ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Методика оцінювання якості та оптимізації інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Сергій САЗОНОВ
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-62
Сергій САЗОНОВ

Керівник: Владислав ЯСКЕВИЧ
канд. техн. наук, доц.

Рецензент: Ольга ПОЛОНЕВИЧ
канд. техн. наук, доц.

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сазонову Сергію Олеговичу

1. Тема кваліфікаційної роботи: «Методика оцінювання якості та оптимізації інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації»

керівник кваліфікаційної роботи Владислав ЯСКЕВИЧ, канд. техн. наук, доцент, доцент кафедри ІПЗ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467.

2. Строк подання кваліфікаційної роботи «19» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, параметри систем для автоматизації тестування, методи оцінювання якості та оптимізації використання згенерованої тестової документації, вимоги до точності та релевантності автоматично згенерованої документації.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації.
2. Дослідження методів оцінювання якості та оптимізації інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації.

3. Розробка методики оцінювання якості та оптимізації інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації.
4. Проведення експериментальних досліджень та аналіз отриманих результатів.
5. Перелік ілюстративного матеріалу: *презентація*
    1. Алгоритм оцінювання якості згенерованої документації.
    2. Кількісні метрики.
    3. Якісні метрики.
    4. Нормалізація отриманих даних та середні показники метрик.
    5. Формула гібридного індексу якості.
    6. Екрана форма програми для автоматичного рахунку hqi.
    7. Експериментальні дослідження.
    8. Порівняльна діаграма результатів роботи інтелектуальних систем
6. Дата видачі завдання «31» жовтня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	31.10 - 04.11.2025	
2	Аналіз існуючих інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації.	04.11 - 07.11.2025	
3	Дослідження методів оцінювання якості та оптимізації інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації.	07.11 - 12.11.2025	
4	Розробка комплексної методики оцінювання якості та оптимізації генерації тестової документації.	12.11 - 23.11.2025	
5	Розробка програмного модуля для автоматичного підрахунку гібридного індексу якості.	23.11 - 24.11.2025	
6	Проведення експериментів з генерації тестової документації та аналіз отриманих результатів	24.11 - 26.11.2025	
7	Оформлення роботи: вступ, висновки, реферат	26.11. - 05.12.2025	
8	Розробка демонстраційних матеріалів	05.12 - 19.12.2025	

Здобувач вищої освіти _____ Сергій САЗОНОВ
(підпис)

Керівник
кваліфікаційної роботи _____ Владислав ЯСКЕВИЧ
(підпис)

РЕФЕРАТ

.

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 87 стор., 7 табл., 6 рис., 34 джерела.

Мета роботи – покращення процесу оцінювання якості та оптимізації використання інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації.

Об'єкт дослідження – процес оцінювання якості та оптимізації використання інтелектуальних систем для автоматизації тестування програмного забезпечення та генерації супровідної тестової документації.

Предмет дослідження – метод оцінювання якості та оптимізації використання інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації.

Короткий зміст роботи: У роботі основну увагу зосереджено на розробці комплексної методики оцінювання якості та оптимізації використання інтелектуальних систем для автоматизованої генерації тестової документації. Для формування цієї методики проаналізовано сучасні інтелектуальні інструменти, їхні можливості та технологічні обмеження в контексті створення тестових артефактів. Визначено релевантні кількісні та якісні метрики, що забезпечують об'єктивну оцінку точності, повноти й релевантності результатів, а також слугують основою для підвищення надійності вихідних даних. Підготовлено структуровані набори промтів і сценаріїв для практичної перевірки методики. Проведене експериментальне порівняння Google Gemini, Microsoft Copilot та OpenAI ChatGPT підтвердило її ефективність і дозволило сформувати рекомендації щодо подальшої оптимізації процесу автоматизованої генерації тест-кейсів.

КЛЮЧОВІ СЛОВА: МЕТОДИКА ОЦІНЮВАННЯ, ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ, ГЕНЕРАЦІЯ СУПРОВІДНОЇ ТЕСТОВОЇ ДОКУМЕНТАЦІЇ, АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ, МЕТРИКИ ЯКОСТІ.

ABSTRACT

Text part of the master's qualification work: 87 pages, 7 pictures, 6 tables, 34 sources.

The purpose of the work is to improve the process of quality assessment and optimisation of the use of intelligent systems for automating software testing processes and generating accompanying test documentation.

The object of research is the process of quality assessment and optimisation of the use of intelligent systems for automating software testing and generating accompanying test documentation.

The subject of research is the method of quality assessment and optimisation of the use of intelligent systems for automating software testing processes and generating accompanying test documentation.

Summary of the work: The work focuses on the development of a comprehensive methodology for assessing the quality and optimising the use of intelligent systems for the automated generation of test documentation. To develop this methodology, modern intelligent tools, their capabilities and technological limitations in the context of creating test artefacts were analysed. Relevant quantitative and qualitative metrics were identified to ensure an objective assessment of the accuracy, completeness, and relevance of the results, as well as to serve as a basis for improving the reliability of the output data. Structured sets of prompts and scenarios were prepared for practical testing of the methodology. An experimental comparison of Google Gemini, Microsoft Copilot, and OpenAI ChatGPT confirmed its effectiveness and allowed recommendations to be made for further optimisation of the automated test case generation process.

KEYWORDS:EVALUATION METHODOLOGY, INTELLIGENT SYSTEMS, GENERATION OF ACCOMPANYING TEST DOCUMENTATION, TESTING AUTOMATION, QUALITY METRICS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ШІ – штучний інтелект

QA – Quality Assurance

ПЗ – програмне забезпечення

API – Application Programming Interface

BDD – Behavior-Driven Development

LQA – Linguistic Quality Assessment

LLM – Large Language Model

CI/CD – Continuous Integration / Continuous Deployment

HITL – Human In The Loop

ВММ – велика мовна модель

ВСТУП

Інтелектуальні системи на основі штучного інтелекту є одним з найбільш перспективних напрямів розвитку інформаційних технологій, і його застосування в розробці програмного забезпечення набуває дедалі більшої популярності. Інтеграція інтелектуальним системам в життєвий цикл розробки програмного забезпечення здатна значно підвищити ефективність, знизити витрати часу та покращити якість кінцевих продуктів.[1] Завдяки своїм можливостям машинного навчання та аналізу великих даних, штучний інтелект дозволяє автоматизувати рутинні та трудомісткі завдання, що раніше виконувалися вручну. Це відкриває нові можливості для підвищення ефективності та точності тестування, а також дозволяє виявляти приховані дефекти, які можуть бути пропущені при традиційному підході. Інтелектуальні системи здатні автоматизувати такі процеси, як генерація тестових даних, виконання регресійних тестів, аналіз логів та створення звітів. Це дозволяє звільнити QA-інженерів від рутинної роботи та зосередитися на більш складних завданнях, що вимагають творчого підходу. Завдяки здатності обробляти великі обсяги даних і виконувати тести паралельно, інтелектуальні системи дозволяють значно скоротити час на тестування та виведення продукту на ринок. Алгоритми машинного навчання здатні виявляти тонкі відхилення від очікуваної поведінки системи, які можуть бути пропущені при ручному тестуванні. Це дозволяє значно підвищити якість програмного забезпечення. Інтелектуальні системи може виявляти нові типи дефектів, які раніше були невідомими або важко виявлялися традиційними методами, а також адаптувати тест-кейси до конкретних користувачів та їхніх поведінкових моделей, що підвищує релевантність тестування.

Однак, важливо розуміти, що ШІ є лише інструментом, який доповнює роботу QA-інженерів, але не може повністю замінити їхню експертизу. Фахівці з тестування все ще необхідні для розробки тестових стратегій, аналізу

результатів тестів, прийняття рішень щодо випуску продукту та забезпечення загальної якості програмного забезпечення.

Об’єкт дослідження – процес оцінювання якості та оптимізації використання інтелектуальних систем для автоматизації тестування програмного забезпечення та генерації супровідної тестової документації.

Предмет дослідження – метод оцінювання якості та оптимізації використання інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації.

Мета роботи – покращення процесу оцінювання якості та оптимізації використання інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації.

Для досягнення поставленої мети в роботі передбачено вирішення наступних завдань:

1. Проаналізувати існуючі інтелектуальні системи для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації.
2. Проаналізувати сучасні методи оцінки якості та оптимізації використання інтелектуальних систем для генерації супровідної тестової документації, визначити їхні ключові можливості та недоліки.
3. Визначити кількісні та якісні метрики оцінки якості згенерованої документації, сформувати комплексну методику оцінювання якості та оптимізації використання інтелектуальних систем для процесів тестування та генерації тестової документації, яка поєднує кількісні та якісні метрики.
4. Підготувати вхідні дані для генеративних мовних моделей, для проведення експериментального дослідження з генерації функціональних тест кейсів.
5. Провести експериментальне дослідження та порівняльний аналіз генерації функціональних тест-кейсів з використанням генеративних мовних моделей Google Gemini, Microsoft Copilot та OpenAI ChatGPT з метою

подальшої оптимізації генерування.

Практична значущість результатів – розроблена методика може бути використана для створення та вдосконалення інтелектуальних систем, що автоматизують процеси тестування програмного забезпечення та генерування супровідної тестової документації. Запропонований підхід підвищує точність, повноту й релевантність згенерованих тестових артефактів, зменшуючи залежність команд розробки від ручної підготовки тестів. Це сприяє оптимізації ресурсів, прискоренню циклів розробки та підвищенню надійності програмних продуктів. Методика забезпечує можливість адаптації інтелектуальних систем до різних проєктних умов та вимог. Вона може бути впроваджена у командах будь-якого масштабу й особливо корисна в організаціях, що прагнуть підвищити рівень автоматизації та якість програмного забезпечення.

Робота пройшла апробацію на науково-практичних конференціях:

1. VI Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT»
2. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ»

За результатами апробації опубліковано тези доповідей [1] та [2].

1. АНАЛІЗ ІСНУЮЧИХ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ ДЛЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ ТА ГЕНЕРАЦІЇ ДОКУМЕНТАЦІЇ

1.1 Сучасні підходи та інструменти на основі ШІ

Інтеграція технологій штучного інтелекту дозволяє значно підвищити ефективність та якість тестування. У дослідженні Романа Витвицького та Володимира Якубовського, представленому в журналі Хмельницького національного університету, аналізується адаптація технологій штучного інтелекту та машинного навчання в українських ІТ-компаніях. Автори наголошують на перевагах автоматизації, таких як зменшення витрат, прискорення релізів і покращення точності тестування. Вони також виділяють проблеми, пов’язані з недостатнім рівнем підготовки кадрів, відсутністю інституційної підтримки та слабким впровадженням новітніх технологій у національний виробничий процес [3].

Автор Ф. Гуральник у своїй статті пропонує концептуальну модель використання ШІ в автоматизованому тестуванні. Він розглядає роль ШІ в побудові сценаріїв тестування, прогнозуванні дефектів, генерації тестових даних та оптимізації процесу тестування. Особлива увага приділяється поєднанню DevOps-практик із інтелектуальними технологіями, що забезпечує гнучкість і динамічність процесу розробки ПЗ. Однак автор також зазначає ризики впровадження ШІ, такі як відсутність стандартизації та залежність від даних. Стаття також звертає увагу на ризики та проблеми, пов’язані з якістю, безпекою, відповідністю нормативним вимогам та інфраструктурою тестування, підкреслюючи необхідність адаптації до швидко змінюваного середовища розробки програмного забезпечення [4].

У роботі Романа Онищенка, Наталії Котенко та Тетяни Жирової зосереджено увагу на ефективності застосування ШІ в тестуванні веб-застосунків. Автори аналізують, як хмарні сервіси, DevOps та

ML-фреймворки можуть забезпечити масштабованість і швидкість перевірки складних ІТ-систем. В статті аналізуються інструменти, які поєднують хмарні сервіси та елементи машинного навчання, що дає змогу забезпечити глибший рівень перевірки систем із великою кількістю вхідних змінних. Дослідження доводить, що автоматизоване тестування з використанням ШІ дає змогу виявляти логічні помилки, які важко знайти за допомогою традиційних методів, а також скорочує загальний час на тестування. Автори підкреслюють важливість хмарних технологій, DevOps та інтеграції ШІ для підвищення надійності ПЗ [5].

Стаття І. Гунько, опублікована у журналі "Вісник Київського інституту бізнесу та технологій", аналізує сучасні тенденції в тестуванні ПЗ, зокрема впровадження ШІ та машинного навчання в автоматизацію тестування. Автор розглядає застосування цих технологій у генерації тестових сценаріїв, прогнозуванні дефектів та оптимізації процесу тестування. Окрім ШІ, вона охоплює такі теми, як тестування безпеки, блокчейн для аудиту процесів, та виклики, пов'язані з регуляторними нормами й етичними питаннями в галузі. Авторка пропонує стратегічний підхід до впровадження ШІ, що враховує не лише технічні, а й управлінські та соціальні аспекти змін у процесах розробки програмного забезпечення [6].

У статті Олександра Муляревича та Юлії Боярінової, опублікованій в Ukrainian Journal of Computing Innovations, аналізується ефективність автоматизованого та ручного тестування в забезпеченні якості ПЗ. Автори досліджують різні аспекти тестування, такі як вартість, час виконання, покриття тестами та виявлення дефектів, порівнюючи результати для обох підходів. Вони також розглядають фактори, які впливають на вибір між автоматизованим та ручним тестуванням, зокрема складність системи, частота змін та вимоги до якості. Стаття надає рекомендації щодо оптимального використання кожного підходу в залежності від умов проекту та специфіки тестових завдань. Автори встановлюють, що автоматизоване тестування є ефективним для стабільних і

часто повторюваних функцій, таких як регресійне тестування та перевірка продуктивності, завдяки можливості виконання великої кількості тестів у короткі строки. Водночас виявлено, що ручне тестування залишається незамінним для оцінки користувацького інтерфейсу, аналізу нових функцій та складних сценаріїв, які потребують гнучкості та детального аналізу. Автори пропонують комбінований підхід, який поєднує переваги обох методів, для досягнення високої якості програмного забезпечення [7].

Інтелектуальні системи на основі штучного інтелекту використовуються в тестуванні програмного забезпечення за допомогою різноманітних методологій та технік, спрямованих на підвищення ефективності та якості процесу тестування.

Прогнозне вибіркове тестування: використовує ШІ та машинне навчання для визначення найбільш важливих тестів, які слід запустити на основі змін коду та історичних даних.

Автоматизоване генерування тестів: автоматично створюються на основі вимог, історій користувачів або існуючого коду.

Самолікування тестів: інструменти можуть автоматично коригувати тест-скрипти при зміні елементів інтерфейсу користувача, зменшуючи необхідність ручного втручання.

Візуальне тестування: застосовує комп'ютерний зір для виявлення візуальних регресій та невідповідностей в інтерфейсі користувача.

Прогнозування дефектів: аналізуючи код та історичні дані, ШІ визначає потенційних місць виникнення помилок.

Інтелектуальне виконання та оптимізація тестів: включає використання ШІ для пріоритизації та планування виконання тестів.

Обробка природної мови (NLP): застосовується для розуміння та обробки тест-кейсів або вимог, написаних природною мовою.

Ринок пропонує широкий спектр інструментів та фреймворків для тестування програмного забезпечення на основі штучного інтелекту, які можна

класифікувати за їх основними функціональними можливостями [8,9].

1.1.1 AI-driven платформи для UI-тестування

Mabl – це платформа для тестування програмного забезпечення, яка використовує штучний інтелект для автоматизації процесів тестування. Вона дозволяє створювати, запускати та підтримувати тести для веб, мобільних і API додатків з мінімальними зусиллями. Mabl використовує технології машинного навчання для автоматичного створення та підтримки тестів. Інструмент аналізує користувацьку поведінку та автоматично генерує тестові сценарії, які потім можуть бути налаштовані та розширені [10].

Переваги:

-Mabl дозволяє значно прискорити процес створення та виконання тестів.

-Інтуїтивний інтерфейс і мінімальна кількість коду роблять Mabl доступним для широкого кола користувачів.

-Mabl легко масштабується для великих проектів і може підтримувати тисячі тестів.

-Платформа надає детальну аналітику результатів тестів, що дозволяє виявляти проблеми на ранніх етапах.

Недоліки:

-В деяких випадках може вимагатися ручне втручання для вирішення складних проблем.

-Ліцензування Mabl може бути досить дорогим для невеликих команд або проектів з обмеженим бюджетом.

-Хоча Mabl пропонує широкий спектр функцій, можливості кастомізації можуть бути обмежені порівняно з іншими інструментами тестування.

Приклади використання:

-Прискорення процесу тестування і зменшення витрат на нього.

-Мають динамічні веб-додатки, які часто змінюються.

-Хочуть підвищити якість програмного забезпечення за допомогою автоматизації.

-Не мають великого досвіду в написанні тестів.

Applitools Eyes – це інструмент для візуального тестування, який використовує технологію штучного інтелекту для порівняння зображень. Він дозволяє автоматично виявляти візуальні відмінності між різними версіями веб-додатків, що є особливо корисним при розробці інтерфейсів користувача [11]. Під час виконання тесту інструмент робить знімки екрану різних елементів веб-сторінки. Знімки порівнюються з базовими зображеннями, які були зроблені раніше і вважаються правильними. Штучний інтелект аналізує зображення і виявляє навіть незначні відмінності в макетах, кольорах, шрифтах та інших візуальних елементах. Інструмент генерує детальні звіти про виявлені відмінності, включаючи зображення з виділеними відмінностями.

Переваги:

-Автоматизація процесу порівняння зображень дозволяє значно прискорити тестування.

-Applitools Eyes може порівнювати цілі веб-сторінки або окремі елементи, що забезпечує повне покриття візуальних аспектів тестування.

-Інструмент легко інтегрується з іншими інструментами для автоматизації тестування, такими як Selenium.

-Підтримка різних браузерів і платформ: Applitools Eyes підтримує широкий спектр браузерів і операційних систем.

Недоліки:

-Комерційна ліцензія може бути дорогою для невеликих проектів.

-Для складних веб-додатків може знадобитися додаткове налаштування інструменту.

-Інструмент може генерувати помилкові спрацювання при незначних змінах стилю, таких як зміна відтінку кольору.

-Якість результатів тестування залежить від якості базових зображень.

Приклади використання:

-Перевірка того, як веб-додаток відображається на різних пристроях і роздільних здатностях екрану.

-Перевірка того, що перекладені версії веб-додатку візуально відповідають оригінальній версії.

-Перевірка того, що зміни в коді не призвели до несподіваних візуальних змін.

1.1.2 Інструменти для генерації модульних та API-тестів

Diffblue Cover – це інструмент для автоматичного створення unit-тестів для Java-коду, який використовує технології штучного інтелекту та аналізу байткоду. Він дозволяє розробникам значно прискорити процес покриття коду тестами без ручного написання тестових сценаріїв. Diffblue автоматично аналізує існуючий код, визначає його логіку, шляхи виконання та створює відповідні JUnit-тести, що відображають реальну поведінку програми [12].

Під час роботи інструмент виконує статичний та динамічний аналіз коду, визначає можливі варіанти вхідних даних та генерує тестові методи, які охоплюють найбільш значущі сценарії. Згенеровані тести можна одразу запускати у проєкті або редагувати вручну. Diffblue також виділяє потенційні збої та складні ділянки коду, де необхідне додаткове покриття.

Переваги:

-Автоматична генерація тестів дозволяє у рази скоротити час створення unit-тестів.

-Інструмент часто досягає високого рівня code coverage із мінімальними зусиллями.

-Швидке написання тестів допомагає підтримувати якість коду під час рефакторингу.

-Підтримка Maven, Gradle, Jenkins, GitHub Actions та інших інструментів CI/CD.

-При зміні бізнес-логіки Diffblue може оновлювати тести відповідно до нового коду.

Недоліки:

-Підтримується лише Java, що робить інструмент менш універсальним.

-Комерційна ліцензія може бути дорогою для маленьких команд.

-Згенеровані тести можуть бути складними для читання.

-Diffblue створює тести для наявного коду, але не покращує його структуру.

Приклади використання:

-Автоматичне створення unit-тестів для великого легасі-коду.

-Підготовка тестового покриття перед масштабним рефакторингом.

-Виявлення непередбачуваної поведінки під час розробки нових функцій.

-Розвантаження команди розробки від рутинного написання типових тестів.

Parasoft – це комплексний набір інструментів для автоматизації тестування, статичного аналізу та контролю якості програмного забезпечення. Платформа охоплює різні види тестування, включаючи unit-тестування,

API-тестування, тестування безпеки, сервісну віртуалізацію та аналіз коду відповідно до стандартів (MISRA, CERT тощо) [13].

Під час роботи Parasoft автоматично аналізує код на відповідність стандартам та можливим дефектам, генерує тестові сценарії для API, перевіряє поведінку систем у різних середовищах та створює докладні звіти. Сервісна віртуалізація дозволяє імітувати поведінку зовнішніх сервісів, що робить тестування стабільнішим і менш залежним від інтеграцій.

Переваги:

-Підтримка великої кількості мов програмування та типів тестування.

-Виявлення помилок на ранніх етапах розробки.

-Можливість тестувати навіть при недоступних або нестабільних зовнішніх сервісах.

-Підтримка CI/CD, різних IDE та репозиторіїв.

-Перевірка відповідності коду галузевим стандартам безпеки та якості.

Недоліки:

-Великий функціонал може вимагати значного часу на впровадження.

-Продукт орієнтований на підприємства, що може бути занадто дорогим для малих команд.

-Для ефективного використання необхідні спеціальні знання та навчання.

-Деякі модулі можуть вимагати значної обчислювальної потужності.

Приклади використання:

-Тестування складних корпоративних систем із великою кількістю інтеграцій.

-Статичний аналіз коду на відповідність стандартам безпеки.

-Автоматизація API-тестування мікросервісної архітектури.

-Створення віртуальних сервісів для моделювання поведінки недоступних компонентів.

1.1.3 Застосування великих мовних моделей для генерації тест-кейсів

Застосування великих мовних моделей, таких як OpenAI GPT, Google Gemini та Microsoft Copilot, відкриває можливість суттєво оптимізувати процес створення тест-кейсів і BDD-сценаріїв, оскільки ці моделі здатні працювати з природною мовою, швидко інтерпретувати вимоги й формувати на їх основі структуровану тестову документацію. OpenAI GPT зазвичай використовують для глибокого опрацювання вимог, генерації альтернативних варіантів поведінки системи та перетворення неформальних описів у чітко сформульовані сценарії, що особливо корисно на етапах аналізу та проєктування тестів. Google Gemini робить акцент на контекстному розумінні складних структур і великих обсягів даних, завдяки чому здатний допомагати в систематизації вимог, узгодженні термінології та формуванні послідовних BDD-описів, які легко інтегруються в інструменти автоматизації. Microsoft Copilot найефективніший у середовищі розробки та тестування, де може безпосередньо працювати з кодом, пропонувати тестові сценарії для конкретних функцій, генерувати супровідну документацію та прискорювати процес створення автоматизованих тестів [14,15,16].

Переваги:

Використання цих моделей полягають у здатності суттєво скоротити час на підготовку тестової документації, забезпечити високу варіативність і повноту тестових сценаріїв та зменшити навантаження на аналітиків і тестувальників при роботі з великими системами або нечітко сформульованими вимогами. Крім того, моделі здатні виявляти неявні залежності, прогалини у вимогах та пропонувати додаткові сценарії, які часто залишаються поза увагою людини.

Недоліки:

Недоліки таких таких систем проявляються у можливій генерації неточних або некоректних сценаріїв, що потребує ретельної перевірки їх людиною, а також у залежності якості результатів від правильності поданого запиту. Важливо враховувати й питання безпеки, адже використання хмарних моделей потребує уважного ставлення до конфіденційності даних, які передаються для обробки.

Приклади використання:

Автоматичне перетворення текстових вимог у структуровані тест-кейси, створення BDD-сценаріїв за шаблоном Given-When-Then на основі опису бізнес-логіки, допомогу у виявленні пропущених сценаріїв або крайових випадків,

Формування тестової документації для нових функцій та прискорення роботи команд, що займаються як ручним, так і автоматизованим тестуванням.

Такі моделі активно застосовують під час аналізу вимог, у процесах CI/CD для генерації тестів, під час перевірки тестової документації та як інтелектуального помічника в командній взаємодії.

1.2 Порівняння існуючих інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації

Таблиця 1.1

Порівняння існуючих інтелектуальних систем

Інструмент	Тип інструменту	Опис	Переваги	Недоліки	Використання
Mabl	Хмарна автоматизація тестування (AI + low-code)	Інструмент для автоматичного енд-ту-енд тестування веб-додатків з використанням ШІ, low-code сценаріїв і візуального аналізу.	Простота використан-ня; автоматичне оновлення тестів; вбудована аналітика; інтеграція з CI/CD.	Обмеження для нетипових сценаріїв; висока вартість; залежність від хмари.	Тестування UI, регресійне тестування; автоматичне виявлення візуальних дефектів; моніторинг продуктивності.
Applitools Eyes	Візуальне тестування (AI)	Інструмент для візуального порівняння інтерфейсів, який визначає відмінності між зображеннями на різних версіях UI.	Висока точність; повне покриття UI; підтримка багатьох браузерів; інтеграція з Selenium.	Помилкові спрацюванн; складне налаштуванн я; залежність від якості базових знімків.	Перевірка UI на різних пристроях; перевірка локалізацій; регресійне візуальне тестування.
OpenAI ChatGPT	Генеративний ШІ (LLM)	Модель ШІ для генерації тексту, коду, документації, відповідей, тестових сценаріїв.	Універсальн ість; широкий функціонал; створення тестів, документаці ї; інтеграція з API.	Може помилятися; потребує перевірки; залежить від контексту; обмеженість у виконанні коду.	Генерація тест-кейсів; написання документації; рефакторинг коду; створення юзер-сторі.

Продовження таблиці 1.1

Порівняння існуючих інтелектуальних систем

Інструмент	Тип інструменту	Опис	Переваги	Недоліки	Використання
Google Gemini	Генеративний ШІ + мультимодальна модель	Потужна модель Google, що працює з текстом, зображеннями, відео та кодом, підтримує комплексну автоматизацію задач.	Потужна мультимодальність; інтеграція з Google Workspace; генерація коду та тестів.	Нестабільність у спеціалізованих галузях; можливі помилки; залежність від екосистеми Google.	Автоматизація тестів; аналіз логів; генерація документації; створення тестової стратегії.
Microsoft Copilot	AI-асистент у екосистемі Microsoft.	Глибока інтеграція з GitHub та Visual Studio; автогенерація коду; аналіз pull-requests.	Глибока інтеграція з GitHub та Visual Studio; автогенерація коду; аналіз pull-requests..	Вимагає Microsoft-екосистему; не завжди точний код; залежність від підписок.	Генерація unit-тестів у VS Code; аналіз PR; створення документації; автоматизація робочих процесів.
Diffblue Cover	Автоматична генерація unit-тестів (AI)	Автоматично створює JUnit-тести для Java-коду, використовуючи ШІ та аналіз байткоду.	Швидке створення тестів; високе покриття коду; інтеграція з CI/CD; автоматичне оновлення тестів.	Лише Java; висока вартість; інколи складний для читання згенерований код.	Генерація тестів для легасі-коду; підготовка до рефакторингу; автоматизація рутинних unit-тестів.
Parasoft	Платформа забезпечення якості (QA)	Комплексне рішення для тестування API, статичного аналізу, сервісної віртуалізації та автоматизації тестування.	Універсальність; підтримка стандартів; глибокий аналіз; потужні DevOps-інтеграції.	Висока ціна; складність налаштування; крута крива навчання.	Тестування корпоративних систем; статичний аналіз безпеки; віртуалізація сервісів для складних інтеграцій.

1.3 Виклики інтеграції, надійності та підтримки інтелектуальних систем на основі штучного інтелекту та шляхи їх подолання

Впровадження інтелектуальних систем на основі штучного інтелекту в тестування програмного забезпечення несе з собою певні виклики та ризики, які необхідно враховувати. Висока вартість ліцензування, складність інтеграції з існуючими інструментами, необхідність якісних даних для навчання моделей та прозорість роботи моделей штучного інтелекту є основними перешкодами для широкого впровадження, оскільки помилки в їхніх висновках можуть призвести до непередбачуваних результатів тестування.

Однією з основних проблем є якість даних, оскільки ефективність моделей ШІ безпосередньо залежить від якості та доступності даних для навчання . Складність моделей ШІ також може бути значним викликом, оскільки розуміння та інтерпретація рішень, прийнятих деякими алгоритмами, може бути складним завданням . Високі початкові витрати на придбання інструментів ШІ, інфраструктуру та навчання персоналу можуть стати перешкодою для деяких організацій . Існує також потреба у кваліфікованих фахівцях, які володіють знаннями як у тестуванні програмного забезпечення, так і в галузі ШІ та машинного навчання . Потенціал для упередженості в моделях ШІ, якщо вони навчаються на нерепрезентативних або упереджених даних, є серйозною етичною проблемою. Складність в інтерпретації результатів ШІ може ускладнити процес налагодження та прийняття рішень на основі отриманих даних . Крім того, інтеграція інструментів ШІ з існуючими системами та процесами тестування може бути складним завданням . Інші ризики включають можливість атакуючих атак на моделі ШІ , недостатнє розуміння контексту ШІ та надмірна залежність від автоматизації без належного людського контролю .

Для успішного впровадження інтелектуальних систем на основі штучного інтелекту в тестування програмного забезпечення слід дотримуватися певних найкращих практик. Важливо визначити чіткі цілі використання ШІ у стратегії

тестування, такі як покращення тестового покриття, скорочення часу виконання тестів або покращення виявлення дефектів [17]. Рекомендується починати з малого, зосереджуючись на автоматизації кількох ключових областей перед масштабуванням. Використання високоякісних даних для навчання моделей ШІ є критично важливим для отримання надійних результатів. Слід поєднувати можливості ШІ з досвідом та критичним мисленням людей-тестувальників, оскільки ШІ повинен доповнювати, а не замінювати їх [18]. Необхідно здійснювати постійний моніторинг та оцінку ефективності інструментів ШІ та якості їхніх результатів. ШІ особливо корисний для автоматизації повторюваних завдань, таких як генерація тестових даних та налаштування середовищ. Важливо також забезпечити безпеку та конфіденційність даних, що використовуються для навчання моделей ШІ [19]. Крім того, слід створювати культуру навчання в команді, заохочуючи експерименти з ШІ та пропонуючи можливості для постійного розвитку навичок. Рекомендується також використовувати звітність та моніторинг для відстеження якості виконання тестів та інтегрувати тестування ШІ в конвеєр CI/CD.

Інтеграція інтелектуальних систем на основі штучного інтелекту в тестування програмного забезпечення призводить до значних змін у ролі та обов'язках команд тестування. Відбувається зсув від ручного виконання до більш стратегічних завдань, таких як планування тестування, навчання та валідація моделей ШІ та дослідницьке тестування. ШІ бере на себе повторювані та рутинні завдання, дозволяючи тестувальникам зосередитися на більш складних та творчих аспектах, таких як розробка тестів для нових функцій та оцінка ризиків. Для ефективної роботи з інструментами та технологіями ШІ фахівцям з контролю якості (QA) необхідно набути нових навичок та пройти навчання. Це включає розуміння принципів роботи ШІ, вміння використовувати відповідні інструменти та інтерпретувати результати їхньої роботи. Хоча ШІ може автоматизувати багато завдань, людський контроль та експертиза залишаються важливими, особливо для прийняття рішень, розуміння контексту

та оцінки досвіду користувача [18]. Очікується, що ШІ сприятиме демократизації тестування та посиленню спільної відповідальності за якість серед команд розробки, але людські програмісти все ще необхідні для складних завдань, що вимагають інтуїції та креативності. Таким чином, роль тестувальників еволюціонує, стаючи більш стратегічною та аналітичною, з акцентом на співпраці з інструментами ШІ для досягнення вищої якості програмного забезпечення.

Встановлення фреймворків управління ШІ є надзвичайно важливим для забезпечення відповідального та етичного використання штучного інтелекту в тестуванні програмного забезпечення. Ці фреймворки включають політики, інструменти та практики, що визначають права прийняття рішень та відповідальність за використання ШІ, охоплюючи не лише управління даними, а й унікальні ризики, пов'язані з генеративним ШІ. Одним з ключових аспектів є виявлення та пом'якшення упереджень у моделях ШІ, що вимагає ретельного аналізу даних та оцінки моделей. Забезпечення конфіденційності та безпеки даних, що використовуються для навчання та тестування, є ще одним важливим етичним міркуванням [19]. Прозорість та пояснюваність моделей ШІ є важливими для побудови довіри до результатів тестування та полегшення налагодження. Визначення відповідальності за дії, вчинені системами ШІ, та забезпечення відповідності нормативним вимогам є також важливими елементами управління ШІ в тестуванні. Фреймворки управління ШІ часто включають такі компоненти, як ведення обліку моделей ШІ, проведення оцінок ризиків, визначення ролей та відповідальності, тестування моделей, відстеження законів та норм, створення документації, управління ризиками постачальників та навчання команд. Принципи етичного ШІ включають справедливість, підзвітність, прозорість та конфіденційність.

2. АНАЛІЗ МЕТОДІВ ТА МЕТРИК ОЦІНЮВАННЯ ЯКОСТІ СИСТЕМ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ ТА ТЕСТОВОЇ ДОКУМЕНТАЦІЇ

2.1 Метрики оцінювання якості згенерованої тестової документації

Оцінювання якості згенерованої тестової документації – це відносно новий напрям, який виник на стику автоматизованого тестування, природної мовної генерації та інженерії програмного забезпечення. Його мета – визначити, наскільки створені системою документи дійсно придатні для практичного використання: чи вони зрозумілі, точні, повні та відповідають вимогам до тестування.

Якість такої документації зазвичай розглядається у кількох вимірах, які разом формують цілісну оцінку. Першим і, мабуть, базовим виміром є адекватність змісту. Це означає, наскільки згенеровані описи тестів відображають реальну логіку тестування, коректно описують передумови, кроки виконання, очікувані результати і не містять суперечностей. Тут часто використовують семантичні метрики, які порівнюють зміст автоматично згенерованої документації з еталонною або з артефактами вимог. Якщо система тестів створюється для реального продукту, то адекватність може вимірюватися через відстань у векторному просторі між твердженнями в документації та специфікаційними описами поведінки системи.

Другим важливим аспектом є точність та узгодженість термінології. Згенерована тестова документація має дотримуватися стандартів проекту: послідовно використовувати назви компонентів, функцій, параметрів і сценаріїв. Тому сучасні системи оцінювання враховують лінгвістичну узгодженість, яка перевіряє сталість використання термінів і відсутність семантичного дрейфу між різними розділами. Метрики цього типу часто

базуються на статистичному аналізі словників або на embedding-моделях, які виявляють, чи не змінюється значення терміна в різних контекстах документа.

Ще один вимір якості – структурна організованість. Високоякісна тестова документація повинна мати чітку логічну побудову, зрозумілу ієрархію тестів, належні зв’язки між кроками та очікуваними результатами. Для цього аналізується внутрішня когерентність тексту: наскільки плавно пов’язані речення, чи не містить документ незавершених думок або розривів у логічному ланцюжку. Такі властивості оцінюють через метрики когезії, показники зв’язності на рівні абзаців або автоматичну перевірку граматичних і семантичних залежностей.

Не менш значущою є читабельність, яка визначає, наскільки легко інженеру зрозуміти згенеровану документацію. Це не лише граматика чи довжина речень, а й стиль, ясність інструкцій та відповідність корпоративним стандартам оформлення. Для цього можуть застосовуватися традиційні метрики читабельності на кшталт Flesch-Kincaid або сучасні нейронні оцінювачі, які враховують контекстуальну складність. Висока читабельність важлива не лише для зручності, але й для зниження ризику неправильного виконання тестів.

Окрему роль відіграє повнота тестової документації. Навіть якщо кожен окремий тест описаний грамотно, набір може бути неповним щодо всієї системи. Тому застосовують методи, які оцінюють покриття вимог або функціональних сценаріїв документацією. Це поєднання семантичного аналізу змісту з відображенням тестів на вимоги, що дозволяє виявити прогалини або дублювання.

В сучасних генеративних системах усе більшого значення набуває оцінка достовірності та правдивості. Документація, створена моделлю, може містити вигадані кроки, некоректні очікування або посилання на неіснуючі компоненти. Тому якість оцінюється не лише через лінгвістичні показники, а й через верифікацію фактів, яку іноді виконують інші моделі-оцінювачі або формальні перевірки проти специфікацій.

Нарешті, сучасні підходи приділяють увагу корисності з точки зору тестувальника. Навіть добре структурована документація може бути неефективною, якщо вона не допомагає швидше знайти дефекти чи не підтримує автоматизацію процесів. Тому ефективність часто перевіряють емпірично, через аналіз впливу використання документації на час тестування або відсоток знайдених помилок.

Таким чином, сучасні метрики оцінювання згенерованої тестової документації виходять далеко за межі простих показників точності тексту. Вони формують комплексну систему, що охоплює семантичну, структурну, стилістичну та практичну якість. Усе це дозволяє не лише виміряти, наскільки модель добре «пише» документи, а й зрозуміти, чи здатен такий контент реально підтримувати якісний процес тестування у виробничих умовах.

Автоматичні метрики порівняння текстів, такі як ROUGE і BERTScore, стали важливими інструментами для оцінювання якості згенерованої документації, зокрема тестової. Їхня головна ідея полягає в тому, щоб кількісно виміряти схожість між текстом, створеним системою, і деяким еталонним зразком, який вважається правильним або бажаним. Проте хоча ці метрики часто використовуються спільно, вони відрізняються як за підходом до вимірювання подібності, так і за здатністю враховувати глибину змісту.

Метрика ROUGE з’явилася ще у контексті автоматичного підсумовування текстів і орієнтована насамперед на лексичну схожість. Її принцип полягає в підрахунку, наскільки часто збігаються окремі слова або послідовності слів між згенерованим текстом і еталоном. У найпростішому вигляді вона відображає, яку частку лексичних одиниць системі вдалося відтворити. У тестовій документації це може показувати, наскільки точно генеративна модель повторює формулювання очікуваних результатів, описів кроків чи термінів. Водночас, така метрика має обмеження – вона не розуміє змісту. Якщо система сформулювала фразу інакше, але передала ту саму ідею, ROUGE може оцінити це як помилку. Тому в оцінюванні документації, де важливі не тільки слова, а й

зміст, ROUGE часто розглядають як нижчий рівень перевірки, корисний для вимірювання лексичної точності, але недостатній для аналізу семантики [20].

BERTScore, навпаки, представляє нове покоління метрик, заснованих на контекстуальних векторних представленнях слів. Вона не просто порівнює послідовності токенів, а використовує глибинну нейронну модель, таку як BERT або її варіанти, для обчислення подібності між сенсом кожного слова чи фрази в контексті речення. Це означає, що BERTScore може розпізнати, коли два тексти мають різну поверхневу форму, але однакове значення [21]. У сфері тестової документації це особливо важливо, оскільки описи кроків або умов можуть бути переформульовані різними способами, але залишатися логічно еквівалентними. Метрика дозволяє оцінити саме семантичну близькість між автоматично згенерованими інструкціями та тими, які підготувала людина.

Водночас, обидві метрики мають свої обмеження. ROUGE не враховує контекст і може завищувати оцінку текстів, які копіюють слова без розуміння сенсу, тоді як BERTScore потребує обчислювальних ресурсів і може бути чутливою до варіацій моделі-енкодера. Проте в оцінюванні документації вони часто доповнюють одна одну: ROUGE показує, наскільки точно система відтворює термінологію чи стандартні фрази, а BERTScore дозволяє перевірити, чи не спотворено зміст при генерації [22,23].

У сучасних дослідженнях обидві метрики дедалі частіше застосовують у поєднанні з людським оцінюванням, щоб перевірити, наскільки добре автоматичні показники узгоджуються з людським сприйняттям якості. Для тестової документації це особливо актуально, оскільки саме семантична правильність, послідовність і точність описів визначають її реальну корисність. Таким чином, ROUGE і BERTScore сьогодні розглядаються не як остаточний еталон, а як інструменти, що дають кількісну основу для аналізу якості, яку потім можна інтерпретувати у світлі практичних вимог до тестових артефактів.

2.2 Метрики продуктивності та вартості інтелектуальних систем

Метрики продуктивності та вартості інтелектуальних систем – це один із ключових напрямів сучасної аналітики в галузі штучного інтелекту, який дозволяє оцінювати не лише точність чи якість моделі, але й її ефективність у реальному використанні. Такі метрики охоплюють як технічні, так і економічні аспекти, відображаючи взаємозв'язок між обчислювальними витратами, часом відгуку, масштабованістю, і цінністю результатів, які система приносить організації чи користувачеві.

У технічному вимірі продуктивність інтелектуальної системи зазвичай пов'язана з її здатністю швидко та стабільно обробляти дані, реагувати на запити й забезпечувати заданий рівень якості результату. У контексті моделей машинного навчання та генеративного ШІ продуктивність часто визначається через час інференсу – тобто, скільки часу потрібно моделі, щоб обробити один запит або згенерувати відповідь. Цей показник стає особливо важливим у системах, де швидкість взаємодії впливає на користувацький досвід, наприклад, у чат-ботах чи рекомендаційних платформах. До того ж, важливим фактором є пропускна здатність – кількість запитів, які система може обробити за одиницю часу. Висока пропускна здатність свідчить про оптимізовану архітектуру та здатність масштабуватися під навантаженням.

Паралельно з цими аспектами розглядається ефективність використання обчислювальних ресурсів. Інтелектуальні системи, особливо ті, що базуються на великих нейронних мережах, можуть вимагати значних обсягів пам'яті, енергії та графічних процесорів. Тому в оцінюванні продуктивності дедалі частіше застосовують метрики, які описують співвідношення між якістю вихідного результату й витраченими ресурсами. Наприклад, співвідношення FLOPs до точності або показник latency-per-token у мовних моделях дають змогу оцінити, наскільки модель ефективна у своїй роботі без надмірного споживання енергії.

У сфері застосування інтелектуальних систем також враховується вартість помилки. Це поняття виходить за межі чисто технічних показників і оцінює, наскільки дорогими можуть бути наслідки неправильної дії моделі. У фінансових, медичних чи критичних інфраструктурах навіть незначна похибка може призвести до великих збитків. Тому продуктивність і вартість тут розглядаються у зв’язці з надійністю: наскільки система стабільна в непередбачуваних умовах і чи виправдані витрати на її використання з погляду ризиків.

У загальному підсумку метрики продуктивності та вартості формують багаторівневу систему оцінювання. На базовому рівні вони вимірюють швидкість, стабільність і ресурсну ефективність; на стратегічному – визначають економічну доцільність. Сучасна тенденція полягає в тому, щоб розглядати не лише показники продуктивності у відриві, а співвідносити їх із якістю результатів і вартістю їх досягнення. Найціннішою вважається та система, яка забезпечує високу якість і швидкість роботи при мінімальних витратах – як матеріальних, так і енергетичних. Таким чином, метрики продуктивності та вартості стають центральним інструментом для оптимізації інтелектуальних систем і визначення їх реальної ефективності в умовах масштабного застосування.

2.3 Людські методи оцінювання

Людські методи оцінювання якості, зокрема експертна оцінка та LQA є основою для глибокого, змістового аналізу інтелектуально згенерованого контенту, зокрема документації, перекладів або тестових матеріалів. На відміну від автоматичних метрик, що покладаються на статистичні або семантичні збіги, людське оцінювання дозволяє врахувати контекст, намір автора, специфіку предметної області та нюанси сприйняття тексту, які не здатні повноцінно відобразити алгоритмічні підходи.

Експертна оцінка передбачає участь фахівців, які володіють глибокими знаннями як у предметній галузі, так і в методології тестування або лінгвістичного аналізу. Такі експерти аналізують тексти за низкою критеріїв, що зазвичай охоплюють точність, повноту, узгодженість, адекватність стилю, ясність і відповідність очікуваному змісту. Цінність цього методу полягає в тому, що експерт може розпізнати не лише поверхневі помилки, але й логічні або змістові відхилення, які автоматичні метрики пропускають. Наприклад, у тестовій документації експерт може визначити, що кроки тесту технічно коректні, але не відповідають реальним сценаріям використання системи, або що очікуваний результат сформульовано двозначно. У такому випадку оцінювання має не лише кількісний, але й діагностичний характер – воно виявляє не просто недоліки, а й їхні причини.

LQA, або лінгвістичне оцінювання якості, історично виникло в галузі локалізації, але сьогодні активно застосовується і для аналізу згенерованих текстів. Його мета – систематизовано виміряти якість мови, стилю й точності переданого змісту. Процес LQA зазвичай включає детальну перевірку граматики, синтаксису, термінології, послідовності стилю й відповідності корпоративним чи технічним стандартам. На відміну від чисто експертної оцінки, яка може бути більш вільною та аналітичною, LQA часто має стандартизовану структуру: кожна помилка класифікується за типом (лексична, граматична, стилістична, термінологічна) та ступенем серйозності. Це дозволяє об’єктивніше оцінювати великі обсяги текстів, зберігаючи при цьому людську точність у розпізнаванні помилок.

Особливість людських методів оцінювання полягає у здатності враховувати інтенції тексту. Людина може розпізнати, чи відповідає стиль цільовій аудиторії, чи зберігається логічна послідовність викладу, чи не порушується тональність і змістові акценти. У документації це критично, адже навіть граматично правильний текст може бути неефективним, якщо він незрозумілий користувачеві або не передає точного контексту інструкції.

Ще однією перевагою людського оцінювання є можливість формування якісного зворотного зв’язку. Експерти не просто ставлять оцінку, а й надають пояснення, чому певний фрагмент вважається помилковим або невдалим, і як його можна покращити. Це робить такі методи особливо корисними у процесі вдосконалення генеративних моделей, оскільки коментарі експертів можуть бути використані для подальшого навчання або тонкого налаштування системи.

Проте людські методи оцінювання мають і свою специфіку – вони потребують часу, ресурсів і чіткої організації. Оцінювачі повинні бути належним чином підготовлені, а сам процес – стандартизований, щоб мінімізувати суб’єктивність. У практиці великих компаній для цього використовують подвійне рецензування або систему вагових коефіцієнтів для різних типів помилок. Зокрема, у LQA результати часто перетворюють у кількісні показники, що дозволяє поєднувати їх із автоматичними метриками й таким чином отримувати змішану, більш збалансовану картину якості.

Загалом, експертна оцінка та LQA не є просто альтернативою машинним метрикам – вони доповнюють їх, забезпечуючи глибину й контекст. Саме людське сприйняття здатне врахувати такі фактори, як природність мови, логічна послідовність і відповідність комунікативній меті тексту. Тому у сфері тестової, технічної чи генеративної документації саме людські методи залишаються остаточним еталоном якості, до якого звіряються всі інші автоматизовані способи оцінювання.

2.4 Порівняння існуючих методів та обґрунтування вибору комплексу метрик для методики

Щоб обґрунтувати вибір комплексу метрик для методики оцінювання якості згенерованої тестової документації, доцільно розглянути три основні групи існуючих підходів: експертні методи оцінювання, метрики продуктивності та вартості систем, а також. Кожен із них має власні переваги й обмеження, і лише їхнє поєднання дає повну картину якості створеного контенту.

2.4.1 Експертна оцінка

Експертна оцінка передбачає, що фахівці з тестування або предметні експерти вручну аналізують згенеровану тестову документацію, звертаючи увагу на її зміст, коректність, логічну узгодженість і відповідність вимогам. Оцінювання проводиться на основі професійного досвіду, знання продукту та прийнятих стандартів тестування, таких як ISO/IEC/IEEE 29119-3:2021 [24]. Експерт визначає, чи достатньо повно описані кроки, чи точні очікувані результати, чи не пропущено важливі сценарії, а також чи відповідає документ структурним вимогам організації.

Основна сила цього підходу полягає в тому, що він дозволяє не лише порівняти текст із вимогами, а й оцінити його справжню практичну придатність. Людина здатна помітити тонкі логічні помилки, невідповідність бізнес-процесам, надлишковість або недостатність кроків, а також неправильні інтерпретації функціональності – те, що автоматичні метрики зазвичай не виявляють. Завдяки цьому метод підходить для проєктів, де важлива безпека, регулювання або складна доменна логіка.

Водночас людська оцінка має істотні обмеження. Вона є повільною, затратною та погано масштабується на великі обсяги документації. До того ж на результати сильно впливає людський фактор: різні експерти можуть по-різному оцінювати один і той самий тест-кейс, що створює ризик суб’єктивності. Попри це, експертний підхід залишається найточнішим для перевірки змістовної валідності та логічної коректності тестової документації.

2.4.2 Автоматичні метрики схожості

Автоматичні метрики базуються на порівнянні згенерованої тестової документації з еталонною – тобто з уже існуючими «правильними» тест-кейсами, які вважаються золотим стандартом. ROUGE аналізує текст із погляду збігів на рівні слів або фраз, тоді як BERTScore використовує семантичні векторні уявлення, що дозволяє оцінювати змістовну близькість між текстами.

Цей підхід добре підходить для масового або швидкого оцінювання, оскільки повністю автоматизований. За допомогою нього можна одразу порівняти сотні або тисячі згенерованих документів, отримати числові оцінки якості, а також легко порівняти різні моделі генерації. Автоматичні метрики особливо ефективні у великих організаціях, де вже накопичено каталог якісних тест-кейсів, що можуть слугувати референсом.

Однак метрики схожості мають фундаментальне обмеження: вони оцінюють не те, наскільки тест-кейс коректний чи корисний, а те, наскільки він схожий на референсний текст. Це означає, що модель може створити логічно правильний і навіть кращий тест-кейс, який отримає низьку оцінку лише тому, що він викладений іншими словами. ROUGE особливо схильно до орієнтації на поверхневу відповідність, тоді як BERTScore хоч і враховує сенс, але залежить від якості та упередженості самої мовної моделі. Крім того, ефективність цього методу повністю залежить від наявності якісних, узгоджених і сучасних еталонних тестів, чого в реальних проєктах часто бракує [25].

2.4.3 LLM-as-a-Judge

Метод LLM-as-a-Judge полягає в тому, що якість тестової документації оцінюється за допомогою іншої великої мовної моделі, якій надають чіткі критерії оцінки. Модель аналізує документ згідно з інструкцією: перевіряє повноту, покриття вимог, логічну послідовність, конкретність очікуваних результатів, відповідність шаблону, узгодженість термінології та багато інших аспектів. По суті, модель імітує поведінку людського експерта, але робить це швидше, дешевше й у великому масштабі.

Цей підхід особливо цінний для проєктів із великим обсягом тестової документації або для CI/CD процесів, де потрібно оперативно оцінювати нові генерації тест-кейсів. Модель може автоматично знаходити проблеми, пропущені кроки або неузгодженості, а також повертати структуровану зворотну інформацію, що робить її корисною не лише для оцінки, а й для покращення якості документації.

Однак LLM-as-a-Judge не позбавлений недоліків. Мовні моделі можуть давати упереджені або непослідовні оцінки, особливо якщо критерії задані нечітко. Також можливі випадки “галюцинацій”, коли модель вигадує проблеми або, навпаки, пропускає очевидні помилки. Для деяких вузьких доменів вона може неправильно інтерпретувати специфічну термінологію або логіку системи, якщо не має достатнього контексту. Тому LLM-оцінювання добре працює в поєднанні з контрольними вибірками ручної перевірки чи із суворо визначеними правилами оцінювання.

Порівнюючи ці підходи, можна зробити висновок, що жоден із них не забезпечує достатньої повноти оцінювання окремо. Автоматичні метрики дають кількісну основу, але не відображають якість сприйняття тексту. Людські методи фіксують смислові й логічні аспекти, але не масштабуються на великі масиви даних. Метрики продуктивності, своєю чергою, встановлюють техніко-економічні межі ефективності, не аналізуючи змісту документації. Тому оптимальним є інтегрований підхід, у якому кожен тип метрик виконує власну функцію в межах єдиної методики.

Обґрунтований вибір комплексу метрик для такої методики має ґрунтуватися на поєднанні трьох рівнів оцінювання. На базовому рівні застосовуються автоматичні метрики (наприклад, ROUGE та BERTScore) для швидкої кількісної оцінки схожості з еталоном. На семантико-логічному рівні використовуються результати експертного або LQA-аналізу, які дозволяють виявити змістові недоліки, двозначності чи структурні проблеми. Нарешті, на системному рівні враховуються показники продуктивності, вартості та енергетичної ефективності, щоб оцінити, наскільки обрана технологія є раціональною для практичного впровадження.

Таблиця 2.1

Порівняння існуючих методик

Методика	**Можливості**	**Недоліки**
Експертна оцінка	Висока точність; Перевірка логіки; Врахування контексту.	Висока собівартість; Низька швидкість; Емпірична варіативність; Персоналізована інтерпретація.
Автоматичні метрики (ROUGE/BERTScore)	Висока швидкість; Структурований результат; Автономно.	Формалістична оцінка; Не оцінює логіку; Нормативно-орієнтоване оцінювання.
LLM-as-a-Judge	Баланс швидкості та змістовності; Здатність виявляти логічні помилки.	Можлива упередженість; Можливі галюцинації системи.

3. РОЗРОБКА ТА АПРОБАЦІЯ МЕТОДИКИ ОЦІНЮВАННЯ ЯКОСТІ ТА ОПТИМІЗАЦІЇ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ ДЛЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ ТА ГЕНЕРАЦІЇ ТЕСТОВОЇ ДОКУМЕНТАЦІЇ

3.1 Обґрунтування необхідності гібридного підходу

Більшість сучасних автоматизованих метрик, таких як ROUGE або BERTScore, були створені для оцінювання якості машинного перекладу чи текстового узагальнення. Вони базуються на обчисленні лексичних та синтаксичних збігів між згенерованим і еталонним текстами, не враховуючи глибинний зміст або контекстну релевантність. Проте у випадку тестових сценаріїв або документації цього недостатньо: два тексти можуть бути схожими за структурою, але один із них може описувати неправильну логіку тесту або некоректно формулювати очікувані результати. Дослідження показують, що автоматичні метрики часто корелюють із людською оцінкою лише частково, особливо коли тексти мають складну семантичну або логічну структуру [26].

У сфері автоматизації тестування програмного забезпечення така обмеженість особливо критична. Якість тестів визначається не тільки лексичною коректністю, але й повнотою охоплення функціональних вимог, відповідністю специфікаціям, правильністю логіки перевірок і зрозумілістю для розробників чи тестувальників. Автоматичні метрики не можуть виявити логічні суперечності в тестових кроках або семантичні відхилення в описах очікуваних результатів. Як показано у роботах із критичного аналізу оцінок машинного перекладу, навіть сучасні метрики не здатні замінити людське судження в контекстах, де потрібне глибоке розуміння цілей тексту.

Ще одним суттєвим обмеженням автоматичних метрик є неврахування доменної специфіки. У тестовій документації часто присутні галузеві стандарти, корпоративні вимоги до стилю, унікальні терміни чи формати подання. Метрики, побудовані на лексичній схожості, не здатні оцінити відповідність цим аспектам.

Саме тому в сучасних дослідженнях активно розвивається концепція Human-in-the-Loop (HITL) – підходу, який поєднує автоматичні засоби оцінювання з експертною людською участю. Людина в цьому процесі виступає когнітивним контролером, здатним оцінювати семантичну адекватність, логічну послідовність і практичну корисність результатів, тоді як автоматичні метрики забезпечують швидкість і узгодженість вимірювань [27,28]. Така взаємодія дає змогу побудувати адаптивну систему оцінювання, у якій результати автоматичних метрик доповнюються експертною оцінкою, а зворотний зв’язок від людини використовується для подальшого навчання моделей і покращення генерації тестів [29].

Таким чином, гібридний підхід є не лише теоретично обґрунтованим, але й практично необхідним. Він компенсує недоліки статистичних метрик, враховує когнітивні та контекстуальні аспекти якості, а також забезпечує баланс між автоматизацією та людським контролем. Для інтелектуальних систем, які генерують тестову документацію, це означає не просто вимірювати формальні збіги, а оцінювати реальну відповідність створених матеріалів завданням тестування та потребам проєкту.

Інтеграція великих мовних моделей, зокрема архітектур на основі трансформера, кардинально змінила підходи до розробки програмного забезпечення. Ці моделі демонструють високу здатність до генерації природних та зв'язних текстів [30]. У сфері забезпечення якості та тестування, ця технологія відкриває можливості для значної автоматизації процесів генерації тест-кейсів.

Основною стратегічною метою впровадження ВММ у QA є прискорення циклу розробки, зниження часових витрат на створення тестів та досягнення вищого рівня тестового покриття, що сприяє загальній ефективності та якості ПЗ.

Незважаючи на вражаючі можливості, які демонструють ВММ, критична проблема виникає при спробі об'єктивної оцінки якості згенерованих артефактів. Звітність про високу точність та релевантність відповідей, заснована на лінгвістичних показниках, може створювати хибне враження про функціональну надійність. Ключовий виклик полягає у відсутності надійних автоматичних механізмів, які б гарантували, що згенеровані тестові випадки та супровідна документація не лише виглядають синтаксично коректно, але й є логічно, функціонально та семантично валідними для цільової системи.

Для критичних систем, де помилки можуть мати серйозні наслідки, цей розрив між лінгвістичною якістю та функціональною коректністю вимагає радикального перегляду методології оцінювання. Необхідний перехід від суто кількісних, автоматичних метрик до гібридного підходу, який інтегрує критичне мислення, контекстуальне розуміння та експертну валідацію, що може надати лише HITL.

Традиційні автоматичні метрики, такі як ROUGE, широко застосовуються для оцінки якості відповідей ВММ у завданнях обробки природної мови .[31] Ці метрики оцінюють схожість згенерованого тексту з еталонними відповідями. Хоча вони надають об'єктивне кількісне вимірювання лексичної та синтаксичної близькості, їх застосування до згенерованого програмного коду або технічної документації виявляється методологічно обмеженим.

Фундаментальний недолік цих метрик полягає у їхній нездатності вимірювати семантичну еквівалентність та, як наслідок, функціональну коректність. Використання ROUGE для оцінки якості згенерованих тестів створює хибну кореляцію між синтаксичною якістю та функціональною валідністю. Критичні емпіричні дані показують, що програми, які є

функціонально некоректними, можуть отримувати вищі показники, ніж програми, які є функціонально еквівалентними еталону. Це явище свідчить про те, що високий бал за лінгвістичними метриками не лише не є гарантією якості виконуваного тестового коду, але й може вводити в оману, сигналізуючи про якість там, де насправді приховані логічні помилки. Оскільки ВММ навчаються на синтаксично коректних наборах даних, вони здатні генерувати артефакти, що мають високу оцінку флюенсу, але містять логічні помилки, які ці автоматичні метрики не можуть виявити.

Спроби вдосконалити автоматичне оцінювання за рахунок використання ембединг-орієнтованих методів, таких як BERTScore, які прагнуть захопити більше семантики, також зіткнулися зі значними обмеженнями. Дослідження підтверджують, що навіть ці моделі не завжди здатні точно фіксувати семантичну еквівалентність коду [31].

Один із найбільших ризиків, пов'язаних з інтелектуальною генерацією, – це феномен галюцинацій ШІ. Галюцинації виникають, коли моделі генерують правдоподібний, але фактично некоректний або хибний результат, який надзвичайно важко перевірити суто автоматичними засобами [32].

Дослідження виявили кілька типових ситуацій некоректної роботи ВММ:

1. Неправильна інтерпретація намірів користувача.
2. Галюцинація моделі (виведення логічно хибної інформації).
3. Неправильне використання прикладних програмних інтерфейсів, часто пов'язане з відсутніми елементами або зловживанням сторонніми бібліотеками.

Особливо небезпечною є схильність моделей до помилок при взаємодії зі складними бібліотечними API. Це може призвести до збоїв у роботі ПЗ та виникнення вразливостей у безпеці. Оскільки автоматичні метрики ігнорують логічну коректність, фокусуючись на формі, існує значний ризик інтеграції помилкового або неоптимального результату, який автоматична система оцінювання помилково схвалить. Єдиний надійний спосіб виявити логічні

помилки, приховані за правдоподібним синтаксисом, – це критична експертиза людини.

Гібридний підхід, відомий як "Людина в циклі", є критично важливим стратегічним інструментом, що забезпечує запобіжнику процесах, автоматизованих ШІ. Він поєднує швидкість та масштабованість машини з критичним міркуванням, етичною оцінкою та наглядом, які може надати лише людина.

Впровадження HITL є незамінним у високоризикових середовищах, таких як фінанси, охорона здоров'я або автономні системи. У таких випадках помилки, спричинені оптимізацією ШІ під кількісні метрики без урахування людських цінностей або контексту, можуть призвести до значних фінансових втрат, юридичних проблем або репутаційної шкоди. HITL слугує системою захисту, вводячи критичні контрольні механізми, які дозволяють експертам виявляти та виправляти помилки. Крім того, людські судження сприяють зменшенню упереджень, забезпеченню чесності та адаптації систем до складності реальних сценаріїв, де ШІ часто не вистачає контекстуального розуміння.

У контексті QA, роль людини виходить за межі простого виправлення помилок і включає необхідну валідацію та верифікацію артефактів. Лише людина-експерт, яка розуміє бізнес-контекст та кінцеві потреби користувача, може здійснити валідацію. Валідація гарантує, що згенеровані тест-кейси не лише відповідають специфікаціям, але й задовольняють реальні потреби користувачів та зберігають цілісність бізнес-логіки.

Експертна оцінка тестових завдань включає введення логічної тестової послідовності інформації та контроль отриманих результатів, який здійснюється або візуально, або за допомогою допоміжних програмних засобів. Цей процес є ключовим для підтвердження, що функція виконується належним чином, відповідно до очікувань.

Гібридні системи повинні включати механізми, що забезпечують

прозорість. Така можливість аудиту цілісності вмісту, згенерованого моделлю, є фундаментальною для впровадження HITL у процеси QA.

З огляду на схильність ВММ до генерації логічно некоректного, але синтаксично правильного результату, це створює ситуацію, коли автоматичні метрики підтверджують якість артефакту, який насправді містить критичні помилки. Людська експертиза, задіяна в циклі, є єдиним надійним механізмом для виявлення цих прихованих логічних помилок, перетворюючи HITL на каталізатор безперервної адаптивності та операційної гнучкості, особливо коли моделі стикаються з нетиповими сценаріями або зміною правил.

У випадку тестової документації , лінгвістичні метрики, такі як ROUGE, є майже повністю нерелевантними, оскільки вони не оцінюють якість вимог як інженерного артефакту.

Документація, згенерована ШІ, повинна підлягати суворій експертній оцінці на відповідність інженерним стандартам якості, що забезпечують її придатність для тестування: недвозначність, тестованість, повнота, послідовність та читабельність для людини.

Для забезпечення повної оцінки якості, методика повинна систематично інтегрувати кількісні та якісні критерії.

Гібридний підхід, заснований на HITL, є не лише механізмом контролю якості, але й основою для безперервного вдосконалення ВММ, які генерують артефакти.

Цикл зворотного зв’язку – це постійний процес збору, аналізу та використання взаємодій користувачів для оптимізації моделі, підвищення її точності та загальної зручності використання. Критично важливим є залучення тестувальників та розробників до оцінки якості згенерованих тестів та надання структурованого зворотного зв'язку. Такий експліцитний зворотний зв'язок дозволяє швидко виявляти та виправляти недоліки, підвищуючи загальну ефективність тестування. Цей якісний зворотний зв'язок від експертів є фундаментальним ресурсом для вдосконалення моделі. Він використовується

для тонкого налаштування моделі, допомагаючи їй еволюціонувати та відповідати реальним інженерним стандартам та очікуванням користувачів. Корекція, внесена людиною-експертом, яка виправляє логічні помилки, неправильне використання API або неточну бізнес-логіку, перетворюється на високоякісний, верифікований навчальний набір даних. Таким чином, HITL не лише контролює якість поточного виводу, але й забезпечує підвищення якості майбутніх ітерацій моделі, перетворюючи людський нагляд на критичний навчальний ресурс.

Найуспішніші реалізації HITL передбачають інтеграцію людського нагляду з самого початку розробки системи ШІ, а не ретроспективне застосування після виникнення проблем. Проактивне залучення людського судження на етапі початкового вибору та маркування даних створює міцну основу для систем ШІ. ВММ демонструють високу ефективність у передбачуваних сценаріях, але часто виявляють недоліки в умовах невизначеності, при роботі з неповними даними або при зміні типових правил. Людська експертиза в циклі дозволяє моделі адаптуватися до змінних ринкових умов та нетипових сценаріїв. Це забезпечує операційну гнучкість, зберігаючи безперервність, точність та контроль, коли моделі стикаються з винятками. Це робить HITL незамінним для створення адаптивних та надійних інтелектуальних систем.

Аналіз методик оцінювання інтелектуальних систем для автоматизації тестування програмного забезпечення доводить, що суто автоматичні підходи є принципово недостатніми для забезпечення необхідного рівня якості та надійності. Автоматичні метрики, такі як ROUGE, провалюються у вимірюванні функціональної коректності та семантичної еквівалентності. Вони оцінюють форму, а не поведінку, що призводить до хибної кореляції та ризику інтеграції функціонально некоректного коду, який отримує високу автоматичну оцінку. Системна схильність ВММ до генерації логічно некоректного, але правдоподібного результату становить неприпустимий ризик у

високотехнологічних QA процесах. Людська експертиза є незамінною для валідації бізнес-логіки , логічної коректності та оцінки здатності виявляти складні дефекти.

Ключовим елементом є вбудований, проактивний механізм "Людини в циклі", який не лише коригує вихідні дані, але й забезпечує постійний цикл зворотного зв'язку. Цей цикл підвищує адаптивність та надійність базової ВММ, перетворюючи людський нагляд на навчальний ресурс.

3.2 Концептуальна модель методики

3.2.1 Розробка кількісних метрик методики

1. Середній час генерації

Метрика вимірює середній час, який витрачається на генерацію одного тест-кейсу, дозволяючи оцінити швидкість та ефективність процесу формування тестової документації. Вона допомагає порівнювати різні моделі, налаштування або підходи до генерації за показником продуктивності.

$$T = \frac{t}{N}, \tag{3.1}$$

де T – середній час генерації одного тест-кейсу;

t – загальна кількість часу витраченого на генерування всіх тест кейсів;

N – кількість згенерованих тест-кейсів.

2. Покриття вимог

Метрика визначає, яку частку функціональних вимог охоплено хоча б одним тест-кейсом.Покриття вимог є однією з базових метрик оцінювання тестів у стандартах ISO/IEC/IEEE 29119-3:2021. Мета – визначити, яка частка функціональних вимог системи охоплена хоча б одним тест-кейсом [33].

$$RC = \frac{N_{cov}}{N_{total}} \times 100\%, \quad (3.2)$$

де RC – відсоток покриття вимог;

N_{cov} – кількість вимог, для яких створено хоча б один тест-кейс;

N_{total} – загальна кількість вимог.

Високе значення RC свідчить про те, що тест кейси відображають логіку системи в достатньому обсязі.

3. Повнота тест-кейсу

Метрика перевіряє наявність обов’язкових атрибутів, таких як:

ID тест-кейсу, назва, передумови, кроки виконання, тестові дані, очікуваний результат. У міжнародних стандартах [33] зазначено мінімальний перелік атрибутів тест-кейсу. Ступінь їх наявності визначає його повноту

$$TCC = \frac{N_{attr\_present}}{N_{attr\_required}} \times 100\%, \quad (3.3)$$

де TCC – повнота тест-кейсу;

$N_{attr\_present}$ – кількість фактично заповнених атрибутів тест-кейсу;

$N_{attr\_required}$ – кількість атрибутів, які повинні бути заповнені згідно зі стандартом.

Значення менш ніж 100% вказує на неповну або некоректно сформовану документацію.

4. Когнітивна складність кроків

Складність кроків оцінюється як середня довжина описів кроків:

$$ALS = \frac{1}{N_{step}} \sum_{i=1}^{N_{step}} w_i , \quad (3.4)$$

де ALS – середня довжина описів кроків;

N_{step} – загальна кількість кроків у тест-кейсах;

w_i – кількість слів у і-му кроці.

Занадто довгі або складні кроки знижують читабельність і підвищують ризик помилок виконання.

3.2.2 Розробка якісних метрик методики

Оцінювання якості результатів, отриманих за допомогою інтелектуальних систем для автоматизації тестування програмного забезпечення, неможливо здійснити виключно на основі об’єктивних числових показників. Кількісні метрики (час генерації, покриття коду, мутаційний рахунок тощо) забезпечують кількісну характеристику системи, однак вони не враховують когнітивні та контекстуальні аспекти якості – такі як логічна послідовність тестів, зрозумілість формулювань, узгодженість з вимогами, або релевантність знайдених дефектів [34]. Саме тому у межах гібридного підходу доцільно застосовувати якісні метрики, що базуються на експертній оцінці .

Основна мета застосування якісних метрик полягає у формуванні систематизованої процедури експертної перевірки якості тестів і супровідної документації, яку згенерувала інтелектуальна система. Така оцінка дає змогу визначити семантичну, логічну та практичну адекватність згенерованих результатів, що не може бути достовірно виміряно автоматичними інструментами.

Кожен із параметрів оцінюється за шкалою Лайкера – за п’ятибальною шкалою.

Шкала Лайкерта – це метод вимірювання ставлення, думок чи рівня згоди людей із певними твердженнями. Її сутність полягає в тому, що респонденту пропонують твердження, а він обирає рівень згоди або незгоди з ним. Формулювання відповідей може бути різним, але класичний варіант передбачає градації від повної згоди до повної незгоди. Часто використовують п’ятирівневу шкалу.

Цей метод зручний тим, що дозволяє перевести суб’єктивні відчуття людини у кількісні показники, які можна статистично аналізувати. Дослідники отримують структуровані дані, які потім можуть агрегувати, будувати середні значення, аналізувати тенденції або порівнювати групи між собою.

У змістовному плані важливим є те, що шкала Лайкерта вимірює не сам факт, а інтенсивність ставлення. Вона добре працює з абстрактними поняттями на кшталт задоволеності, рівня прийнятності чи бажаності. Наприклад, можна з’ясувати не просто те, чи подобається людині послуга, а наскільки сильно їй це подобається. Саме ця градаційність робить шкалу корисною в аналітиці.

Важливо також розуміти, що відповіді на шкалі Лайкерта є порядковими: кожен наступний рівень означає більше або менше згоди, але різниця між пунктами не обов’язково математично рівна. Проте в прикладних дослідженнях їх часто трактують як інтервальні, що дає змогу використовувати середні значення, коефіцієнти кореляції та інші методи кількісного аналізу..

Оптимальність діапазону від 1 до 5 у шкалі Лайкерта не випадкова. Вона сформувалася як компроміс між точністю вимірювання та комфортом для респондента. Коли людина відповідає на запитання, вона має утримувати в пам’яті варіанти відповідей, інтерпретувати їх і швидко зробити вибір. П’ять пунктів – це межа, яка дозволяє зберегти інтуїтивність і водночас забезпечити достатню диференціацію відповідей. Якщо шкала стає ширшою, наприклад сім чи дев’ять пунктів, частина людей починає відчувати невпевненість, бо межі

між сусідніми позиціями стають менш очевидними. Це призводить до збільшення випадковості відповідей і зниження надійності.

П’ятирівнева шкала також добре збалансована з погляду статистичного аналізу. Вона забезпечує симетрію: дві позиції означають згоду різної інтенсивності, дві – незгоду, а центральна точка служить нейтральною або «важко сказати». Така структура робить розподіли відповідей передбачуваними та зручними для обробки, дозволяє легко порівнювати групи, будувати середні значення і спостерігати тенденції.

Тому п’ятирівнева шкала утвердилася як стандарт: вона досить деталізована для дослідника й водночас достатньо проста та психологічно комфортна для респондента.

Якісні метрики були розроблені у вигляді чек листів:

1. Зміст тестів-кейсів

☐ Тест-кейси мають зрозумілі назви

☐ Кроки описані послідовно

☐ Очікувані результати чітко вказано

2. Документація

☐ Документи узгоджені між собою

☐ Не виявлено дублікатів

☐ Використано уніфіковану термінологію

3. Якість результатів

☐ Використання неточних, або двозначних термінів

☐ Виявлені баги мають практичну значущість

☐ Документація зрозуміла без контексту розробника

Шкала оцінювання:

1 – Дуже низька якість.

Тест-кейси мають суттєві недоліки, які унеможливлюють їх коректне використання.

2 – Низька якість.

Є відчутні прогалини та неточності, які заважають стабільному використанню. Тест-кейси потребують значного доопрацювання та уточнення.

3 – Середня якість.

Тест-кейси загалом зрозумілі і виконувані. Недоліки не критичні, але помітні. Застосування можливе без великих труднощів, проте покращення явно підвищить ефективність.

4 – Висока якість.

В тест-кейсах спостерігаються лише дрібні недоліки, що не впливають на використання. Найважливіші частини оформлені коректно і відповідають очікуваним стандартам.

5 – Відмінна якість.

Тест-кейси оформлено максимально чітко, логічно та повністю відповідно до стандартів. Недоліки відсутні або мінімальні й не мають значення.

3.3 Формула єдиного гібридного індексу якості

Якісні показники є доповненням до кількісних метрик, формуючи двокомпонентну систему оцінювання, де кількісні показники забезпечують об’єктивність, а експертна оцінка – семантичну глибину. У результаті формується інтегрований індекс якості, що враховує обидва типи вимірювань:

Кількісні метрики фіксують формальні показники – швидкість, покриття, стабільність, валідність.

Якісні метрики через чек-лист або анкету дозволяють оцінити когнітивні якості результатів – наскільки документація реально зрозуміла, практична та корисна.

Такий підхід забезпечує баланс між об’єктивністю та осмисленістю оцінки, дозволяючи оцінювати не лише технічну правильність результатів, але й їх зрозумілість, практичну застосовність і когнітивну якість.

Застосування якісних метрик у складі комплексної методики оцінювання дозволяє сформувати комплексну систему вимірювання якості інтелектуальних інструментів для тестування ПЗ. Поєднання кількісних і якісних показників забезпечує надійність, контекстну релевантність і практичну придатність оцінки результатів генерації тестів і супровідної документації. Таким чином, гібридний підхід з якісною складовою створює адаптивну та збалансовану систему оцінювання, що відповідає вимогам сучасних підходів до тестування програмного забезпечення та розробки інтелектуальних систем.

Сучасні інтелектуальні системи для автоматизації тестування програмного забезпечення генерують значні обсяги тестів і супровідної документації, які необхідно оцінювати не лише з позиції технічної коректності, а й з урахуванням якості змісту, логіки та практичної придатності. Для досягнення цього у межах дослідження розроблено комплексну методику оцінювання, яка поєднує кількісні та якісні метрики в єдину інтегровану оцінку.

Таким чином, інтеграція двох груп метрик дозволяє сформувати збалансовану методику оцінювання, у якій технічна об’єктивність підкріплюється когнітивною експертизою людини.

Перед об’єднанням усі метрики мають бути приведені до єдиної шкали, наприклад, [0; 1].

Для цього використовується мін-макс нормалізація:

$$Q' = \frac{Q - Q_{min}}{Q_{max} - Q_{min}} \quad , \qquad (3.5)$$

де Q' - нормалізоване значення метрики;

Q - фактичне значення метрики;

Q_{min} - мінімальне значення в наборі даних;

Q_{max} - максимальне значення в наборі даних.

Але для метрик середній час генерації та когнітивна складність кроків використовується зворотня мін-макс нормалізація, оскільки чим нижче значення даних метрик, тим вище оцінюється робота інтелектуальних систем.

$$Q' = \frac{Q_{max} - Q}{Q_{max} - Q_{min}} \quad , \tag{3.5}$$

де Q' - нормалізоване значення метрики;

Q - фактичне значення метрики;

Q_{min} - мінімальне значення в наборі даних;

Q_{max} - максимальне значення в наборі даних.

Метрики покриття вимог та повнота тест-кейсу не потребують нормалізації, оскільки їх значення вже знаходяться в діапазоні [0; 1].

Після нормалізації значень визначається середній кількісний показник якості:

$$Q_{auto} = \frac{1}{n} \sum_{i=1}^{n} Q'_i \quad , \tag{3.6}$$

де Q_{auto} – середній кількісний показник якості;

Q'_i – нормалізоване значення і-ї кількісної метрики;

n – кількість кількісних метрик.

Та обчислюється середній якісний показник якості:

$$Q_{exp} = \frac{1}{m} \sum_{j=1}^{m} Q'_i \quad , \tag{3.7}$$

де Q_{exp} – середній якісний показник якості;

Q'_i – нормалізоване значення j-ї якісної метрики;

m – кількість якісних метрик.

Таким чином, як кількісні, так і якісні метрики зводяться до уніфікованої шкали для подальшої інтеграції у формулу HQI.

У межах гібридного підходу пропонується розраховувати індекс якості системи, який узагальнює результати кількісного аналізу та експертного оцінювання.

У результаті формується зважена комбінована оцінка якості

$$HQI = \alpha \times Q_{auto} + (1 - \alpha) \times Q_{exp}, \quad (3.8)$$

де HQI – гібридний індекс якості;

Q_{auto} – середнє значення кількісних метрик (T, RC, TCC, ALS);

Q_{exp} – середнє значення якісних метрик, отриманих на основі експертного чек-листа;

α – ваговий коефіцієнт, який визначає відносну важливість кількісних метрик у загальній оцінці ($0 \leq \alpha \leq 1$).

Вагові коефіцієнти α та $1 - \alpha$ визначають співвідношення між машинними і людськими оцінками.

Їх значення залежить від характеру системи, цілей оцінювання та рівня автономності генератора тестів.

Таблиця 3.1

Таблиця пояснень вибору значення α

Характер процесу	**Рекомендоване значення α**	**Пояснення**
Мінімальне втручання людини	0.7-0.8	Основний акцент - на точності алгоритму та продуктивності
Людина бере участь у перевірці тестів	0.5-0.6	Баланс між швидкістю та якістю формулювань
Людина активно взаємодіє із системою	0.3-0.4	Основний акцент - на зручності, зрозумілості та адаптивності

В окремих випадках ваги можуть бути визначені експериментально на основі статистичної оцінки кореляції між автоматичними й експертними результатами.

Запропонована методика забезпечує об’єктивність, завдяки кількісним показникам, контекстуальну точність, завдяки експертній участі, гнучкість налаштування, через регулювання вагових коефіцієнтів та пояснюваність результатів, що особливо важливо для інтелектуальних систем на основі штучного інтелекту в тестуванні.

Таким чином, методика дозволяє реалізувати адаптивну систему оцінювання, де результати не лише вимірюються, але й аналізуються, забезпечуючи високий рівень довіри до автоматизованих інструментів.

3.4 Алгоритми оцінювання якості та оптимізації роботи інтелектуальних систем

Запропонована комплексна методика оцінювання якості інтелектуальних систем для автоматизації тестування програмного забезпечення передбачає

поетапне поєднання автоматизованого аналізу результатів із експертною оцінкою, що дозволяє сформувати інтегровану характеристику ефективності роботи системи. Алгоритм застосування цієї методики складається з низки логічно пов’язаних етапів, спрямованих на забезпечення об’єктивності, повторюваності та пояснюваності процесу оцінювання.

3.4.1 Формування вхідних даних та збір вихідних даних про результати роботи системи

На першому етапі здійснюється формування вхідних даних та збір результатів, отриманих у процесі роботи інтелектуальної системи автоматизованого тестування. До таких результатів належать згенеровані тест-кейси. На цьому етапі важливо забезпечити повноту зібраних даних, оскільки подальше оцінювання якості залежить від репрезентативності вибірки.

До вхідних даними належать екрана форма логіну, вимоги до форми та промти для інтелектуальних систем

3.4.2 Обчислення кількісних метрик

Другий етап полягає у розрахунку кількісних метрик, що характеризують технічну якість і продуктивність системи. До таких метрик належать:

-середній час генерації одного тест-кейсу;

-відсоток покриття вимог;

-повнота тест-кейсу;

-середня довжина описів кроків.

3.4.3 Проведення експертної оцінки

На третьому етапі здійснюється якісне оцінювання результатів роботи системи за допомогою експертного чек-листа або анкети. Експерт оцінює зміст тестів, якість документації та загальну якість результатів. Оцінювання проводиться за п’ятибальною шкалою Лайкера, після чого обчислюється

усереднений якісний показник. Результати експертного аналізу документуються, щоб забезпечити відтворюваність оцінки у наступних ітераціях.

3.4.4 Інтеграція результатів у єдиний індекс якості

На четвертому етапі здійснюється інтеграція результатів автоматичного та експертного аналізу у єдиний гібридний індекс якості (HQI).

У більшості практичних випадків α встановлюється у межах 0.5–0.7, що забезпечує баланс між точністю машинного аналізу та глибиною людської інтерпретації. Рівні інтерпретації наведені в таблиці 3.2.

Таблиця 3.2

Таблиця інтерпретації значень HQI

Діапазон HQI	**Рівень якості**	**Рекомендації**
0.85 – 1.00	Високий	Система стабільна, тести якісні, документація повна
0.65 – 0.84	Середній	Необхідна оптимізація певних аспектів (логіка, стиль, узгодженість)
< 0.65	Низький	Потрібне доопрацювання генератора тестів або моделі документації

Такий підхід дозволяє не лише кількісно оцінити рівень ефективності системи, а й визначити напрямки для її оптимізації: підвищення валідності

тест-кейсів, покращення стилістики документації або корекцію моделей генерації. Крім кількісного результату, експерт надає якісний зворотний зв’язок, який допомагає ідентифікувати вузькі місця системи.

Алгоритм застосування комплексної методики забезпечує поєднання обчислюваної об’єктивності та когнітивної інтерпретації, створюючи прозорий і гнучкий механізм оцінювання інтелектуальних систем.

Він дозволяє не лише вимірювати якість, а й формувати адаптивний цикл покращення, у якому результати оцінювання стають основою для подальшої оптимізації алгоритмів генерації тестів.

3.4.5 Оптимізація системи за результатами оцінювання

Останній етап полягає у зворотному зв’язку – результати оцінювання використовуються для навчання або налаштування інтелектуальної системи.

Можливі напрями оптимізації:

-корекція алгоритмів генерації тестів (наприклад, уникнення дублювання сценаріїв);

-покращення лінгвістичних моделей для формування документації;

підвищення узгодженості між тестами та вимогами.

Після отримання нових результатів та обрахування HQI, можна оцінити наскільки було оптимізовано роботу системи. Таким чином, комплексна методика не лише оцінює якість, а й створює механізм вдосконалення системи через участь людини у процесі оптимізації. Блок схема алгоритму оцінювання якості та оптимізації роботи інтелектуальних систем зображена на рисунку. 3.1

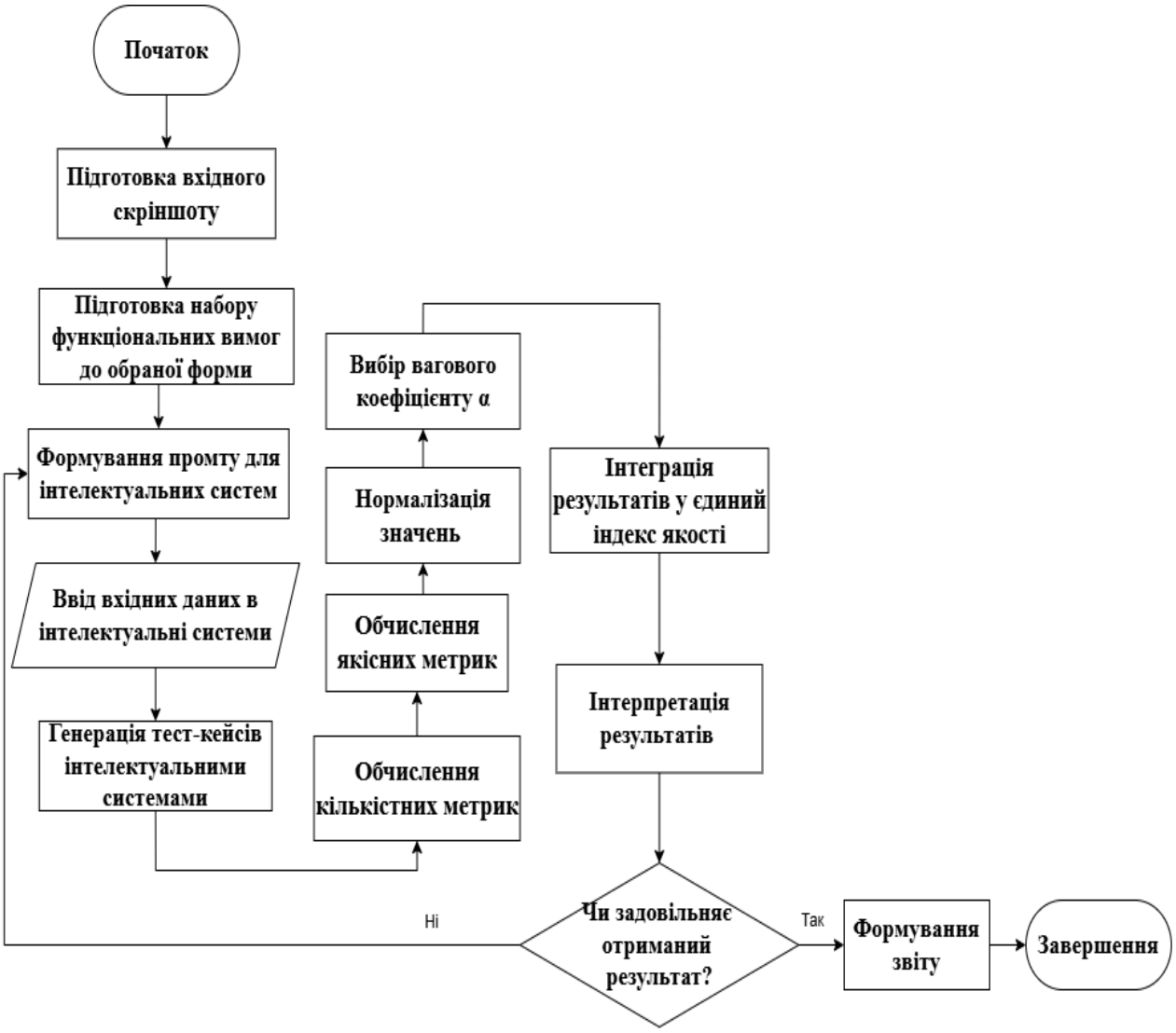


Рис. 3.1 Блок схема алгоритму оцінювання якості та оптимізації роботи інтелектуальних систем

3.5 Практичне застосування комплексної методики

Для опробування розробленого методу було обрано три інтелектуальні системи Google Gemini, OpenAI ChatGPT, Microsoft Copilot

Як вхідні дані з відкритого джерела було взято екрану форму сторінки логіну, що представлена на рисунку 3.2.

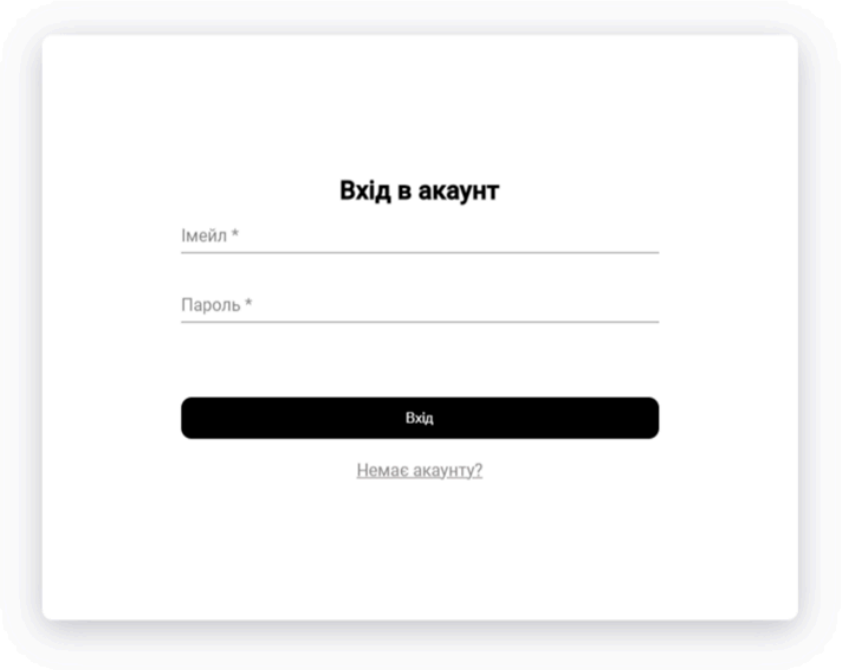


Рис. 3.2 Вхідна екранна форму сторінки логіну

Також, з того ж джерела було взято функціональні вимоги до цієї форми:

1. Поле “Імейл” повинно дозволяти введення символів відповідно до стандартного формату електронної адреси (унікальне_ім'я_користувача@домен).
2. Поле “Імейл” повинно відображати введений текст у звичайному форматі.
3. Поле “Імейл” повинно бути обов’язковим для заповнення.
4. Якщо поле “Імейл” залишене порожнім, система повинна відобразити повідомлення “Введіть Emai” про необхідність його заповнення.
5. Якщо введене значення у полі “Імейл” не відповідає формату електронної адреси, система повинна відобразити повідомлення “Неправильний формат імейлу” про неправильний формат.
6. Система повинна блокувати виконання спроби входу у разі некорректного формату “Імейл”.

7. Поле “Пароль” повинно дозволяти введення будь-яких доступних символів у прихованому вигляді.

8. Поле “Пароль” повинно бути обов’язковим для заповнення.

9. Якщо поле “Пароль” залишене порожнім, система повинна відобразити повідомлення “Введіть пароль” про необхідність його заповнення.

10. Поле “Пароль” повинно приховувати введені символи за допомогою маскування (●●●●).

11. Система повинна блокувати виконання спроби входу у разі порожнього поля “Пароль”.

12. Кнопка “Вхід” повинна бути доступною для натискання завжди, коли форма доступна на екрані.

13. Після натискання кнопки “Вхід” система повинна валідувати значення полів “Імейл” і “Пароль”.

14. Якщо обидва поля заповнені коректно, система повинна надіслати запит авторизації.

15. Під час обробки запиту на кнопці має відображатися індикатор завантаження (спінер), а повторне натискання має бути заблоковане.

16. Якщо хоча б одне поле містить некоректні або порожні дані, система повинна відхилити спробу входу.

17. Якщо введена пара “Імейл” і “Пароль” не відповідає жодному існуючому обліковому запису, система повинна відобразити повідомлення *"Невірний імейл або пароль"* про невірні дані.

18. У разі успішної авторизації система повинна перенаправити користувача на внутрішню сторінку застосунку.

19. Натискання на посилання “Немає акаунту?” повинно перенаправляти користувача на сторінку реєстрації.

20. Перехід за посиланням “Немає акаунту?” не повинен вимагати заповнення будь-яких полів.

21. На формі повинні бути відображені наступні елементи: Заголовок: "Вхід в акаунт" , Поле вводу: "Імейл *" (текстове поле) , Поле вводу: "Пароль *" (текстове поле) , Кнопка: "Вхід" (Primary Button) , Посилання: "Немає акаунту?" (текстове посилання).

22. Форма логіну повинна відображатися без помилок на екрані користувача.

23. Форма логіну повинна зберігати введені користувачем дані до моменту оновлення сторінки або успішного входу.

24. Форма логіну повинна відправляти дані лише після натискання користувачем кнопки “Вхід”.

25. Система повинна відображати повідомлення про помилки без перезавантаження сторінки.

Для експерименту №1 було сформовано наступний промт: “Згенеруй функціональні тест кейси для цієї форми”.

Після введення промту в інтелектуальні системи було отримано функціональні тест-кейси.

Для автоматизації процесів нормалізації, усереднення та обрахунку гібридного індексу якості було розроблено програмний модуль.

Екранна форма розробленого програмного модуля представлена на рисунку 3.3.

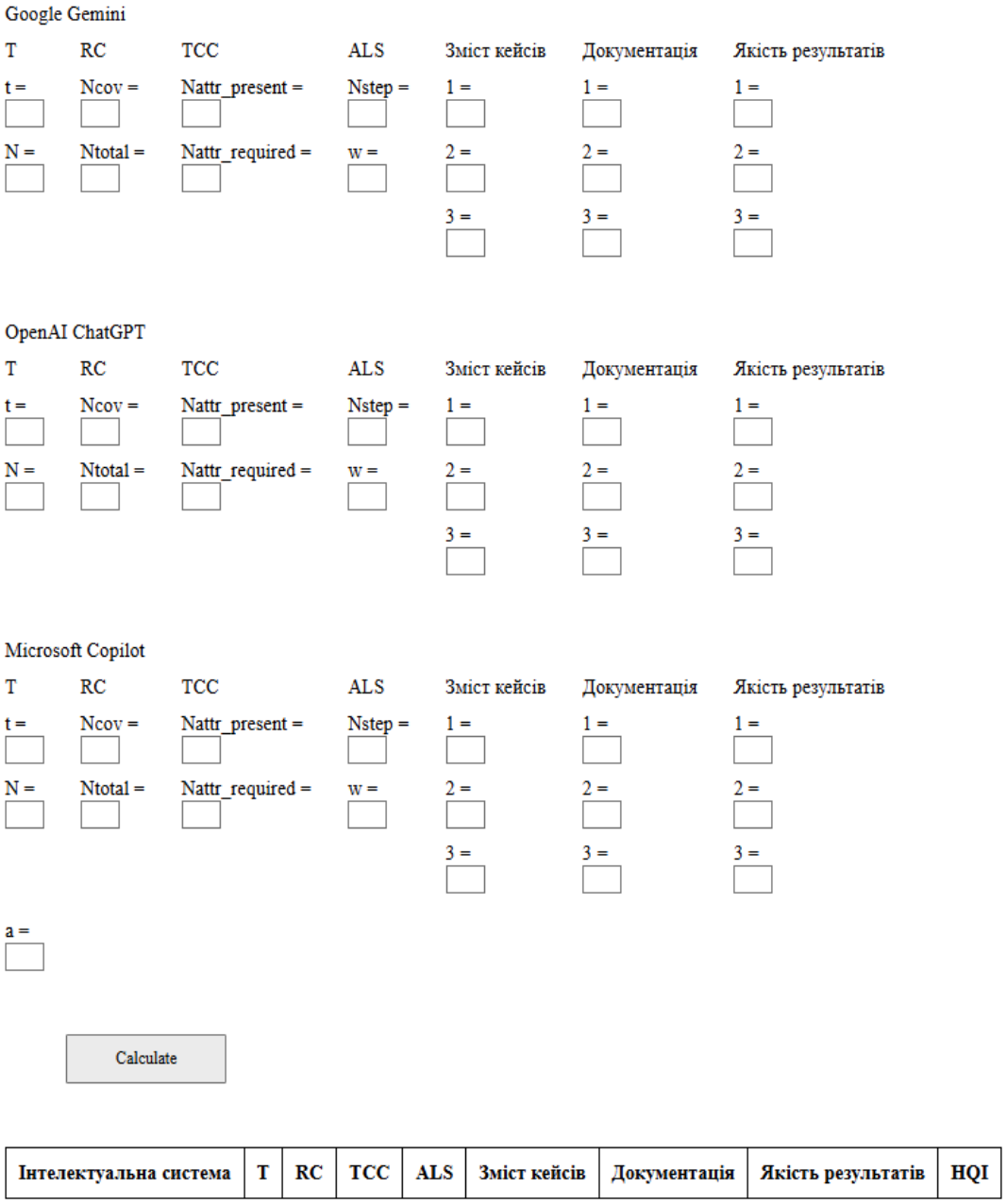


Рис. 3.3 Екранна форма програмного модуля для автоматизації процесів нормалізації, усереднення та обрахунку гібридного індексу якості

Для вагового коефіцієнта α було обрано три значення 0.3, 0.5 та 0.8.

Після підрахунку всіх метрик та їх автоматичної обробки було отримано наступні результати, які наведено в таблиці 3.3.

Таблиця 3.3

Результати експерименту №1

Інтелектуальна система	**T**	**RC**	**TCC**	**ALS**	**Зміст тест-кейсів**	**Докумен-тація**	**Якість результатів**	**HQI**
α = 0.3								
Google Gemini	1.00	0.32	0.83	0.86	0.58	0.83	0.67	0,71
OpenAI ChatGPT	0.00	0.48	0.67	0.00	0.42	0.67	0.42	0,44
Microsoft Copilot	0.53	0.28	0.67	1.00	0.58	0.75	0.50	0,61
α = 0.5								
Google Gemini	1.00	0.32	0.83	0.86	0.58	0.83	0.67	0,72
OpenAI ChatGPT	0.00	0.48	0.67	0.00	0.42	0.67	0.42	0,39
Microsoft Copilot	0.53	0.28	0.67	1.00	0.58	0.75	0.50	0,61
α = 0.8								
Google Gemini	1.00	0.32	0.83	0.86	0.58	0.83	0.67	0,74
OpenAI ChatGPT	0.00	0.48	0.67	0.00	0.42	0.67	0.42	0,33
Microsoft Copilot	0.53	0.28	0.67	1.00	0.58	0.75	0.50	0,62

Для прикладу розберемо результати роботи Google Gemini:

Кількісні метрики:

Час генерації тестів: 1.00

Відсоток покриття вимог: 0.32

Повнота тест-кейсу: 0.83

Середня довжина описів кроків: 0.86

Середнє значення кількісних метрик:

Q_{auto}= 0.75

Якісні метрики:

Зміст тест-кейсів: 0.58

Повнота документації: 0.83

Якість результатів: 0.67

Середнє значення якісних метрик:

$Q_{exp} = 0.69$

За умови, що $\alpha = 0.5$, гібридний індекс якості дорівнює:

$HQI = 0.5 \times 0.75 + 0.5 \times 0.69 = 0.72$

Отже, гібридний індекс якості системи = 0.72, або 72%, що свідчить про середній рівень як технічної, так і когнітивної якості.

-Система генерує 12 тест-кейсів із середнім покриттям вимог 32%.

-Зміст тест-кейсів було оцінено на 0.58, повноту документації - на 0.83, та якість результатів на 0.67.

-За умови, що $\alpha = 0.3$ та 0.8, гібридний індекс якості дорівнює 0.71 та 0.74, що свідчить про перевагу оцінки кількісних метрик над якісними.

Отже, рівень якості системи оцінено як середній, із рекомендацією покращити семантичну точність описів кроків і узгодженість з вимогами.

Для експерименту №2 було додано уточнюючі фрази та сформовано наступний промт:

“Згенеруй детальні функціональні тест кейси для цієї форми, на цій формі є такі елементи:

поле “Імейл”,

поле “Пароль”,

кпопка “Вхід” та посилання “немає аккаунту””.

Після підрахунку всіх метрик та їх автоматичної обробки було отримано наступні результати, які наведено в таблиці 3.4.

Таблиця 3.4

Результати експерименту №2

Інтелектуальна система	**T**	**RC**	**TCC**	**ALS**	**Зміст тест-кейсів**	**Докумен-тація**	**Якість результатів**	**HQI**
α = 0.3								
Google Gemini	1.00	0.52	0.83	0.00	0.75	0.75	0.92	0,74
OpenAI ChatGPT	0.00	0.48	0.83	0.40	0.67	0.58	0.50	0,54
Microsoft Copilot	0.78	0.52	0.67	1.00	0.67	0.83	0.67	0,73
α = 0.5								
Google Gemini	1.00	0.52	0.83	0.00	0.75	0.75	0.92	0,70
OpenAI ChatGPT	0.00	0.48	0.83	0.40	0.67	0.58	0.50	0,51
Microsoft Copilot	0.78	0.52	0.67	1.00	0.67	0.83	0.67	0,73
α = 0.8								
Google Gemini	1.00	0.52	0.83	0.00	0.75	0.75	0.92	0,63
OpenAI ChatGPT	0.00	0.48	0.83	0.40	0.67	0.58	0.50	0,46
Microsoft Copilot	0.78	0.52	0.67	1.00	0.67	0.83	0.67	0,74

Для експерименту №3 було додано уточнюючі фрази та сформовано наступний промт:

“Згенеруй детальні функціональні тест кейси для цієї форми за такими вимогами *і далі перелік вимог*, на цій формі є такі елементи:

поле “Імейл”,

поле “Пароль”,

кпопка “Вхід” та посилання “немає аккаунту””.

Після підрахунку всіх метрик та їх автоматичної обробки було отримано наступні результати, які наведено в таблиці 3.5.

Таблиця 3.5

Результати експерименту №3

Інтелектуальна система	**T**	**RC**	**TCC**	**ALS**	**Зміст тест-кейсів**	**Докумен-тація**	**Якість результатів**	**HQI**
α = 0.3								
Google Gemini	1.00	0.60	1.00	1.00	0.92	0.92	0.75	0,87
OpenAI ChatGPT	0.00	0.64	0.83	0.40	0.75	0.92	0.67	0,68
Microsoft Copilot	0.81	0.60	0.83	0.00	0.67	0.92	0.83	0,73
α = 0.5								
Google Gemini	1.00	0.60	1.00	1.00	0.92	0.92	0.75	0,88
OpenAI ChatGPT	0.00	0.64	0.83	0.40	0.75	0.92	0.67	0,62
Microsoft Copilot	0.81	0.60	0.83	0.00	0.67	0.92	0.83	0,68
α = 0.8								
Google Gemini	1.00	0.60	1.00	1.00	0.92	0.92	0.75	0,89
OpenAI ChatGPT	0.00	0.64	0.83	0.40	0.75	0.92	0.67	0,53
Microsoft Copilot	0.81	0.60	0.83	0.00	0.67	0.92	0.83	0,61

3.6 Порівняльний аналіз результатів

На основі отриманих результатів експерименту №1 видно чітку ієрархію систем, яка зберігається незалежно від значення α, хоча розрив між системами змінюється.

Аналіз кількісних метрик виявляє значний розрив між системами. Беззаперечним лідером є Google Gemini з Q_{auto} = 0.75, що пояснюється найвищим показником за часом генерації (T = 1.00), високою повнотою тест-кейсів (TCC = 0.83) та значною середньою довжиною описів кроків (ALS = 0.86). Це свідчить про високу технічну продуктивність і швидкість системи.

Microsoft Copilot посідає друге місце з Q_{auto} = 0.62. Його сильна сторона – максимальна середня довжина описів кроків (ALS = 1.00), що може вказувати на більш деталізовані інструкції, проте його швидкість (T=0.53) та покриття вимог (RC=0.28) є нижчими, ніж у Google Gemini.

Система OpenAI ChatGPT демонструє критично низьке значення Q_{auto} = 0.28. Це зумовлено найнижчими показниками за часом генерації (T = 0.00) та середньою довжиною описів кроків (ALS = 0.00). При цьому її показник відсоток покриття вимог (RC = 0.48) є найвищим серед усіх систем.

У сфері якісних метрик, різниця між системами є менш великою, але тенденція домінування зберігається. Google Gemini знову має найвищий бал Q_{exp} = 0.69, демонструючи збалансовану високу оцінку за всіма трьома параметрами (Зміст кейсів, Документація, Якість результатів).

Microsoft Copilot йде слідом із Q_{exp} = 0.61, показуючи високу повноту документації (0.75). Це значення лише трохи відстає від лідера.

OpenAI ChatGPT має Q_{exp} = 0.50, що, хоча й є найнижчим показником, проте значно перевищує його власний Q_{auto} (0.28). Це означає, що, незважаючи на технічні недоліки, згенерований контент та документація цієї системи є задовільними.

Порівняння гібридного індексу якості (HQI) за трьох різних значень вагового коефіцієнта показує, як змінюється загальна оцінка залежно від пріоритету кількісних метрик.

Google Gemini стабільно є лідером у всіх сценаріях (HQI від 0.71 до 0.74). Оскільки Q_{auto} системи (0.75) вищий, ніж Q_{exp} (0.69), збільшення α (тобто надання більшої ваги кількісним показникам) призводить до послідовного зростання його HQI (від 0.71 при α=0.3 до 0.74 при α=0.8).

OpenAI ChatGPT демонструє найбільшу чутливість до зміни α. Його HQI знаходиться в діапазоні від 0.44 до 0.33. Через критично низький Q_{auto}, збільшення вагового коефіцієнта α призводить до різкого зниження його загальної оцінки, що підкреслює його суттєву залежність від експертної оцінки.

Microsoft Copilot демонструє найбільшу стійкість. Оскільки його Q_{auto} (0.62) та Q_{exp} (0.61) майже рівні, його HQI залишається практично незмінним (0.61 при $\alpha = 0.3$ та $\alpha = 0.5$; 0.62 при $\alpha = 0.8$), що свідчить про збалансованість його продуктивності як з технічної, так і з якісної точок зору.

Google Gemini є найбільш ефективною системою для генерації тест-кейсів за цими метриками, перевершуючи конкурентів як за швидкістю та повнотою (що є критичним для автоматизації), так і за експертною оцінкою.

ChatGPT намагався охопити найбільше (RC=0.48), але зробив це "поверхнево" (низькі бали за зміст і структуру). Gemini та Copilot сфокусувалися на меншій кількості вимог, але прописали їх значно якісніше.

Рекомендації:

Для швидкої генерації готових тестів: Однозначно Google Gemini. Найвищий HQI та збалансовані метрики.

Для детальних інструкцій: Microsoft Copilot (завдяки ALS=1.00).

Для мозкового штурму вимог: Можна використати ChatGPT, оскільки він знаходить більше сценаріїв покриття (RC=0.48), але їх доведеться переписувати вручну.

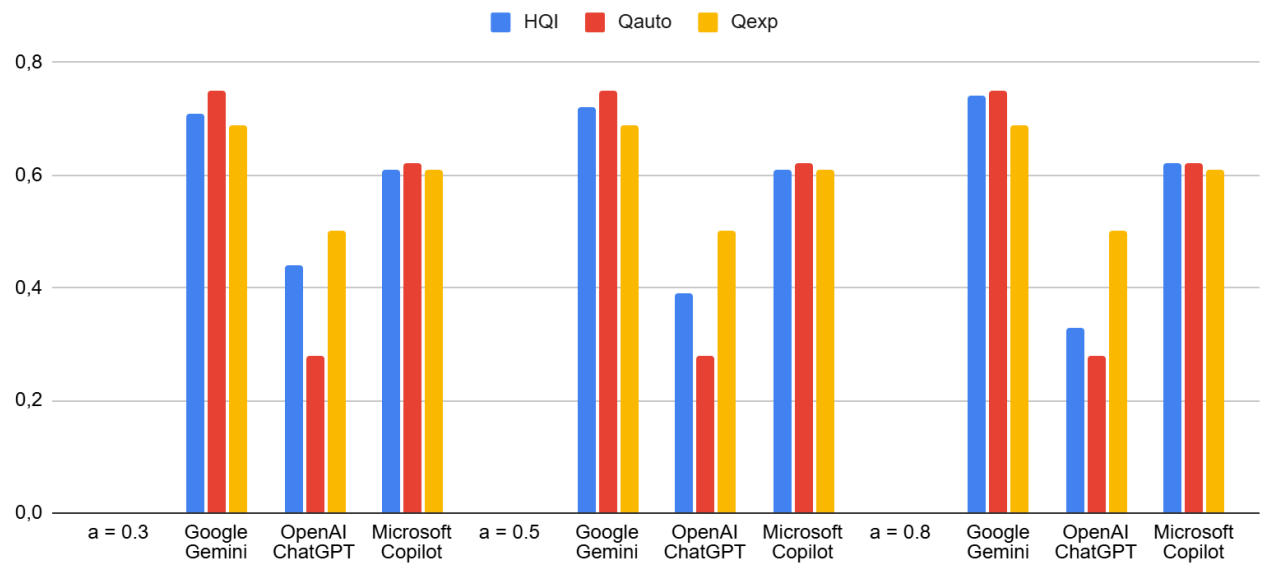


Рис. 3.4 Порівняння гібридного індексу якості, середнього значення кількісних та якісних метрик результатів генерації тестової документації в експерименті №1

В експерименті №1 беззаперечним лідером була Google Gemini з найвищим HQI у всіх вагових сценаріях (α від 0.71 до 0.74). Однак, в експерименті №2 лідерство переходить до Microsoft Copilot при високому пріоритеті кількісних метрик, тоді як Google Gemini зберігає лідерство лише при низькому α завдяки своїй високій якісній оцінці. Система OpenAI ChatGPT залишається найменш ефективною в обох наборах даних, незважаючи на незначне загальне покращення.

В експерименті №1 система Google Gemini мала домінуючий Q_{auto} (0.75) над Q_{exp} (0.69). В експерименті №2 ситуація кардинально змінюється: Q_{auto} різко падає до 0.59, тоді як Q_{exp} значно зростає до 0.80. Це падіння кількісної оцінки спричинене критичним показником середньої довжини описів кроків (ALS = 0.00), що вказує на відсутність деталізації в новому сценарії. Як наслідок, чутливість HQI змінюється на протилежну: якщо раніше HQI зростав зі збільшенням α, то в експерименті №2 HQI падає (з 0.74 до 0.63) при збільшенні

ваги Q_{auto} (0.75), що підкреслює залежність системи від високої експертної оцінки в цьому новому контексті.

Microsoft Copilot демонструє найбільш суттєве та послідовне покращення. в експерименті №2 спостерігається значне зростання як Q_{auto} (з 0.62 до 0.74), так і Q_{exp} (з 0.61 до 0.72). Це підвищення Q_{auto}, яке включає максимальний показник ALS = 1.00 в обох наборах, робить Copilot найбільш ефективною системою при $\alpha = 0.8$, де його HQI становить 0.74. Це свідчить про те, що Copilot продемонстрував високу стабільність та значне покращення загальної продуктивності, особливо коли технічні метрики мають пріоритет.

Система OpenAI ChatGPT демонструє загальне підвищення обох середніх метрик у Експерименті №2 (Q_{auto} з 0.28 до 0.43, Q_{exp} з 0.50 до 0.58). Покращення Q_{auto} стало можливим завдяки зростанню ALS з 0.00 до 0.40, що усунуло один із критичних недоліків попереднього сценарію. Проте, критичний нульовий показник часу генерації (T = 0.00) зберігається в обох експериментах, що є незмінною слабкістю, яка сильно обмежує його HQI. У результаті, HQI цієї системи залишається найнижчим і також падає зі зростанням α (з 0.54 до 0.46), оскільки кількісна складова залишається її головною проблемою.

В експерименті №2 Google Gemini, OpenAI ChatGPT та Microsoft показують зростання оцінки гібридного індексу якості, робота систем була оптимізована на 2%, 12% та 12% відповідно (при $\alpha = 0.5$). Google Gemini показує зростання оцінки якісних метрик, але невелике падіння оцінки кількісних, а OpenAI ChatGPT та Microsoft Copilot мають значне зростання оцінки кількісних метрик. Найсильнішою з систем в експерименті №2 стає Microsoft Copilot, який при рівновазі метрик $\alpha = 0.5$ має HQI 0.73, а також лідирує при пріоритеті кількісних показників.

Загалом, Microsoft Copilot виявляється найбільш збалансованою та стабільною системою при зміні експериментальних умов, забезпечуючи високий Q_{auto} та Q_{exp} у другому експерименті.

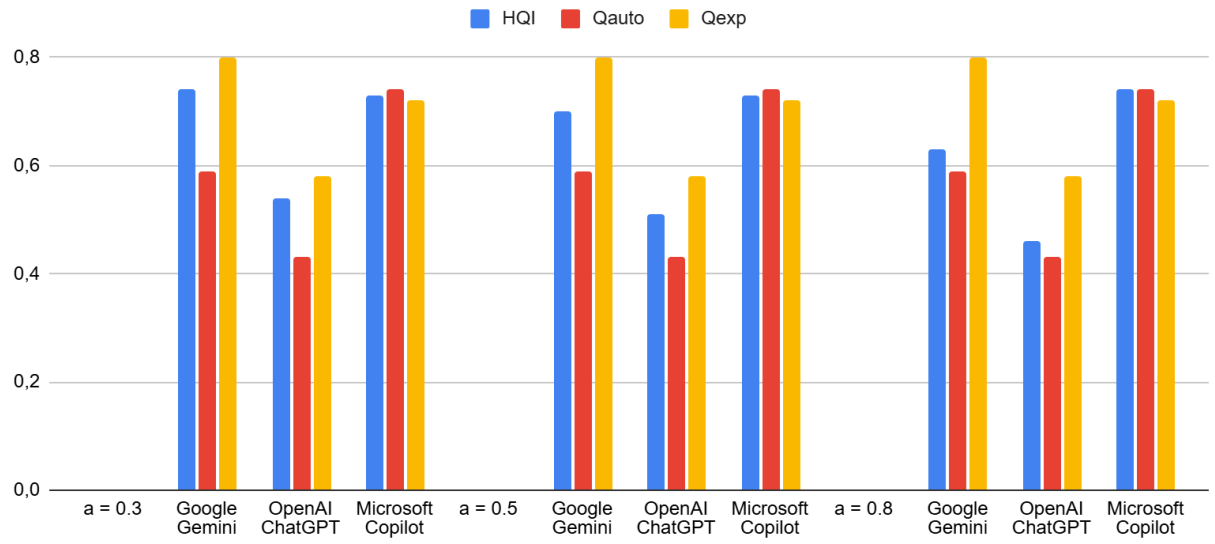


Рис. 3.5 Порівняння гібридного індексу якості, середнього значення кількісних та якісних метрик результатів генерації тестової документації в експерименті №2

Google Gemini демонструє найвищу стабільність у часі генерації (T=1.00) у всіх трьох експериментах, що є його ключовою перевагою.

В експерименті №3 Google Gemini досягає піку продуктивності: Q_{auto} становить 0.90, що є найвищим показником серед усіх систем у всіх сценаріях, завдяки максимальній оцінці метрик T, TCC, ALS. Це дозволяє системі стати беззаперечним лідером за HQI (від 0.87 до 0.89) незалежно від вагового коефіцієнта α. Зростання α у цьому сценарії призводить до збільшення HQI, підкреслюючи, що в Експерименті №3 її кількісна ефективність перевищила якісну (Q_{auto}= 0.90$ проти Q_{exp}= 0.86$).

Microsoft Copilot демонструє найбільш збалансовану та стабільну роботу щодо ALS, де його показник становить 1.00 у Експериментах №1 та №2, однак, в експерименті №3 Q_{auto} Microsoft Copilot різко знижується до 0.56, зокрема через падіння ALS до 0.00. Це спричиняє значне зниження його HQI (з 0.73 до 0.61) при зростанні α, оскільки кількісна складова стала його найбільшою слабкістю в цьому сценарії, попри високу якісну оцінку Q_{exp} = 0.80.

OpenAI ChatGPT є найменш ефективною системою у всіх трьох експериментах. Головною і незмінною критичною проблемою є найнижчий показник середнього часу генерації тест кейсів (T = 0.00) в усіх експериментах.

В експерименті №1 OpenAI ChatGPT мала критично низький Q_{auto} = 0.28. В експерименті №2 спостерігалося незначне покращення Q_{auto} до 0.43. В експерименті №3 Q_{auto} залишається на низькому рівні 0.45, що в поєднанні з найвищим показником покриття вимог (RC = 0.64) в даному експерименті вказує на те, що система може якісно охоплювати вимоги, але її технічна реалізація (час і деталізація кроків) залишається проблематичною. В усіх випадках HQI системи падає зі збільшенням α, оскільки переважна вага надається її слабкій кількісній метриці Q_{auto}.

Google Gemini домінує у більшості сценаріїв, а в експерименті №3 демонструє найкращий загальний результат (HQI = 0.89).

В експерименті №3 Google Gemini та OpenAI ChatGPT показують ще більше зростання оцінки гібридного індекс якості, робота систем була оптимізована на 18% та 11% відповідно (при α = 0.5).

Microsoft Copilot показав найбільший стрибок ефективності та лідерство в Експерименті №2, але його продуктивність суттєво залежить від умов експерименту, що підтверджується падінням його Q_{auto} в Експерименті №3.

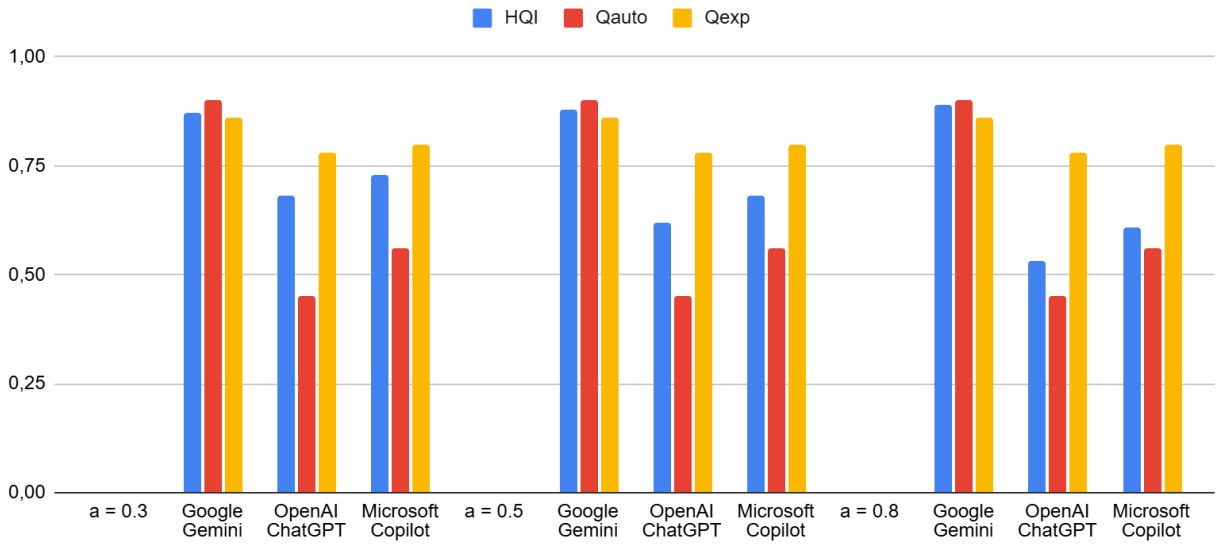


Рис. 3.6 Порівняння гібридного індексу якості, середнього значення кількісних та якісних метрик результатів генерації тестової документації в експерименті №3

У всіх трьох експериментах HQI OpenAI ChatGPT знижується при збільшенні α. HQI Google Gemini зростає у Експериментах №1 та №3 (де $Q_{auto} > Q_{exp}$) і падає у Експерименті №2 (де $Q_{exp} > Q_{auto}$). Це демонструє, що HQI є ефективним інструментом для визначення пріоритетів оптимізації: При формуванні промту для системи з низьким Q_{auto} (наприклад, OpenAI ChatGPT) потрібно зосередитися на технічних аспектах тест-кейсів, тоді як системи з високим Q_{exp} (наприклад, Google Gemini в Експерименті №2) можуть підвищити загальну оцінку, працюючи над кількісними параметрами.

ВИСНОВКИ

У результаті проведеного дослідження та розробки досягнуто поставленої мети – створено комплексну методику оцінювання якості та оптимізації інтелектуальних систем для автоматизації тестування та генерації тестової документації.

Проаналізовано предметну галузь. Існуючі інтелектуальні системи вирізняються значним різноманіттям, пропонуючи інструменти для великої кількості процесів тестування програмного забезпечення. Такий широкий спектр рішень дає змогу обирати найбільш оптимальні інструменти під конкретні потреби проєкту, поєднуючи їх для досягнення максимальної ефективності тестування.

Проаналізовано сучасні методи оцінки згенерованої документації, такі як експертні, автоматичні та нейромережеві моделі. Визначено їхні ключові недоліки: висока собівартість, низька швидкість, нормативно-орієнтоване оцінювання, формалістична оцінка, можлива упередженість, можливі галюцинації системи

Визначено чотири кількісних та три якісних метрики та сформовано комплексну методику оцінювання якості та оптимізації генерації тест кейсів, що поєднує кількісні та якісні метрики. Запропонована модель включає оцінку точності, структури, стилю, та відповідності вимогам, що забезпечує поєднання обчислювальної об’єктивності та когнітивної інтерпретації, створюючи прозорий і гнучкий механізм оцінювання інтелектуальних систем, що дозволяє не лише кількісно оцінити рівень ефективності системи, а й визначити напрямки для її оптимізації.

Проведено експериментальне дослідження. Згідно експерименту №1 при однакових вхідних даних Google Gemini забезпечує вищу якість генерації тест кейсів, тоді як OpenAI ChatGPT та Microsoft Copilot поступаються за якістю.

Після аналізу даних експерименту №1, було вирішено оптимізувати роботу інтелектуальних систем, додавши більш уточнюючі формулювання у промти. В експерименті №2 Google Gemini, OpenAI ChatGPT та Microsoft показують зростання оцінки гібридного індексу якості, робота систем була оптимізована на 2%, 12% та 12% відповідно (при $\alpha = 0.5$). Google Gemini показує зростання оцінки якісних метрик, але невелике падіння оцінки кількісних, а OpenAI ChatGPT та Microsoft Copilot мають значне зростання оцінки кількісних метрик. В експерименті №3 Google Gemini та OpenAI ChatGPT показують ще більше зростання оцінки гібридного індекс якості, робота систем була оптимізована на 18% та 11% відповідно (при $\alpha = 0.5$), за рахунок значного зростання оцінки кількісних та якісних метрик. В усіх трьох експериментах Google Gemini показує найбільшу оцінку гібридного індекс якості з трьох експериментів.

Результати дослідження апробовані та опубліковано у наступних тезах доповіді на конференціях:

1. Сазонов С.О., Яскевич В.О. Використання штучного інтелекту в життєвому циклі розробки програмного забезпечення. VI Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT», 15 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 110-112.

2. Сазонов С.О., Яскевич В.О. Штучний інтелект у прогнозуванні ризиків та пріоритизації тестування програмного забезпечення. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ», 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 313-316.

ПЕРЕЛІК ПОСИЛАНЬ

1. Сазонов С.О., Яскевич В.О. Використання штучного інтелекту в життєвому циклі розробки програмного забезпечення. VI Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT», 15 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 110-112.
2. Сазонов С.О., Яскевич В.О. Штучний інтелект у прогнозуванні ризиків та пріоритизації тестування програмного забезпечення. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ», 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 313-316.
3. Витвицький Р., Якубовський В. Використання штучного інтелекту та машинного навчання для автоматизації процесів тестування програмного забезпечення в Україні. *Herald of Khmelnytskyi National University. Technical Sciences*. 2024. № 345(6(2)). С. 21–27. DOI: https://doi.org/10.31891/2307-5732-2024-345-6-2
4. Гуральник Ф. Application of Artificial Intelligence for Software Testing Automation. *SWorldJournal*. 2024. 1(28-01). С. 71–88. DOI: https://doi.org/10.30888/2663-5712.2024-28-00-042
5. Онищенко Р., Котенко Н., Жирова Т. Роль та ефективність засобів штучного інтелекту в тестуванні програмного забезпечення. *Інформаційні технології та суспільство*. 2024. (2(13)). С. 66–70. DOI: https://doi.org/10.32689/maup.it.2024.2.10
6. Hunko I. Software Testing in 2023: New Trends and Challenges. *Herald of Kyiv Institute of Business and Technology*. 2023. 49(1–2). С. 25–36. DOI: https://doi.org/10.37203/kibit.2023.49.03

7. Муляревич О.В., Боярінова Ю.Є. Аналіз ефективності автоматизованого та ручного тестування в забезпеченні якості програмного забезпечення. *Ukrainian Journal of Computing Innovations*. 2024. (1). DOI: https://doi.org/10.5281/zenodo.14066456
8. DigitalOcean. What Is AI Software Testing? Improving Quality Assurance with Artificial Intelligence. [Електронний ресурс]. URL: https://www.digitalocean.com/resources/articles/ai-software-testing (дата звернення: 11.11.2025).
9. LambdaTest. Machine Learning in Software Testing. [Електронний ресурс]. URL: https://www.lambdatest.com/blog/machine-learning-in-software-testing/ (дата звернення: 11.11.2025).
10. Mabl – Official documentation. [Електронний ресурс]. URL: https://help.mabl.com/ (дата звернення: 11.11.2025).
11. Applitools Eyes – Official documentation. [Електронний ресурс]. URL: https://applitools.com/docs/ (дата звернення: 11.11.2025).
12. Diffblue Cover – Official documentation. [Електронний ресурс]. URL: https://docs.diffblue.com (дата звернення: 11.11.2025).
13. Parasoft – Official documentation. [Електронний ресурс]. URL: https://www.parasoft.com (дата звернення: 11.11.2025).
14. OpenAI ChatGPT – Official documentation. [Електронний ресурс]. URL: https://platform.openai.com/docs/overview (дата звернення: 11.11.2025).
15. Google Gemini – Official documentation. [Електронний ресурс]. URL: https://ai.google.dev/gemini-api/docs (дата звернення: 11.11.2025).
16. Microsoft Copilot – Official documentation. [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/microsoft-copilot-studio (дата звернення: 11.11.2025).
17. The CTO Club. AI in Software Testing. [Електронний ресурс]. URL: https://thectoclub.com/ai-ml/ai-in-software-testing/ (дата звернення: 11.11.2025).

18. Palo Alto Networks. What Is AI Governance? [Електронний ресурс]. URL: https://www.paloaltonetworks.com/cyberpedia/ai-governance (дата звернення: 11.11.2025).
19. NearForm. A 5-Stage Guide to Turbocharging the Quality Assurance Process with Generative AI. [Електронний ресурс]. URL: https://www.nearform.com/digital-community/a-5-stage-guide-to-turbocharging-the-quality-assurance-process-with-generative-ai/ (дата звернення: 11.11.2025).
20. Barbella M., Tortora G. Rouge Metric Evaluation for Text Summarization Techniques. DOI: http://dx.doi.org/10.2139/ssrn.4120317.
21. Hanna M., Bojar O. A Fine-Grained Analysis of BERTScore. In: *Proceedings of the Sixth Conference on Machine Translation*. 2021. С. 507–517. URL: https://aclanthology.org/2021.wmt-1.59/.
22. Akter M., Bansal N., Karmaker S.K. Revisiting Automatic Evaluation of Extractive Summarization Task: Can We Do Better than ROUGE? *Findings of the Association for Computational Linguistics: ACL 2022*. 2022. С. 1547–1560. URL: https://aclanthology.org/2022.findings-acl.122/.
23. Kaster M., Zhao W., Eger S. Global Explainability of BERT-Based Evaluation Metrics by Disentangling along Linguistic Factors. *EMNLP 2021 Proceedings*. 2021. С. 8912–8925. URL: https://aclanthology.org/2021.emnlp-main.701/ .
24. ISO/IEC/IEEE 29119-1:2022. Software and systems engineering – Software testing – Part 1: General concepts. Geneva: ISO, 2022. URL: https://www.iso.org/standard/81291.html (дата звернення: 11.11.2025).
25. Nguyen H. et al. A Comparative Study of Quality Evaluation Methods for Text Summarization. 2024. URL: https://arxiv.org/abs/2407.00747 (дата звернення: 11.11.2025).
26. Zhang et al. BERTScore: Evaluating Text Generation with BERT. 2020. DOI: https://doi.org/10.48550/arXiv.1904.09675 .

27. Human in the Loop: Ensuring Accuracy and Ethics in AI Systems. OpenXcell. [Електронний ресурс]. URL: https://www.openxcell.com/blog/human-in-the-loop/ (дата звернення: 11.11.2025).
28. Human in the Loop: 101 Guide With Examples. SPD Technology. [Електронний ресурс]. URL: https://spd.tech/artificial-intelligence/human-in-the-loop/ (дата звернення: 11.11.2025).
29. Amershi S. et al. Guidelines for Human-AI Interaction. *AI Magazine*. 2014. С. 105–120. DOI: https://doi.org/10.1609/aimag.v35i4.2513 .
30. Terragni V. LLMLOOP: Improving LLM-Generated Code and Tests through Automated Iterative Feedback Loops. 2025. [Електронний ресурс]. URL: https://valerio-terragni.github.io/assets/pdf/ravi-icsme-2025.pdf (дата звернення: 11.11.2025).
31. On the Limitations of Embedding Based Methods for Measuring Functional Correctness for Code Generation. 2024. URL: https://arxiv.org/html/2405.01580v1 (дата звернення: 11.11.2025).
32. Аналіз переваг та недоліків генерації коду із використанням великих мовних моделей в сучасних IDE. 2025. DOI: https://doi.org/10.35546/kntu2078-4481.2025.2.2.36 (дата звернення: 11.11.2025).
33. ISO/IEC/IEEE 29119-3:2021. Software and systems engineering – Software testing – Part 3: Test documentation. Geneva: ISO, 2021. URL: https://www.iso.org/standard/79429.html (дата звернення: 11.11.2025).
34. Freitag M. et al. Experts, Errors, and Context: A Large-Scale Study of Human Evaluation for Machine Translation. 2021. DOI: https://doi.org/10.48550/arXiv.2107.10821 (дата звернення: 11.11.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Методика оцінювання якості та оптимізації інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації»

Виконав: студент групи ПДМ-62 Сергій САЗОНОВ

Керівник: канд. техн. наук, доцент, доцент кафедри ІПЗ Владислав ЯСКЕВИЧ

Київ - 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: покращення процесу оцінювання якості та оптимізації використання інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації.

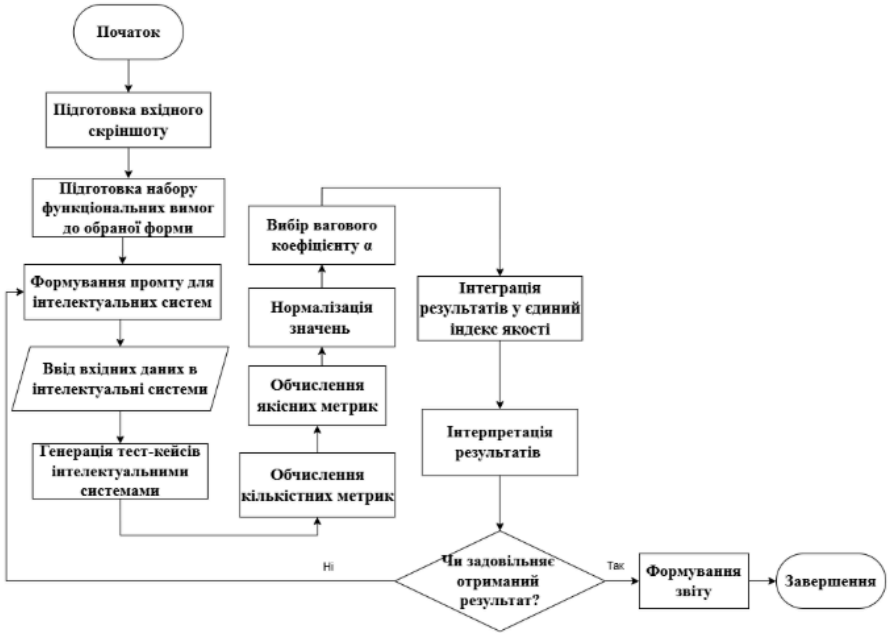
Об'єкт дослідження: процес оцінювання якості та оптимізації використання інтелектуальних систем для автоматизації тестування програмного забезпечення та генерації супровідної тестової документації.

Предмет дослідження: метод оцінювання якості та оптимізації використання інтелектуальних систем для автоматизації процесів тестування програмного забезпечення та генерації супровідної тестової документації.

АКТУАЛЬНІСТЬ РОБОТИ

Методика	Можливості	Недоліки
Експертна оцінка	Висока точність; Перевірка логіки; Врахування контексту.	Висока собівартість; Низька швидкість; Емпірична варіативність; Персоналізована інтерпретація.
Автоматичні метрики (ROUGE/BERTScore)	Висока швидкість; Структурований результат; Автономно.	Формалістична оцінка; Не оцінює логіку; Нормативно-орієнтоване оцінювання.
LLM-as-a-Judge	Баланс швидкості та змістовності; Здатність виявляти логічні помилки.	Можлива упередженість; Можливі галюцинації системи.
Комплексна методика оцінювання	Багатовимірна оцінка якості Виявлення логічних та концептуальних помилок Зменшення суб'єктивності Підтримка ітеративного покращення генерації	Залишкова залежність від експертів

АЛГОРИТМ ОЦІНЮВАННЯ ЯКОСТІ ЗГЕНЕРОВАНОЇ ДОКУМЕНТАЦІЇ



КІЛЬКІСНІ МЕТРИКИ

$$T = \frac{t}{N},$$

де T - середній час генерації одного тест-кейсу;

t - загальна кількість часу витраченого на генерування всіх тест кейсів;

N - кількість згенерованих тест-кейсів.

$$TCC = \frac{N_{attr\_present}}{N_{attr\_required}} \times 100\%,$$

де TCC - повнота тест-кейсу;

$N_{attr\_present}$ - кількість фактично заповнених атрибутів тест-кейсу;

$N_{attr\_required}$ - кількість атрибутів, які повинні бути заповнені згідно зі стандартом.

$$RC = \frac{N_{cov}}{N_{total}} \times 100\%,$$

де RC - відсоток покриття вимог;

N_{cov} - кількість вимог, для яких створено хоча б один тест-кейс;
N_{total} - загальна кількість вимог.

$$ALS = \frac{1}{N_{step}} \sum_{i=1}^{N_{step}} w_i,$$

де ALS - середня довжина описів кроків;

N_{step} - загальна кількість кроків у тест-кейсах;

w_i - кількість слів у і-му кроці.

ЯКІСНІ МЕТРИКИ

1. Зміст тестів

- ☐ Тест-кейси мають зрозумілі назви
- ☐ Кроки описані послідовно
- ☐ Очікувані результати чітко вказано

2. Документація

- ☐ Документи узгоджені між собою
- ☐ Не виявлено дублікатів
- ☐ Використано уніфіковану термінологію

3. Якість результатів

- ☐ Використання неточних, або двозначних термінів
- ☐ Виявлені баги мають практичну значущість
- ☐ Документація зрозуміла без контексту розробника

Шкала оцінювання:

1 — Дуже низька якість.
Тест-кейси мають суттєві недоліки, які унеможливлюють їх коректне використання.

2 — Низька якість.
Є відчутні прогалини та неточності, які заважають стабільному використанню. Тест-кейси потребують значного доопрацювання та уточнення.

3 — Середня якість.
Тест-кейси загалом зрозумілі і виконувані. Недоліки не критичні, але помітні. Застосування можливе без великих труднощів, проте покращення явно підвищить ефективність.

4 — Висока якість.
В тест-кейсах спостерігаються лише дрібні недоліки, що не впливають на використання. Найважливіші частини оформлені коректно і відповідають очікуваним стандартам.

5 — Відмінна якість.
Тест-кейси оформлено максимально чітко, логічно та повністю відповідно до стандартів. Недоліки відсутні або мінімальні й не мають значення.

НОРМАЛІЗАЦІЯ ОТРИМАНИХ ДАНИХ ТА СЕРЕДНІ ПОКАЗНИКИ МЕТРИК

$$Q' = \frac{Q - Q_{min}}{Q_{max} - Q_{min}} \quad ,$$

де Q′ - нормалізоване значення метрики;

Q - фактичне значення метрики;

Q_{min} - мінімальне значення в наборі даних;

Q_{max} - максимальне значення в наборі даних.

$$Q' = \frac{Q_{max} - Q}{Q_{max} - Q_{min}} \quad ,$$

де Q′ - нормалізоване значення метрики;

Q - фактичне значення метрики;

Q_{min} - мінімальне значення в наборі даних;

Q_{max} - максимальне значення в наборі даних.

$$Q_{auto} = \frac{1}{n} \sum_{i=1}^{n} Q'_i \quad ,$$

де Qauto - середній кількісний показник якості;

Q'_i — нормалізоване значення i-ї кількісної метрики;

n - кількість кількісних метрик.

$$Q_{exp} = \frac{1}{m} \sum_{j=1}^{m} Q'_i \quad ,$$

де Q_{exp} - середній якісний показник якості;

Q'_j — нормалізоване значення j-ї якісної метрики;

m - кількість якісних метрик.

7

ФОРМУЛА ГІБРИДНОГО ІНДЕКСУ ЯКОСТІ

$$HQI = \alpha \times Q_{auto} + (1 - \alpha) \times Q_{exp} \quad ,$$

де HQI - гібридний індекс якості

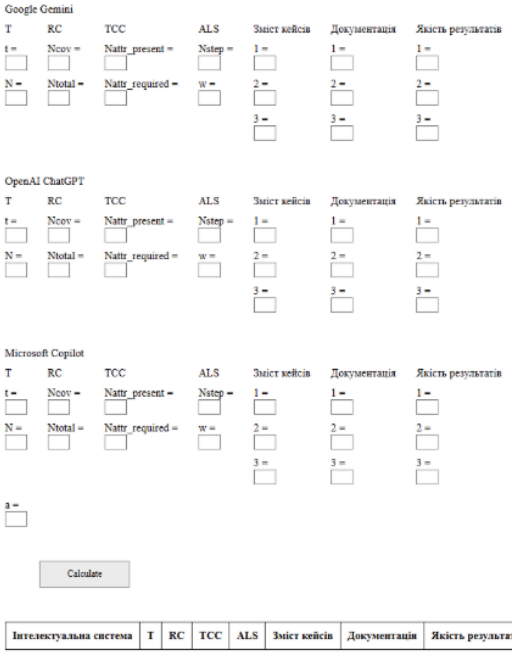
Q_{auto} - середнє значення кількісних метрик (T, RC, TCC, ALS)

Q_{exp} - середнє значення якісних метрик, отриманих на основі експертного чек-листа

α - ваговий коефіцієнт, який визначає відносну важливість кількісних метрик у загальній оцінці ($0 \leq \alpha \leq 1$).

8

ЕКРАНА ФОРМА ПРОГРАМИ ДЛЯ АВТОМАТИЧНОГО РАХУНКУ HQI



9

ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

Експеримент №1

Інтелектуальна система	T	RC	TCC	ALS	Зміст тестів	Документація	Якість результату
Google Gemini	1.00	0.32	0.83	0.86	0.58	0.83	0.67
OpenAI ChatGPT	0.00	0.48	0.67	0.00	0.42	0.67	0.42
Microsoft Copilot	0.53	0.28	0.67	1.00	0.58	0.75	0.50

α = 0.3

Інтелектуальна система	HQI
Google Gemini	0,71
OpenAI ChatGPT	0,44
Microsoft Copilot	0,61

α = 0.5

Інтелектуальна система	HQI
Google Gemini	0,72
OpenAI ChatGPT	0,39
Microsoft Copilot	0,61

α = 0.8

Інтелектуальна система	HQI
Google Gemini	0,74
OpenAI ChatGPT	0,33
Microsoft Copilot	0,62

10

ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

Експеримент №2

Інтелектуальна система	T	RC	TCC	ALS	Зміст тестів	Документація	Якість результату
Google Gemini	1.00	0.52	0.83	0.00	0.75	0.75	0.92
OpenAI ChatGPT	0.00	0.48	0.83	0.40	0.67	0.58	0.50
Microsoft Copilot	0.78	0.52	0.67	1.00	0.67	0.83	0.67

α = 0.3

Інтелектуальна система	HQI
Google Gemini	0,74
OpenAI ChatGPT	0,54
Microsoft Copilot	0,73

α = 0.5

Інтелектуальна система	HQI
Google Gemini	0,70
OpenAI ChatGPT	0,51
Microsoft Copilot	0,73

α = 0.8

Інтелектуальна система	HQI
Google Gemini	0,63
OpenAI ChatGPT	0,46
Microsoft Copilot	0,74

ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

Експеримент №3

Інтелектуальна система	T	RC	TCC	ALS	Зміст тестів	Документація	Якість результату
Google Gemini	1.00	0.60	1.00	1.00	0.92	0.92	0.75
OpenAI ChatGPT	0.00	0.64	0.83	0.40	0.75	0.92	0.67
Microsoft Copilot	0.81	0.60	0.83	0.00	0.67	0.92	0.83

α = 0.3

Інтелектуальна система	HQI
Google Gemini	0,87
OpenAI ChatGPT	0,68
Microsoft Copilot	0,73

α = 0.5

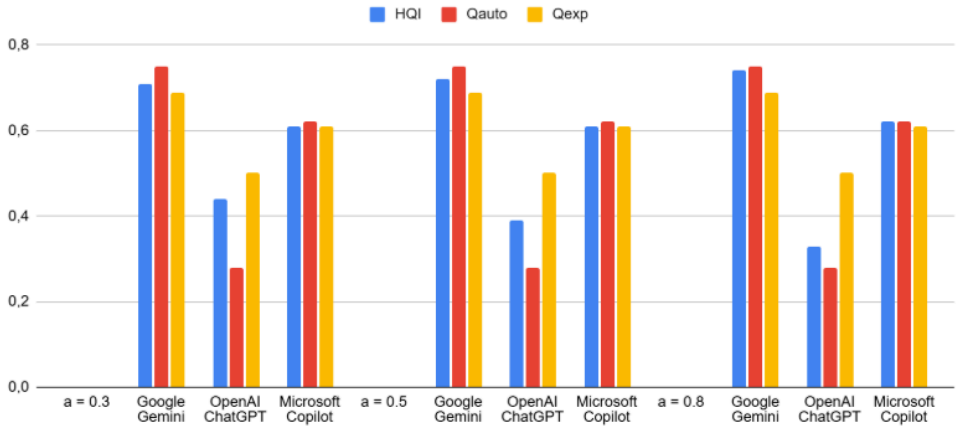
Інтелектуальна система	HQI
Google Gemini	0,88
OpenAI ChatGPT	0,62
Microsoft Copilot	0,68

α = 0.8

Інтелектуальна система	HQI
Google Gemini	0,89
OpenAI ChatGPT	0,53
Microsoft Copilot	0,61

ПОРІВНЯЛЬНА ДІАГРАМА РЕЗУЛЬТАТІВ РОБОТИ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ

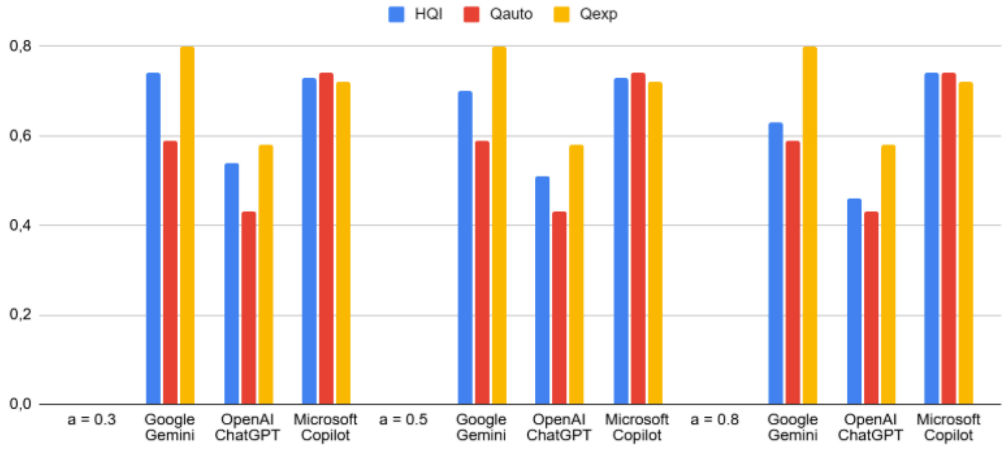
Порівняння гібридного індексу якості, середнього значення кількісних та якісних метрик результатів генерації тестової документації в експерименті №1



13

ПОРІВНЯЛЬНА ДІАГРАМА РЕЗУЛЬТАТІВ РОБОТИ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ

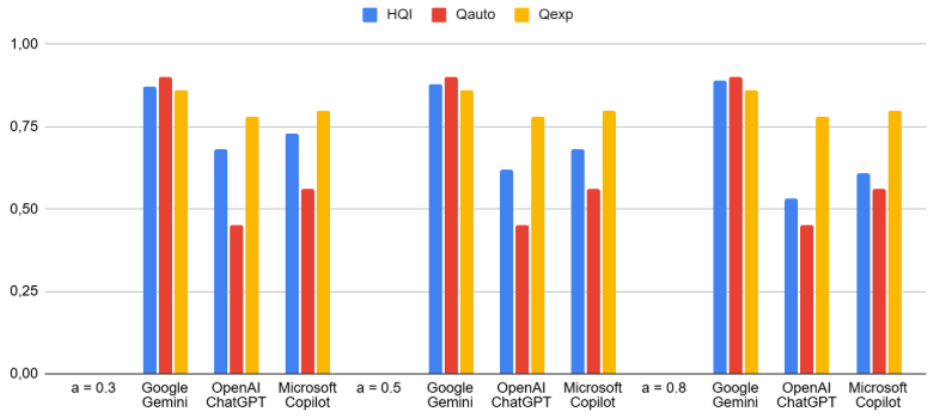
Порівняння гібридного індексу якості, середнього значення кількісних та якісних метрик результатів генерації тестової документації в експерименті №2



14

ПОРІВНЯЛЬНА ДІАГРАМА РЕЗУЛЬТАТІВ РОБОТИ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ

Порівняння гібридного індексу якості, середнього значення кількісних та якісних метрик результатів генерації тестової документації в експерименті №3



15

ВИСНОВКИ

1. Проаналізовано предметну галузь. Існуючі інтелектуальні системи вирізняються значним різноманіттям, пропонуючи інструменти для великої кількості процесів тестування програмного забезпечення.

2. Проаналізовано сучасні методи оцінки згенерованої документації, такі як експертні, автоматичні та нейромережеві моделі. Визначено їхні ключові недоліки: висока собівартість, низька швидкість, нормативно-орієнтоване оцінювання, формалістична оцінка, можлива упередженість, можливі галюцинації системи

3. Визначено чотири кількісних та три якісних метрики та сформовано комплексну методику оцінювання якості генерації тест кейсів, що поєднує кількісні та якісні метрики. Запропонована методика включає оцінку точності, структури, стилю, та відповідності вимогам, що забезпечує прозорий і гнучкий механізм оцінювання інтелектуальних систем та дозволяє не лише кількісно оцінити рівень ефективності системи, а й визначити напрямки для її оптимізації.

4. Проведено експериментальне дослідження. Згідно експерименту №1 при однакових вхідних даних Google Gemini забезпечує вищу якість генерації тест кейсів. Після аналізу даних експерименту №1, було вирішено оптимізувати роботу інтелектуальних систем, додавши більш уточнюючі формулювання у промти. В експерименті №2 Google Gemini, OpenAI ChatGPT та Microsoft показують зростання оцінки гібридного індексу якості, робота систем була оптимізована на 2%, 12% та 12% відповідно (при $\alpha = 0.5$). Google Gemini показує зростання оцінки якісних метрик, а OpenAI ChatGPT та Microsoft Copilot мають значне зростання оцінки кількісних метрик. В експерименті №3 Google Gemini та OpenAI ChatGPT показують ще більше зростання оцінки гібридного індекс якості, робота систем була оптимізована на 18% та 11% відповідно (при $\alpha = 0.5$), за рахунок значного зростання оцінки кількісних та якісних метрик. В усіх трьох експериментах Google Gemini показує найбільшу оцінку гібридного індекс якості з трьох експериментів.

16

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Сазонов С.О., Яскевич В.О. Використання штучного інтелекту в життєвому циклі розробки програмного забезпечення.
VI Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT», 15 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 110-112.

2. Сазонов С.О., Яскевич В.О. Штучний інтелект у прогнозуванні ризиків та пріоритизації тестування програмного забезпечення. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ», 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 313-316.

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНОГО МОДУЛЯ

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <link href="./style.css" rel="stylesheet">
  <script src="./script.js"></script>
</head>
<body>
  <div class="subject">
    <p class="name_subject"
id="name_subject_One">Google Gemini</p>
    <div class="sub_metrics">
      <div class="quan_metrics">
        <div class="single_quan_metric">
          <p>T</p>
          <div class="metric_value">
            <p>t = </p>
            <input id="tOne">
          </div>
          <div class="metric_value">
            <p>N = </p>
            <input id="NOne">
          </div>
        </div>
        <div class="single_quan_metric">
          <p>RC</p>
          <div class="metric_value">
            <p>Ncov = </p>
            <input id="NcovOne">
          </div>
          <div class="metric_value">
            <p>Ntotal = </p>
            <input id="NtotalOne">
          </div>
        </div>
        <div class="single_quan_metric">
          <p>TCC</p>
          <div class="metric_value">
            <p>Nattr_present = </p>
            <input
id="Nattr_presentOne">
          </div>
          <div class="metric_value">
            <p>Nattr_required = </p>
            <input
id="Nattr_requiredOne">
          </div>
        </div>
        <div class="single_quan_metric">
          <p>ALS</p>
          <div class="metric_value">
            <p>Nstep = </p>
            <input id="NstepOne">
          </div>
          <div class="metric_value">
            <p>w = </p>
            <input id="wOne">
          </div>
        </div>
      </div>
      <div class="qual_metrics">
        <div class="single_qual_metric">
          <p>Зміст кейсів</p>
          <div class="metric_value">
            <p>1 = </p>
            <input id="zm_first_One">
          </div>
          <div class="metric_value">
            <p>2 = </p>
            <input id="zm_second_One">
          </div>
          <div class="metric_value">
            <p>3 = </p>
            <input id="zm_third_One">
          </div>
        </div>
        <div class="single_qual_metric">
          <p>Документація</p>
          <div class="metric_value">
            <p>1 = </p>
            <input id="doc_first_One">
          </div>
          <div class="metric_value">
            <p>2 = </p>
            <input id="doc_second_One">
          </div>
          <div class="metric_value">
            <p>3 = </p>
            <input id="doc_third_One">
          </div>
        </div>
        <div class="single_qual_metric">
          <p>Якість результатів</p>
          <div class="metric_value">
            <p>1 = </p>
            <input id="yak_first_One">
          </div>
          <div class="metric_value">
            <p>2 = </p>
            <input id="yak_second_One">
          </div>
          <div class="metric_value">
            <p>3 = </p>
            <input id="yak_third_One">
          </div>
        </div>
      </div>
    </div>
  </div>
  <div class="subject">
    <p class="name_subject"
id="name_subject_Two">OpenAI ChatGPT</p>
    <div class="sub_metrics">
      <div class="quan_metrics">
        <div class="single_quan_metric">
```

```
                        <p>T</p>
                        <div class="metric_value">
                            <p>t = </p>
                            <input id="tTwo">
                        </div>
                        <div class="metric_value">
                            <p>N = </p>
                            <input id="NTwo">
                        </div>
                    </div>
                    <div class="single_quan_metric">
                        <p>RC</p>
                        <div class="metric_value">
                            <p>Ncov = </p>
                            <input id="NcovTwo">
                        </div>
                        <div class="metric_value">
                            <p>Ntotal = </p>
                            <input id="NtotalTwo">
                        </div>
                    </div>
                    <div class="single_quan_metric">
                        <p>TCC</p>
                        <div class="metric_value">
                            <p>Nattr_present = </p>
                            <input
id="Nattr_presentTwo">
                        </div>
                        <div class="metric_value">
                            <p>Nattr_required = </p>
                            <input
id="Nattr_requiredTwo">
                        </div>
                    </div>
                    <div class="single_quan_metric">
                        <p>ALS</p>
                        <div class="metric_value">
                            <p>Nstep = </p>
                            <input id="NstepTwo">
                        </div>
                        <div class="metric_value">
                            <p>w = </p>
                            <input id="wTwo">
                        </div>
                    </div>
                </div>
                <div class="qual_metrics">
                    <div class="single_qual_metric">
                        <p>Зміст кейсів</p>
                        <div class="metric_value">
                            <p>1 = </p>
                            <input id="zm_first_Two">
                        </div>
                        <div class="metric_value">
                            <p>2 = </p>
                            <input id="zm_second_Two">
                        </div>
                        <div class="metric_value">
                            <p>3 = </p>
                            <input id="zm_third_Two">
                        </div>
                    </div>
                    <div class="single_qual_metric">
                        <p>Документація</p>
                        <div class="metric_value">
                            <p>1 = </p>
                            <input id="doc_first_Two">
                        </div>
                        <div class="metric_value">
                            <p>2 = </p>
                            <input id="doc_second_Two">
                        </div>
                        <div class="metric_value">
                            <p>3 = </p>
                            <input id="doc_third_Two">
                        </div>
                    </div>
                    <div class="single_qual_metric">
                        <p>Якість результатів</p>
                        <div class="metric_value">
                            <p>1 = </p>
                            <input id="yak_first_Two">
                        </div>
                        <div class="metric_value">
                            <p>2 = </p>
                            <input id="yak_second_Two">
                        </div>
                        <div class="metric_value">
                            <p>3 = </p>
                            <input id="yak_third_Two">
                        </div>
                    </div>
                </div>
            </div>
            <div class="subject">
                <p class="name_subject"
id="name_subject_Three">Microsoft Copilot</p>
                <div class="sub_metrics">
                    <div class="quan_metrics">
                        <div class="single_quan_metric">
                            <p>T</p>
                            <div class="metric_value">
                                <p>t = </p>
                                <input id="tThree">
                            </div>
                            <div class="metric_value">
                                <p>N = </p>
                                <input id="NThree">
                            </div>
                        </div>
                        <div class="single_quan_metric">
                            <p>RC</p>
                            <div class="metric_value">
                                <p>Ncov = </p>
                                <input id="NcovThree">
                            </div>
                            <div class="metric_value">
                                <p>Ntotal = </p>
                                <input id="NtotalThree">
                            </div>
                        </div>
```

```
                    <div class="single_quan_metric">
                        <p>TCC</p>
                        <div class="metric_value">
                            <p>Nattr_present = </p>
                            <input
id="Nattr_presentThree">
                        </div>
                        <div class="metric_value">
                            <p>Nattr_required = </p>
                            <input
id="Nattr_requiredThree">
                        </div>
                    </div>
                    <div class="single_quan_metric">
                        <p>ALS</p>
                        <div class="metric_value">
                            <p>Nstep = </p>
                            <input id="NstepThree">
                        </div>
                        <div class="metric_value">
                            <p>w = </p>
                            <input id="wThree">
                        </div>
                    </div>
                </div>
                <div class="qual_metrics">
                    <div class="single_qual_metric">
                        <p>Зміст кейсів</p>
                        <div class="metric_value">
                            <p>1 = </p>
                            <input id="zm_first_Three">
                        </div>
                        <div class="metric_value">
                            <p>2 = </p>
                            <input
id="zm_second_Three">
                        </div>
                        <div class="metric_value">
                            <p>3 = </p>
                            <input id="zm_third_Three">
                        </div>
                    </div>
                    <div class="single_qual_metric">
                        <p>Документація</p>
                        <div class="metric_value">
                            <p>1 = </p>
                            <input id="doc_first_Three">
                        </div>
                        <div class="metric_value">
                            <p>2 = </p>
                            <input
id="doc_second_Three">
                        </div>
                        <div class="metric_value">
                            <p>3 = </p>
                            <input id="doc_third_Three">
                        </div>
                    </div>
                    <div class="single_qual_metric">
                        <p>Якість результатів</p>
                        <div class="metric_value">
                            <p>1 = </p>
                            <input id="yak_first_Three">
                        </div>
                        <div class="metric_value">
                            <p>2 = </p>
                            <input
id="yak_second_Three">
                        </div>
                        <div class="metric_value">
                            <p>3 = </p>
                            <input id="yak_third_Three">
                        </div>
                    </div>
                </div>
            </div>
            <div class="alpha_value">
                <p>a =</p>
                <input id="alpha">
            </div>
            <button class="calculate"
onclick="calculate()" >Calculate</button>
            <div class="res_table">
                <table>
                    <thead>
                        <tr>
                            <th>Інтелектуальна
система</th>
                            <th>T</th>
                            <th>RC</th>
                            <th>TCC</th>
                            <th>ALS</th>
                            <th>Зміст кейсів</th>
                            <th>Документація</th>
                            <th>Якість результатів</th>
                            <th>HQI</th>
                        </tr>
                    </thead>
                    <tbody id="tbody">
                    </tbody>
                </table>
            </div>
        </body>
        </html>

        *{
            margin: 0px;
            padding: 0px;
            font-family: 'Roboto';
        }

        body{
            margin-left: 50px;
        }

        table{
            margin-bottom: 100px;
        }

        .subject{
            margin: 50px 100px 20px 0px;
```

```
        }
        .name_subject{
            margin-bottom: 10px;
        }

        button{
            margin: 50px;
            padding: 10px 40px;
        }

        .metric_value{
            margin-top: 10px;
        }

        input{
            width: 30px;
            height: 20px;
        }

        .quan_metrics, .qual_metrics {
            display: flex;
            flex-direction: row;
        }
        .single_quan_metric, .single_qual_metric{
            margin-right: 30px;
        }
        .sub_metrics{
            display: flex;
            flex-direction: row;
        }

        table, th, td{
            border: 1px solid black;
            border-collapse: collapse;
        }


        th, td{
            padding: 10px;
            text-align: left;
        }

        .alpha_value{
            margin-top: 10px;
        }

        function calculate() {
            // One dif
            let tOne =
parseFloat(document.getElementById("tOne").value);
            let NOne =
parseFloat(document.getElementById("NOne").value);

            let NcovOne =
parseFloat(document.getElementById("NcovOne").val
ue);
            let NtotalOne =
parseFloat(document.getElementById("NtotalOne").va
lue);
            let Nattr_presentOne =
parseFloat(document.getElementById("Nattr_presentO
ne").value);
            let Nattr_requiredOne =
parseFloat(document.getElementById("Nattr_required
One").value);

            let NstepOne =
parseFloat(document.getElementById("NstepOne").val
ue);
            let wOne =
parseFloat(document.getElementById("wOne").value);

            let zm_first_One =
parseFloat(document.getElementById("zm_first_One")
.value);
            let zm_second_One =
parseFloat(document.getElementById("zm_second_On
e").value);
            let zm_third_One =
parseFloat(document.getElementById("zm_third_One"
).value);

            let doc_first_One =
parseFloat(document.getElementById("doc_first_One"
).value);
            let doc_second_One =
parseFloat(document.getElementById("doc_second_O
ne").value);
            let doc_third_One =
parseFloat(document.getElementById("doc_third_One
").value);


            let yak_first_One =
parseFloat(document.getElementById("yak_first_One"
).value);
            let yak_second_One =
parseFloat(document.getElementById("yak_second_O
ne").value);
            let yak_third_One =
parseFloat(document.getElementById("yak_third_One
").value);

            let T_One = tOne/NOne;
            let RC_One = NcovOne / NtotalOne;
            let TCC_One =
Nattr_presentOne/Nattr_requiredOne;
            let ALS_One = wOne/NstepOne;

            let zm_One = zm_first_One +
zm_second_One + zm_third_One;
            let doc_One = doc_first_One +
doc_second_One + doc_third_One;
            let yak_One = yak_first_One +
yak_second_One + yak_third_One;

            // Two dif

            let tTwo =
parseFloat(document.getElementById("tTwo").value);
```

```
        let NTwo =
parseFloat(document.getElementById("NTwo").value);

        let NcovTwo =
parseFloat(document.getElementById("NcovTwo").val
ue);
        let NtotalTwo =
parseFloat(document.getElementById("NtotalTwo").va
lue);

        let Nattr_presentTwo =
parseFloat(document.getElementById("Nattr_presentT
wo").value);
        let Nattr_requiredTwo =
parseFloat(document.getElementById("Nattr_required
Two").value);

        let NstepTwo =
parseFloat(document.getElementById("NstepTwo").va
lue);
        let wTwo =
parseFloat(document.getElementById("wTwo").value);

        let zm_first_Two =
parseFloat(document.getElementById("zm_first_Two"
).value);
        let zm_second_Two =
parseFloat(document.getElementById("zm_second_Tw
o").value);
        let zm_third_Two =
parseFloat(document.getElementById("zm_third_Two"
).value);

        let doc_first_Two =
parseFloat(document.getElementById("doc_first_Two"
).value);
        let doc_second_Two =
parseFloat(document.getElementById("doc_second_T
wo").value);
        let doc_third_Two =
parseFloat(document.getElementById("doc_third_Two
").value);


        let yak_first_Two =
parseFloat(document.getElementById("yak_first_Two"
).value);
        let yak_second_Two =
parseFloat(document.getElementById("yak_second_T
wo").value);
        let yak_third_Two =
parseFloat(document.getElementById("yak_third_Two
").value);


        let T_Two = tTwo/NTwo;
        let RC_Two = NcovTwo / NtotalTwo;
        let TCC_Two =
Nattr_presentTwo/Nattr_requiredTwo;
        let ALS_Two = wTwo/NstepTwo;

        let zm_Two = zm_first_Two +
zm_second_Two + zm_third_Two;
        let doc_Two = doc_first_Two +
doc_second_Two + doc_third_Two;
        let yak_Two = yak_first_Two +
yak_second_Two + yak_third_Two;

        // Three dif

        let tThree =
parseFloat(document.getElementById("tThree").value)
;
        let NThree =
parseFloat(document.getElementById("NThree").value
);

        let NcovThree =
parseFloat(document.getElementById("NcovThree").v
alue);
        let NtotalThree =
parseFloat(document.getElementById("NtotalThree").
value);

        let Nattr_presentThree =
parseFloat(document.getElementById("Nattr_presentT
hree").value);
        let Nattr_requiredThree =
parseFloat(document.getElementById("Nattr_required
Three").value);

        let NstepThree =
parseFloat(document.getElementById("NstepThree").v
alue);
        let wThree =
parseFloat(document.getElementById("wThree").value
);

        let zm_first_Three =
parseFloat(document.getElementById("zm_first_Three
").value);
        let zm_second_Three =
parseFloat(document.getElementById("zm_second_Th
ree").value);
        let zm_third_Three =
parseFloat(document.getElementById("zm_third_Thre
e").value);

        let doc_first_Three =
parseFloat(document.getElementById("doc_first_Thre
e").value);
        let doc_second_Three =
parseFloat(document.getElementById("doc_second_T
hree").value);
        let doc_third_Three =
parseFloat(document.getElementById("doc_third_Two
").value);


        let yak_first_Three =
parseFloat(document.getElementById("yak_first_Thre
e").value);
```

```
        let yak_second_Three =
parseFloat(document.getElementById("yak_second_T
hree").value);
        let yak_third_Three =
parseFloat(document.getElementById("yak_third_Thre
e").value);


        // Metric calc

        let T_Three = tThree/NThree;
        let RC_Three = NcovThree / NtotalThree;
        let TCC_Three =
Nattr_presentThree/Nattr_requiredThree;
        let ALS_Three = wThree/NstepThree;

        let zm_Three = zm_first_Three +
zm_second_Three + zm_third_Three;
        let doc_Three = doc_first_Three +
doc_second_Three + doc_third_Three;
        let yak_Three = yak_first_Three +
yak_second_Three + yak_third_Three;

        // Normalisation

        T_arr = [T_One, T_Two, T_Three];

        T_One_norm = (Math.max(...T_arr) -
T_One)/(Math.max(...T_arr) - Math.min(...T_arr));
        T_Two_norm = (Math.max(...T_arr) -
T_Two)/(Math.max(...T_arr) - Math.min(...T_arr));
        T_Three_norm = (Math.max(...T_arr) -
T_Three)/(Math.max(...T_arr) - Math.min(...T_arr));


        ALS_arr = [ALS_One, ALS_Two,
ALS_Three];

        ALS_One_norm = (Math.max(...ALS_arr) -
ALS_One)/(Math.max(...ALS_arr) -
Math.min(...ALS_arr));
        ALS_Two_norm = (Math.max(...ALS_arr)
- ALS_Two)/(Math.max(...ALS_arr) -
Math.min(...ALS_arr));
        ALS_Three_norm =
(Math.max(...ALS_arr) -
ALS_Three)/(Math.max(...ALS_arr) -
Math.min(...ALS_arr));

        zm_One_norm = (zm_One - 3)/(15 - 3);
        zm_Two_norm = (zm_Two -3)/(15 - 3);
        zm_Three_norm = (zm_Three - 3)/(15 - 3);

        doc_One_norm = (doc_One - 3)/(15 - 3);
        doc_Two_norm = (doc_Two - 3)/(15 - 3);
        doc_Three_norm = (doc_Three - 3)/(15 -
3);

        yak_One_norm = (yak_One - 3)/(15 - 3);
        yak_Two_norm = (yak_Two - 3)/(15 - 3);
        yak_Three_norm = (yak_Three - 3)/(15 -
3);

        // HQI calc

        const alpha =
parseFloat(document.getElementById("alpha").value);

        let one_norm_quan_arr = [T_One_norm ,
RC_One , TCC_One , ALS_One_norm];
        let two_norm_quan_arr = [T_Two_norm ,
RC_Two , TCC_Two , ALS_Two_norm];
        let three_norm_quan_arr = [T_Three_norm
, RC_Three , TCC_Three , ALS_Three_norm];

        let one_norm_qual_arr = [zm_One_norm ,
doc_One_norm , yak_One_norm];
        let two_norm_qual_arr = [zm_Two_norm ,
doc_Two_norm , yak_Two_norm];
        let three_norm_qual_arr =
[zm_Three_norm , doc_Three_norm ,
yak_Three_norm];

        let M_auto_One = 0;
        for (let i of one_norm_quan_arr) {
          M_auto_One = M_auto_One + i;
        }
        let Qauto_One =
(1/one_norm_quan_arr.length)*M_auto_One;
        console.log("Qauto_One = ", Qauto_One)

        let E_exp_One = 0;
        for (let i of one_norm_qual_arr) {
          E_exp_One = E_exp_One + i;
        }
        let Qexp_One =
(1/one_norm_qual_arr.length)*E_exp_One;
        console.log("Qexp_One = ", Qexp_One)

        let M_auto_Two = 0;
        for (let i of two_norm_quan_arr) {
          M_auto_Two = M_auto_Two + i;
        }
        let Qauto_Two =
(1/two_norm_quan_arr.length)*M_auto_Two;
        console.log("Qauto_Two = ", Qauto_Two)

        let E_exp_Two = 0;
        for (let i of two_norm_qual_arr) {
          E_exp_Two = E_exp_Two + i;
        }
        let Qexp_Two =
(1/two_norm_qual_arr.length)*E_exp_Two;
        console.log("Qexp_Two = ", Qexp_Two)

        let M_auto_Three = 0;
        for (let i of three_norm_quan_arr) {
          M_auto_Three = M_auto_Three + i;
        }
        let Qauto_Three =
(1/three_norm_quan_arr.length)*M_auto_Three;
```

```
            console.log("Qauto_Three = ",
Qauto_Three)

            let E_exp_Three = 0;
            for (let i of three_norm_qual_arr) {
               E_exp_Three = E_exp_Three + i;
            }
            let Qexp_Three =
(1/three_norm_qual_arr.length)*E_exp_Three;
            console.log("Qexp_Three = ", Qexp_Three)

            let HQI_One =
(alpha*Qauto_One)+((1-alpha)*Qexp_One);
            let HQI_Two =
(alpha*Qauto_Two)+((1-alpha)*Qexp_Two);
            let HQI_Three =
(alpha*Qauto_Three)+((1-alpha)*Qexp_Three);

            //Table forming
            const tableBody =
document.getElementById("tbody");

            let newCell;

            rowOne = tableBody.insertRow();
            newCell = rowOne.insertCell().textContent
=
document.getElementById("name_subject_One").inner
Text;
            for (let i of one_norm_quan_arr) {
               newCell =
rowOne.insertCell().textContent = i.toFixed(2);
            }
            for (let i of one_norm_qual_arr) {
               newCell =
rowOne.insertCell().textContent = i.toFixed(2);
            }
            newCell = rowOne.insertCell().textContent
= HQI_One.toFixed(2);

            rowTwo = tableBody.insertRow();
            newCell = rowTwo.insertCell().textContent
=
document.getElementById("name_subject_Two").inne
rText;

            for (let i of two_norm_quan_arr) {
               newCell =
rowTwo.insertCell().textContent = i.toFixed(2);
            }
            for (let i of two_norm_qual_arr) {
               newCell =
rowTwo.insertCell().textContent = i.toFixed(2);
            }
            newCell = rowTwo.insertCell().textContent
= HQI_Two.toFixed(2);

            rowThree = tableBody.insertRow();
            newCell =
rowThree.insertCell().textContent =
document.getElementById("name_subject_Three").inn
erText;
            for (let i of three_norm_quan_arr) {
               newCell =
rowThree.insertCell().textContent = i.toFixed(2);
            }
            for (let i of three_norm_qual_arr) {
               newCell =
rowThree.insertCell().textContent = i.toFixed(2);
            }
            newCell =
rowThree.insertCell().textContent =
HQI_Three.toFixed(2);


         }
```