначення найбільш ефективної з них для використання в подальшому тестуванні програмного забезпечення.

Також для виконання згенерованих автотестів було сформовано фреймворк для запуску тестів за допомогою мови програмування Java, тестових фреймворків TestNG та Selenium, з використанням патерну Page Object (див. рис 3.1).

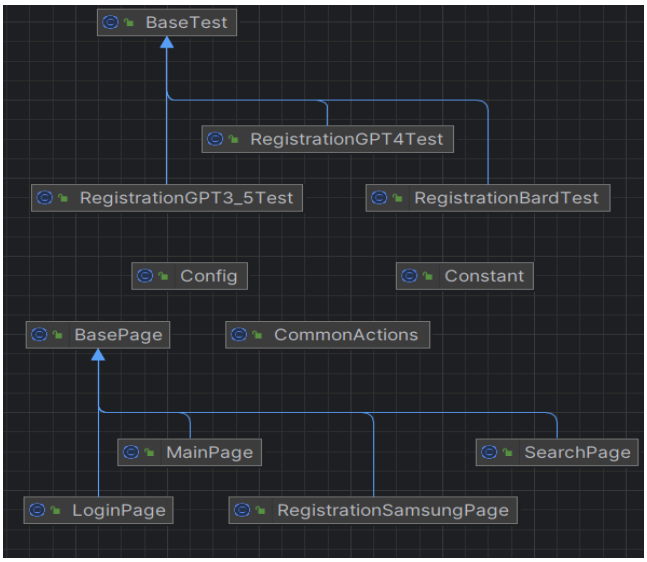

Рисунок 3.1 – UML діаграма проекту з використанням Page Object (рисунок виконано самостійно [24])

Проведення такого дослідження дозволить отримати об'єктивну оцінку можливостей кожної з мовних моделей і визначити їхню придатність для автоматизації тестування. Окрім того, результати дослідження будуть корисними для розробників програмного забезпечення та тестувальників, які шукають ефективні методи автоматизації тестування. Враховуючи широке застосування мовних моделей у сфері тестування, це дослідження може мати значний практичний вплив на розвиток автоматизованих процесів тестування програмного забезпечення.

3.2 Проведення експериментальних досліджень

Для експерименту з використанням LLM, таких як GPT-3.5, GPT-4 та Gemini, було використано 3 use cases (приклад див. рис 3.2) для реєстрації користувача на сайті samsungshop.com.ua. Use case слугував основою для тестового сценарію і для кожної моделі використовувався один і той же промпт, за результатами якого генерувався автоматизований тест. В промпті було

описано, що саме необхідно зробити: використовувати мову програмування Java, тестовий фреймворк Selenium, TestNG, та дотримуватись патерну Page Object. Також було запропоновано використовувати класи, в яких описана сторінка, а також методи і поля цих класів. У процесі виконання кожного промпту час формування промптів та їх кількість була додана в зведену таблицю. Після генерації автотесту, проводився запуск тесту в Intellij IDE, для перевірки працездібності тесту. У разі генерації невідповідного результату, промпт уточнювався і процес генерації повторювався, поки не був згенерований робочий автоматизований тест. Всього було використано 10 тестових сценаріїв, для яких було сформовано 10 автоматизованих тестів.

```
Use Case: Успішна реєстрація нового користувача

1. Передумови:
   - Користувач має доступ до Інтернету
   - Користувач відкрив веб-сторінку https://samsungshop.com.ua/auth/register/
   - Користувач має валідну електронну адресу та номер телефону

2. Основний сценарій:
   - Користувач вводить своє ім'я в поле "Ім'я"
   - Користувач вводить свій номер телефону в поле "Номер телефону"
   - Користувач вводить свою електронну адресу в поле "Електронна адреса"
   - Користувач вводить свій пароль в поле "Пароль"
   - Користувач натискає кнопку "Зареєструватись"
   - Система перевіряє введені дані
   - Система відображає повідомлення про успішну реєстрацію та відображає номер
     телефону на екрані

3. Альтернативний сценарій:
   - Користувач вводить некоректні дані (наприклад, неправильний формат електронної
     адреси або номеру телефону)
   - Система відображає повідомлення про помилку

4. Післяумови:
   - Користувач успішно зареєстрований на сайті
   - Користувач може використовувати свій обліковий запис для покупок та інших дій на сайті

5. Вимоги:
   - Поля для введення даних мають бути чітко видимими
   - Повідомлення про помилки мають бути зрозумілими та інформативними
   - Система має відображати номер телефону користувача після реєстрації

6. Обробка помилок:
   - Якщо користувач вводить некоректні дані, система повинна відображати відповідне
     повідомлення про помилку
   - Якщо користувач вводить вже зареєстровану електронну адресу, система повинна
     відображати повідомлення про помилку.
```

Рисунок 3.2 – Use Case для генерації автоматизованих тестів (рисунок виконано самостійно)

Данні генерації, такі як час виконання промпту, та кількість промптів занесені у таблицю (див. табл. 3.1). Усі три моделі проявили високу ефективність у створенні автоматизованих тестів для перевірки функціональності веб-сайту samsungshop.com.ua. Проте, найбільш точні та надійні результати були отримані від моделі Gemini (див. рис. 3.3), яка виявилася швидшою та більш надійною у порівнянні з GPT-4 та GPT-3.5 (див. рис. 3.4). Хоча кількість промптів не була найменшою, проте швидкість генерації тестів в цілому виявилась кращою (175,33 сек., див. табл. 3.1). Це може бути пов'язано з тим, що модель Gemini володіє покращеною здатністю до розуміння та генерації тексту, що дозволяє їй більш точно відтворювати необхідні промпти для створення автотестів.

Крім того, Gemini проявив себе ефективнішим у вирішенні завдань, пов'язаних з валідацією даних, таких як перевірка формату пошти та телефону, а також у відображенні помилок при невірному введенні даних.

Рисунок 3.3 – Діаграма кількості промптів для генерації 10 автотестів (рисунок виконано самостійно)

Інша модель GPT4, також проявила себе добре при генерації автоматичних тестів, але час на формування промптів доволі значний, хоча при додаткових уточнюючих промптах, час на генерацію відповіді значно зменшується. Однак загальної ефективності вона все ж виявила себе належним чином, забезпечуючи більшість успішних результатів. Таким чином, модель GPT-4 може бути використана для створення автоматизованих тестів, проте рекомендується уточнити та вдосконалити вхідні промпти для отримання більш точних результатів.

Таблиця 3.1 – Зведений результат генерації автоматизованих тестів з використанням LLM

Сценарій	Великі мовні моделі(LLM)								
	GPT4			GPT-3.5			Bard (Gemini)		
	промпти для створення автотесту, кількість	виконання запиту, сек.	Результат	промпти для створення автотесту, кількість	виконання запиту, сек.	Результат	промпти для створення автотесту, кількість	виконання запиту, сек.	Результат
1. Перевірка можливості успішної реєстрації нового користувача з правильною інформацією.	2	92.21	passed	6	89.25	passed	4	43.91	passed
2. Перевірка валідації порожніх полів воду	4	66.72	passed	5	72.33	passed	4	39.91	passed
3. Перевірка відображення помилки при введенні імені меньше 4 символів	1	9.61	faild	1	9.52	faild	2	15.29	faild
4.Перевірка відображення помилки при введенні не вірного формату пошти	1	9.38	passed	2	15.4	passed	2	11.26	passed
5. Перевірка відображення помилки при введенні не вірного формату телефону	1	8.57	passed	1	8.84	passed	2	10.98	passed
6. Перевірка відображення помилки при введені паролю меньше 8 символів	1	8.42	passed	2	20.36	passed	1	4.81	passed
7. Перевірка відображення помилки при введені паролю більше 30 символів	1	9.88	passed	1	9.67	passed	1	5.53	passed
8. Спроба регестрації користувача з вже зареєстрованою поштою	1	19.57	passed	2	26.54	passed	2	12.4	passed
9. Вхід в кабінет з валідними даними.	1	10.18	passed	1	9.68	passed	1	8.04	passed
10. Пошук товару по сайті	2	67.01	passed	3	34.99	passed	3	23.2	passed
Всього:	15	301.55		24	296.58		22	175.33	

Модель GPT3.5 показала посередньо і по деяким показникам мала відставання. Так наприклад загальна швидкість виконання промптів 296,58 секунд (див.табл. 3.1), проте кількість уточнюючих промптів виявилась більшою за всіх. Це впливає на загальний час створення автотесту, тому що для уточнення кожного нового промпту потрібен додатковий час.

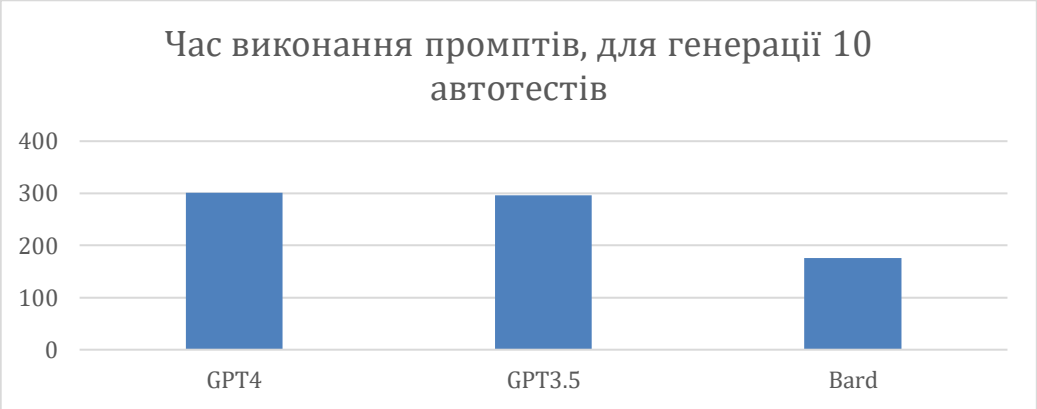

Рисунок 3.4 – Діаграма часу виконання промптів для генерації 10 автотестів

На основі згенерованих мовними моделями Gemini, GPT-4 та GPT-3.5 автотестів проведено серію експериментів, десять окремих запусків набору автоматичних тестів (10 тестів в наборі) для кожної моделі. Кожен набір складався з десяти тестових сценаріїв, що охоплювали визначені при генерації тестів аспекти функціональності програмного забезпечення. Результати кожного запуску були зафіксовані та виведені у вигляді алюр репортів (див. рис. 3.5), які містили інформацію про результат виконання кожного тесту, а також час, витрачений на його виконання. В цьому репорті також можна побачити результат не пройдених тесту. Невдале виконання цього тесту зумовлене знаходженням багу при введенні значно короткого імені при реєстрації користувача. Після цього результати були включені до загальної таблиці для подальшого аналізу та порівняння.

Цей підхід дозволив систематично дослідити та оцінити продуктивність кожної мовної моделі у контексті генерації автотестів, надаючи об'єктивні дані для подальшого вибору оптимального рішення у сфері тестування програмного забезпечення.

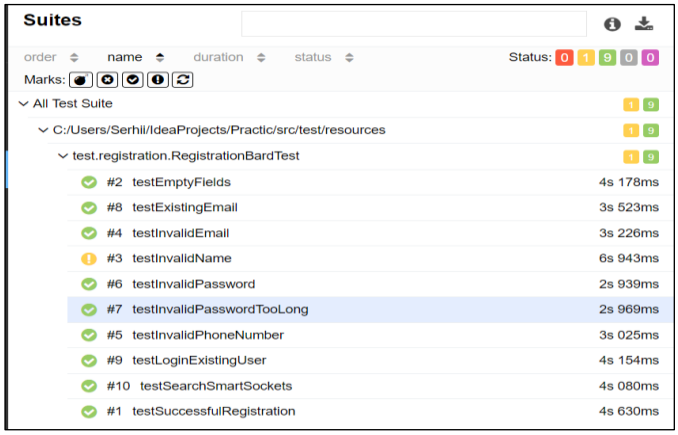

Рисунок 3.5 – Виведення результатів запуску набору автоматизованих тестів за допомогою Allure report (рисунок виконано самостійно)

Аналізуючи результати десяти запусків тестового набору, можна зробити висновок, що тести, згенеровані моделлю Gemini, проходили швидше в порівнянні з іншими моделями. Час виконання тестів, розроблених за допомогою Gemini, був найменшим серед усіх досліджуваних варіантів, що свідчить про його високу продуктивність у контексті автогенерації тестових сценаріїв. Це може вказувати на більшу ефективність та точність роботи моделі Gemini у порівнянні з GPT-4 та GPT-3.5 в умовах автоматизованого тестування програмного забезпечення. (див. рис. 3.6).

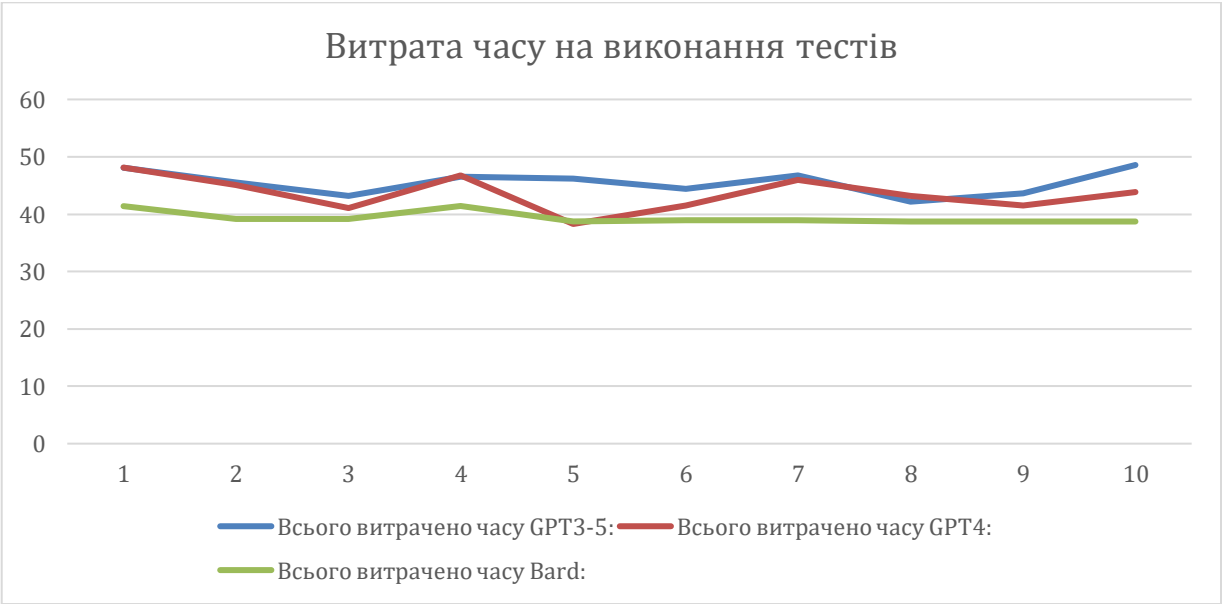

Рисунок 3.6 – Графік витраченого часу на кожен тестовий набір за 10 запусків (рисунок виконано самостійно)

Також після проведення 10 запусків тестового набору та аналізу результатів можна зробити висновок, що тести, розроблені за допомогою моделі Gemini, дозволяють виявити баги швидше в порівнянні з іншими методами(див. рис. 3.7).

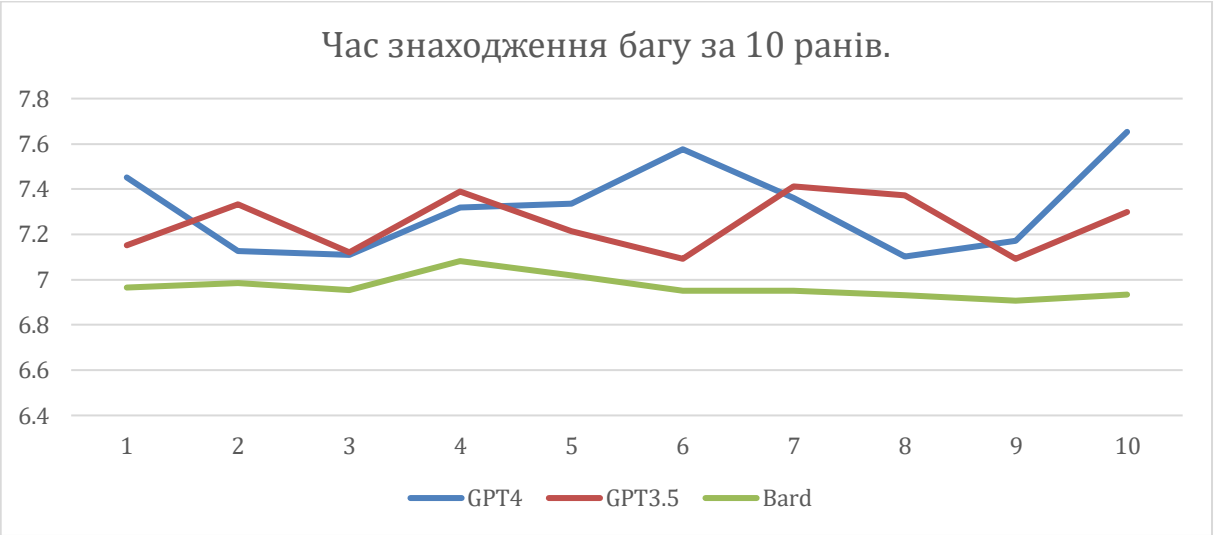

Рисунок 3.7 – Графік знаходження багу за 10 запусків тестового набору (рисунок виконано самостійно)

Отримані результати експерименту підтверджують, що використання LLM Gemini у процесі створення автоматизованих тестів для програмного забезпечення є важливим та обґрунтованим кроком. Ця модель демонструє високу ефективність у виявленні помилок та недоліків у програмному коді, а також забезпечує швидку та точну генерацію коду, згідно наданих вимог. Інші моделі показали трохи менші показники, так модель GPT-4 мала значний час генерації першого тесту, але після уточнень вимог, та генерацію подальших тестів було значно прискорено. Це свідчить що модель навчається на попередньо запропонованих даних. Модель GPT-3.5 має швидку генерацію відповіді, але для формування робочого автотесту треба надати більше уточнюючих даних. З урахуванням того, що час та якість тестування є критичними для успішного випуску програмного продукту на ринок, використання моделі Gemini може значно збільшити ефективність та продуктивність процесу розробки.

4 НОВІ МОДЕЛІ ГЕНЕРАТИВНИХ АЛГОРИТМІВ ТА ПРИКЛАДНЕ ВИКОРИСТАННЯ ДОСЛІДЖЕННЯ

4.1 Оновлені моделі після проведеного експерименту

Одна із провідних мовних моделей, яка доступна для широкого загалу користувачів та використовується багатьма у різноманітних сферах модель від Google – Gemini. Постійно оновлює свої функції та алгоритми для покращення відповідей, створених на основі промптів користувачів, що при вдалому використанні може значно підвищити ефективність роботи користувача да досягнути цілей, які користувач ставить перед собою.

Після проведення експерименту було презентовано оновлення Gemini та ChatGPT4, тому було прийнято рішення зробити огляд цих моделей, для розуміння можливостей використання їх у майбутньому. Розглянимо основні функції, які надали розробники та зробимо висновки про ефективність і подальший розвиток таких моделей, а також перспективу використання для автоматизації тестування програмного забезпечення.

Gemini від Google розширює доступ до ключових функцій моделі для більшої кількості мов і країн. Це оновлення надає людям у всьому світі можливість використовувати Gemini на своїх мобільних пристроях для написання текстів, планування, навчання та інших завдань.

В травні 2024р. відбулося розширення доступності мобільного додатка Gemini на нові мови та країни світу, що дозволить користувачам отримувати допомогу від Gemini навіть у дорозі. З Gemini на мобільному пристрої стає можливим використовувати текст, голос або зображення для отримання підтримки: наприклад, можна сфотографувати пробите колесо та отримати інструкції з ремонту або скористатися допомогою в написанні листа подяки. Це є важливим першим кроком у створенні справжнього помічника на основі штучного інтелекту – такого, що розмовляє, мультимодальний і корисний.

На пристроях Android для використання Gemini необхідно активувати Google Асистент на телефоні. Це забезпечує новий користувацький досвід, що

пропонує легкий доступ до Gemini, а також контекстну довідку прямо на екрані. В Україні ця функція наразі доступна англійською мовою для голосових запитів, однак українською можна вводити запити текстом.

Відбулося значне скорочення часу відповіді та покращення загальної продуктивності системи. Оптимізація обробки запитів призвела до більш плавного та швидкого функціонування моделі.

Додано нові функції генерації тексту, зокрема написання різних видів творчого контенту та розширення можливостей перекладу. Підвищено здатність моделі розуміти та відповідати на складні запитання та запити.

Також додано деякі додаткові вдосконалення. Покращено точність та надійність моделі. Розширено набір даних, на якому навчається модель, що призвело до глибшого розуміння світу та надання більш релевантних відповідей. Впроваджено низку інших покращень для підвищення загального користувацького досвіду. Gemini навчається на значно більшому та різноманітнішому наборі даних, що включає текст, код та інші види інформації. Цей набір даних ретельно відбирається та очищається, щоб гарантувати його високу якість.

Розробники Gemini впровадили нові та вдосконалені алгоритми машинного навчання, які дозволяють моделі краще розуміти та обробляти інформацію. Ці алгоритми більш стійкі до шуму та помилок у даних.

Код Gemini було оптимізовано для більш ефективного використання обчислювальних ресурсів. Це призвело до покращення швидкості роботи та загальної продуктивності моделі. Розробники Gemini постійно тестують та вдосконалюють модель, щоб гарантувати її точність та надійність. Вони використовують різні методи, такі як бенчмаркінг та аналіз помилок, щоб визначити та виправити будь-які проблеми.

Gemini також враховує відгуки користувачів, щоб покращити свою точність та надійність. Користувачі можуть повідомляти про будь-які проблеми, з якими вони стикаються, а розробники Gemini можуть використовувати цю інформацію для вдосконалення моделі.

Згідно описаних вище оновлень, удосконалень та оптимізації можна зробити висновок, що постійна підтримка та розвиток мовної моделі від Google може покращити використання у автоматизації тестування ПЗ, та збільшити ефективність в розробці тестового фреймворку та написанні тестових скриптів. Що в свою чергу призведе до підвищення ефективності тестування та зменшення витрат на розробку та впровадження готового продукту.

Також було презентовано нову версію Chat GPT-4o. Літера «o» або «omni» вказує на здатність моделі сприймати та відтворювати інформацію у форматах тексту, аудіо, фото, відео, а також їхніх комбінаціях. За даними OpenAI, взятих х відкритих джерел, GPT-4o працює вдвічі швидше, ніж GPT-4 Turbo — платна версія ШІ-бота компанії.

Оновлення акцентоване на функціях аналізу відео та аудіо. У режимі голосового чату GPT-4o здатен відповідати з швидкістю, еквівалентною швидкості людської відповіді. Модель також може розпізнавати емоції та відповідати з різними інтонаціями, включаючи спів. Додатково, штучний інтелект тепер можна перебивати. Використання голосового чату можна використовувати для навчання, повідомивши модель про тему, рівень складності навчання та уточнювати незрозумілі деталі з додатковими прикладами для роз’яснення.

Деякі функції доступні тільки на мобільних пристроях або на операційній системі MacOS, але в найближчому майбутньому також буде доступно і на Windows. Для користувачів платної версії, модель обіцяє бути в п’ять разів швидше та з доступом більших функцій.

Щодо використання моделі для генерації автоматизованих тестів для тестування ПЗ, то згідно опису удосконалень алгоритмів моделі та заявленою швидкістю відповіді, процес створення тестів має покращитись, порівняно з попередніми версіями моделі. Також якість тестів буде залежати від складності тесту, та детального опису поставленої задачі.

Постійне удосконалення та конкуренція серед компаній-розробників мовних моделей роблять продукт , який використовується в різних сферах життя,

різними людьми, що в майбутньому може покращити та полегшити життя, використовуючи ШІ в рутинних занять, навчанні, роботі, тощо.

4.2 Прикладне використання результатів експерименту та мовних моделей в реальних проектах

В останні роки штучний інтелект з генерацією (GenAI) привернув увагу та зацікавленість у сфері програмної інженерії. Інструменти GenAI, такі як GitHub Copilot і ChatGPT, були швидко досліджені в різних професійних контекстах.

Використання ШІ для автоматизованого тестування спрямоване на визначення оптимальних стратегій застосування штучного інтелекту для створення автоматизованих тестових сценаріїв, для подальшого використання в розробці тестового фреймворку[4].

Зі зростанням складності програмного забезпечення та підвищенням вимог до його якості, автоматизація тестування стала невід'ємною складовою процесу розробки. Впровадження мовних моделей та штучного інтелекту (ШІ) в інструменти автоматизації тестування програмного забезпечення відкриває нові горизонти для ефективності та точності тестування. Використання таких передових технологій, як GPT-3.5, GPT-4 та інші сучасні моделі, дозволяє значно скоротити час на написання тестів, зменшити людські помилки та підвищити рівень автоматизації.

Мовні моделі, зокрема, надають можливість створення автотестів на основі природної мови, що дозволяє тестувальникам та розробникам легше спілкуватися з інструментами тестування. Це сприяє більшій інтеграції процесів розробки та тестування, поліпшенню комунікації між командами та прискоренню циклів випуску. ШІ, в свою чергу, аналізує величезні обсяги даних, виявляє патерни та передбачає можливі помилки, що робить процес тестування більш проактивним та адаптивним.

Використання мовних моделей у поєднанні з ШІ також дозволяє створювати більш складні сценарії тестування, що охоплюють різноманітні випадки використання, та забезпечує більш ґрунтовне тестування додатків. Це

особливо важливо в умовах, коли час виходу на ринок є критичним фактором. Завдяки інноваціям у цій сфері, компанії можуть значно підвищити якість своїх продуктів, зменшити витрати на тестування та прискорити процес розробки.

Розглянемо можливість використання мовних моделей з інструментами, які використовуються в автоматизації програмного забезпечення. Confluence та Jira є двома потужними інструментами від Atlassian, які часто використовуються разом для забезпечення ефективної роботи команд, зокрема команд тестувальників. Обидва інструменти мають свої унікальні функції, але їх інтеграція дозволяє значно підвищити продуктивність і організованість роботи.

Confluence – це платформа для спільної роботи, що використовується для створення, організації та обміну документацією. Вона забезпечує створення вимог, редагування тестової документації, включаючи тестові плани, стратегії тестування, чек-листи, результати тестування тощо. Можливість спільного редагування документів, коментування та обговорення матеріалів у режимі реального часу. Також доступна зручна організація інформації у вигляді сторінок і просторів, які легко шукати і переходити до необхідних розділів та вимог.

Jira – це система управління проектами, що використовується для відстеження завдань, багів та інших питань, пов'язаних з розробкою і тестуванням ПЗ. Вона забезпечує управління завданнями, створення та відстеження завдань, багів, історій користувачів і епосів. Має досить розвинуті інструменти для створення звітів і дашбордів, що допомагають оцінити прогрес і ефективність команди. Також надає можливість налаштування автоматичних процесів, таких як призначення задач, відправка повідомлень тощо.

Інтеграція Confluence та Jira дозволяє забезпечити безперебійний обмін інформацією між інструментами, що є надзвичайно корисним для тестувальників. На цьому етапі ми можемо виділити потенційну взаємодію між цими двома інструментами та мовною моделлю. Основною метою цієї взаємодії буде виявлення вимог, які описані в Confluence та створення на основі них історій користувачів, тестових сценаріїв, тестових випадків, чек листів тощо.

Припустимо, що в Confluence створено документ з вимогами до нової функції (див. рис. 4.1).

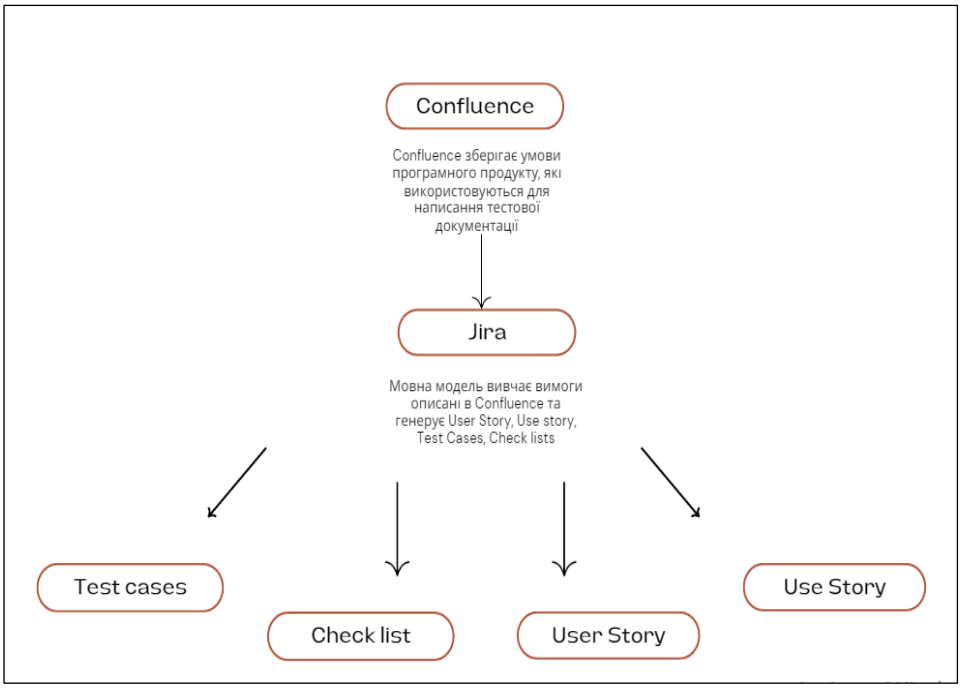

Рисунок 4.1 – Схема генерації документації за допомогою LMM (рисунок виконано самостійно)

Мовна модель аналізує цей документ і автоматично створює у Jira наступні елементи:

– історії користувачів, що описують функціональність з точки зору кінцевого користувача;
– тестові сценарії, що перевіряють відповідність реалізації вимогам;
– конкретні тестові випадки з детальним описом кроків, очікуваних результатів та необхідних вхідних даних;
– чек-листи для ручного тестування специфічних аспектів функції.

Це дозволяє значно скоротити час на створення тестової документації та забезпечити її узгодженість з вимогами. Крім того, автоматизоване створення та підтримка звітів у Confluence на основі даних з Jira дозволить команді тестування завжди бути в курсі поточного стану проекту та виявлених проблем.

Наступним етапом буде створення безпосередньо автоматизованих тестів на основі існуючих тестових артефактів, які вже будуть створені в Jira. Потрібно

буде розробити плагін, який підтримується середовищем розробки, наприклад Intellij IDE, та за допомогою цього плагіну буде відбуватися взаємодія мовної моделі з задокументованими вимогами в Confluence та Jira тікетах. Мовна модель, інтегрована в плагін, буде аналізувати вимоги описані в Confluence, відповідний Use Case в Jira, а також аналізувати тестовий автоматизований фреймворк, на основі якого будуть генеруватись автотести. Користувач має можливість керувати та вибирати які саме артефакти потрібно використовувати, а також додавати чи прибирати вимоги до майбутнього автотесту, додавати додаткові параметри, які класи, бібліотеки, методи і т.д використовувати.

На основі аналізу Use Case та вимог з Confluence, плагін генерує код автоматизованого тесту. Код включає всі необхідні імпорти, класи, методи та кроки для виконання тесту. Також створюються нові файли з тестами в проекті IntelliJ IDEA в відповідних директоріях. Всі згенеровані тести включають детальні коментарі, що пояснюють кожен крок тесту та очікувані результати. Користувач може переглянути згенеровані тести, внести необхідні корективи та запустити їх для перевірки коректності. Використовуючи можливості IntelliJ IDEA, тести можна запускати безпосередньо з IDE, а результати виконання будуть відображатися у зручному вигляді. У разі якщо згенерований результат не підійшов, або необхідно усунути недоліки, то існує можливісь виправити запит і згенерувати автоматизований тест з урахуванням недоліків.

Після успішного виконання тестів, результати можуть бути автоматично оновлені в Jira. Плагін може створити звіти про виконання тестів та прикріпити їх до відповідних задач у Jira. Алгоритм виконання запиту та генерації автоматизованого тесту відображено рисунку 4.2.

Концепт інтеграції мовних моделей та штучного інтелекту в процеси автоматизації тестування програмного забезпечення демонструє значний потенціал для підвищення ефективності та якості розробки. Застосування плагінів у середовищі розробки, таких як IntelliJ IDEA, для автоматичного створення тестів на основі Use Case з Jira та вимог з Confluence, дозволяє суттєво зменшити обсяг ручної роботи, знизити ризик помилок та прискорити процес тестування.

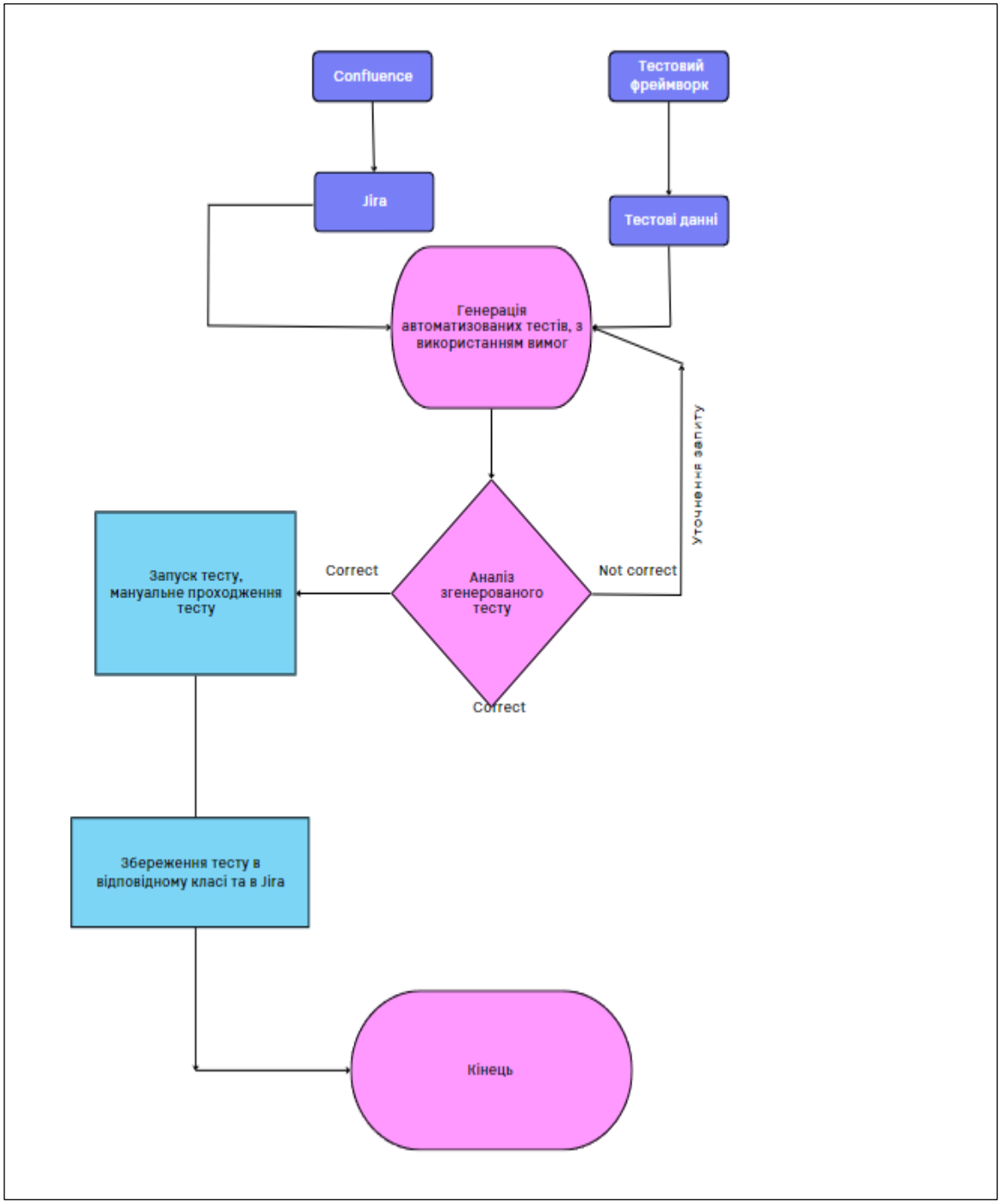

Рисунок 4.2 – Блок-схема генерації автоматизованого тесту (рисунок виконано самостійно)

Основна ідея полягає в інтеграції мовних моделей, таких як GPT-4, Gemini та інших з інструментами управління проектами та розробки. Плагін у IntelliJ IDEA, отримуючи номер задачі з Jira, автоматично витягує необхідну інформацію та передає її мовній моделі для аналізу. Модель, у свою чергу, генерує тестові

сценарії відповідно до вимог з Confluence. Таким чином, процес створення тестів стає значно більш автоматизованим та менш залежним від людського фактора.

Цей підхід забезпечує не лише швидкість і точність у створенні тестів, але й дозволяє стандартизувати процес тестування. Мовна модель автоматично визначає мову програмування та тестовий фреймворк, що забезпечує узгодженість тестів з існуючим кодовою базою проекту. Генерація коду автоматизованих тестів включає всі необхідні компоненти, такі як імпорти, класи, методи та кроки, що забезпечує високу якість та читабельність коду.

Використання такого плагіну також сприяє інтеграції результатів тестування з Jira, де результати можуть бути автоматично оновлені, створюючи звіти про виконання тестів та прикріплюючи їх до відповідних задач. Це дозволяє команді розробників та тестувальників мати постійний доступ до актуальної інформації про стан проекту, що підвищує прозорість та ефективність комунікації в команді.

Таким чином, інтеграція мовних моделей у процеси автоматизації тестування ПЗ є перспективним напрямом, що дозволяє значно підвищити ефективність, точність та стандартизацію процесів тестування. Це, у свою чергу, сприяє покращенню якості кінцевого продукту та скороченню часу розробки, що є важливими факторами у сучасних умовах високої конкуренції на ринку програмного забезпечення.

ВИСНОВКИ

Під час підготовки кваліфікаційної роботи були розглянуті актуальні питання застосування методів штучного інтелекту в автоматизованому тестуванні програмного забезпечення. У роботі детально досліджено різноманітні методи ШІ та, за допомогою задачі багатокритеріального вибору, було обрано метод машинного навчання для подальшого використання в експерименті, спрямованому на дослідження його ефективності у розробці автоматизованих тестів.

Важливим аспектом роботи стало аналіз підходів науковців та фахівців у галузі розробки програмного забезпечення до оцінки ролі методів ШІ у тестуванні ПЗ. Дослідження виявило, що методи штучного інтелекту можуть суттєво підвищити ефективність та точність процесів тестування, зменшуючи обсяг ручної праці та мінімізуючи ризик людських помилок.

Досягнення в галузі машинного навчання, зокрема використання нейронних мереж і алгоритмів глибокого навчання, показали перспективність інтеграції цих методів у автоматизовані системи тестування. Наприклад, методи машинного навчання можуть бути використані для генерації тестових випадків, ідентифікації аномалій в поведінці програмного забезпечення, а також для автоматичного виявлення та виправлення помилок.

На підставі проведеного аналізу предметної області, у контексті розробки автоматизованих тестів, особливу увагу було приділено мовним моделям, таким як GPT-3.5, GPT-4, Gemini, які здатні генерувати складні сценарії тестування на основі вимог та специфікацій, зберігаючи високу якість та читабельність коду. Застосування таких моделей дозволяє забезпечити стандартизованість тестових випадків, підвищити прозорість процесів тестування та полегшити комунікацію між розробниками та інженерами з тестування ПЗ.

У ході експерименту було запропоновано дослідити доцільність використання мовних моделей для генерації автоматизованих тестів для тестування веб застосунків. Для кожної моделі було сформовано однакові тестові

сценарії, на основі яких кожна з моделей формувала автоматизовані тести, з подальшим записом результатів генерації у таблицю, а на основі згенерованих тестів створювався тестовий набір. Після цього було проведено запуск тестів і збір метрик для порівняння результатів тестів, які заносились в зведену таблицю. Під час проведеного експерименту було виявлено що, мовна модель на основі Gemini показала себе краще за кількома показниками, такими як швидкість генерації тестів, точність генерації тестів та інші.

Для дослідження використовувались наступні мовні моделі: GPT-3.5, GPT-4, і модель на основі Gemini. Кожній моделі було надано однакові вхідні дані, що включали тестові сценарії для різних аспектів веб-застосунку. Після генерації тестів кожною моделлю, тести були запущені на аналогічному тестовому середовищі для забезпечення справедливості експерименту.

Отримані результати вказують на значний потенціал застосування мовних моделей для автоматизації тестування. Модель Gemini, завдяки своїй високій швидкості та точності, може стати ключовим інструментом для розробників, дозволяючи скоротити час на створення тестів і підвищити загальну якість програмного забезпечення. Висока стабільність виконання тестів підтверджує надійність цієї моделі, що особливо важливо в умовах постійних змін і оновлень у веб-застосунках.

Перспективи подальших досліджень включають аналіз впливу різних типів вхідних даних на ефективність мовних моделей, а також розширення спектру застосувань моделей, таких як Gemini, на інші області програмного забезпечення. Додатково, вивчення можливостей інтеграції мовних моделей з існуючими системами безперервної інтеграції та доставки (CI/CD) може відкрити нові горизонти для автоматизації процесів тестування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Люта М. В., Розломій І. О., Новикова К. В. Аналіз методів тестування програмного забезпечення : thesis. 2017. URL: https://er.knutd.edu.ua/handle/123456789/8421 (дата останнього звернення: 14.02.2024).

2. Лановий О. Ф. Візуалізація в методах тестування програмного забезпечення: thesis. 2017. URL: http://openarchive.nure.ua/handle/ document/9432 (дата звернення: 02.03.2024).

3. Фундукян А. Метод розстановки пріоритетів тестів. education and science of today: intersectoral issues and development of sciences / chair О. Лановий. 2021. URL: https://doi.org/10.36074/logos-19.03.2021.v2.31 (дата звернення: 29.02.2024).

4. Ларченко С. О. Застосування ШІ в автоматизації тестування програмного забезпечення // 28-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». Зб. матеріалів форуму. Т. 6., / редкол.: Федорович ОЄ, Субботін СА та інш. – Харків: ХНУРЕ. 2024. – 958 с. URL: https://doi.org/10.30837/IYF.IIS.2024.565 (дата останнього звернення: 15.05.2024).

5. ISTQB. (2021). ISTQB Glossary of Terms. International Software Testing Qualifications Board. https://glossary.istqb.org/en_US/term/testing-9 (дата останнього звернення: 18.02.2024).

6. Слабоспицька О.О. «Формальні методи побудови програм: тестування та оцінка надійності» URL: https://studfile.net/preview/9772980/page:3/ 2019, (дата останнього звернення 12.02.2023).

7. Khankhoje R. AI in Test Automation: Overcoming Challenges, Embracing Imperatives. International Journal on Soft Computing, Artificial Intelligence and Applications. 2024. Vol. 13, no. 1. P. 01–10. URL: https://doi.org/10.5121/ijscai.2024.13101 (дата останнього звернення: 14.02.2024).

8. Дж. Маккарті, "Програми зі здоровим глуздом", Матеріали симпозіуму з механізації процесів мислення, т. 1. Лондон: Канцелярія Її Величності, 1958, с. 77-

84.

9. Prasanna Kumar P. Unleashing the Power of Large Language Models: Revolutionizing Text Understanding and Generation, AI Mind, 2023. URL: https://pub.aimind.so/unleashing-the-power-of-large-language-models-revolutionizing-text-understanding-and-generation-65db251e6ff9 (дата останнього звернення 14.02.2024).

10. Explainable Artificial Intelligence to Advance Structural Health Monitoring / D. Luckey et al. Structural Integrity. Cham, 2021. P. 331–346. URL: https://doi.org/10.1007/978-3-030-81716-9_16 (дата останнього звернення: 20.02.2024).

11. РОЗПОРЯДЖЕННЯ від 02 грудня 2020 р. № 1556-р Про схвалення Концепції розвитку штучного інтелекту в Україні URL: https://www.kmu.gov.ua/npas/pro-shvalennya-koncepciyi-rozvitku-shtuchnogo-intelektu-v-ukrayini-s21220 (дата останнього звернення 2.12.2023).

12. An Automated Approach for Diagnosing Allergic Contact Dermatitis Using Deep Learning to Support Democratization of Patch Testing / M. R. Hall et al. Mayo Clinic Proceedings: Digital Health. 2024. Vol. 2, no. 1. P. 131–138. URL: https://doi.org/10.1016/j.mcpdig.2024.01.006 (дата останнього звернення: 8.03.2024).

13. Russell S. J., Norvig P. Lecture PowerPoints for Artificial Intelligence: A Modern Approach. Pearson Education, Limited, 2020.

14. SALOKY, Tomáš та ŠEMINSKÝ, Yaroslav. Штучний інтелект та машинне навчання. 2005. URL: http://conf.uni-obuda.hu/SAMI2005/SALOKY.pdf. (дата звернення: 28.03.2024).

15. Predicting Open Education Competency Level: A Machine Learning Approach / G. Ibarra-Vazquez et al. Heliyon. 2023. P. e20597. URL: https://doi.org/10.1016/j.heliyon.2023.e20597 (дата останнього звернення: 22.03.2024).

16. Сфери застосування штучного інтелекту URL: https://aiconference.com.ua/uk/news/oblasti-primeneniya-iskusstvennogo-intellekta-92253 (дата останнього звернення 5.4.2024).

17. Оптимізації процесу тестування на проекті URL: https://careers.epam.ua/blog/qa-qc-testing (дата останнього звернення: 28.02.2023).

18. Spaniol M. J., Rowland N. J. AI-assisted scenario generation for strategic planning. FUTURES & FORESIGHT SCIENCE. 2023. URL: https://doi.org/10.1002/ffo2.148 (дата останнього звернення: 01.04.2024).

19. Hayatou Oumarou, El Mansour F. Automatic Generation of Unit Test Data for Dynamically Typed Languages. Indonesian Journal of Computer Science. 2023. Vol. 12, no. 5. URL: https://doi.org/10.33022/ijcs.v12i5.3396 (дата останнього звернення: 05.04.2024).

20. Kumar G., Chopra V. Automatic Test Data Generation for Basis Path Testing. Indian Journal Of Science And Technology. 2022. Vol. 15, no. 41. P. 2151–2161. URL: https://doi.org/10.17485/ijst/v15i41.1503 (дата останнього звернення: 05.04.2024).

21. Mani Padmanabhan E. a. Topological Data Analysis for Software Test Cases Generation. International Journal on Recent and Innovation Trends in Computing and Communication. 2023. Vol. 11, no. 9. P. 2046–2053. URL: https://doi.org/10.17762/ijritcc.v11i9.9203 (дата останнього звернення: 15.04.2024).

22. Mohammed K. AI in cloud computing: Exploring how cloud providers can leverage AI to optimize resource allocation, improve scalability, and offer AI-as-a-service solutions. Advances in Engineering Innovation. 2023. Vol. 3, no. 1. P. 22–26. URL: https://doi.org/10.54254/2977-3903/3/2023035 (дата останнього звернення: 15.04.2024).

23. Дідковська М.В., Тимошенко Ю.О, Тестування: Основні визначення, аксіоми та принципи, 2010. 61с. URL: http://mmsa.kpi.ua/sites/default/files/disciplines/%D0%A0%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0%20%D1%96%20%D1%82%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC/didkovska_m_v_testing_definition_part1.pdf (дата останнього звернення: 11.04.2024).

24. Посилання на код, використаний в роботі, знаходиться за адресою: https://github.com/SergLach/Practic